Combinational Techniques for Sequential Equivalence Checking

Hamid Savoj[1] **David Berthelot**[1] **Alan Mishchenko**[2] **Robert Brayton**[2]

Envis Corporation[1] and Department of EECS, University of California, Berkeley[2]

{hamid, david}@envis.com and {alanmi, brayton}@eecs.berkeley.edu

Abstract

Often sequential logic synthesis can lead to substantially easier verification problems, compared to the general-case for sequential equivalence checking (SEC). We prove that for a class of sequential transforms based on finite unrolling of a sequential design, SEC can be reduced to combinational equivalence checking (CEC). This class includes many sequential clock gating techniques. A method based on this was applied to large industrial examples, which are problematic for a conventional SEC engine, but sequential equivalence was reduced to less than a minute in most cases with the new approach.

1 Introduction

Consider a sequential circuit, A, which is to be optimized by a k-step unrolling process; combinational synthesis is applied to the first frame while the other k-1 frames are to be left untouched. This synthesis is done so that no difference between the two circuits is observed i.e. neither at the POs of each of the k frames nor at the flip-flop (FF) inputs of the final frame. The last k-1 copies of A are used only to produce ODCs for transforming the combinational part of A into the combinational part of a new sequential circuit B.

Several questions arise in similar types of synthesis:

1. Is the derived circuit B *sequentially* equivalent to A? This is not obvious because it is the k - 1 copies of A that provide the ODCs for B, and not B producing those ODCs as would be the case for sequential operation of machine B. Although there is a known 1-1 correspondence between the flip-flops in A and B, their state-transition functions are not necessarily the same.
2. Suppose A is unrolled n times and the last copy of A is synthesized using satisfiability don't cares (SDCs) provided by the first n - 1 copies of A. Are A and B sequentially equivalent?
3. More generally, suppose A is unrolled n + k times and the nth copy of A is synthesized, using both ODCs and SDCs, to produce B. Is the derived machine B sequentially equivalent to A?

In Section 2, we answer these three questions, affirmatively for the first two with Theorems 1 and 2, but we give a counterexample for the last question. Theorem 1 can be expected to apply generally when a synthesis transform can be argued from non-observability principles and Theorem 2 when non-controllability is used. In Section 3, we discuss relevant literature and parallels to the results obtained in this paper, and in Section 4, we give some experimental results illustrating how these methods can make sequential equivalence checking (SEC) much more effective and practical on certain types of problems. Section 5 summarizes and poses some open questions for future research.

2 Sequential Equivalence

The first question concerns equivalent behaviors of two related *combinational* circuits, A^k (A unrolled k times) and (B, A^{k-1}) (circuit B feeding into A unrolled k-1 times)[1]. The two circuits should agree combinationally, so that all the POs at each time-frame and the FF inputs at the end of frame k should agree. We say that (B, A^{k-1}) and A^k are combinationally equivalent and denote this by $(B, A^{k-1}) = A^k$.[2] This is depicted in Figure 1 where k = 3. The PIs of these related combinational circuits are the regular PI inputs at each frame and the FF outputs at the beginning of the first frame. The POs being compared are the regular POs at each frame and the FF inputs at the end of the last frame. Note that to create the related combinational circuit (B, A^{k-1}) from A and B, it is necessary that there is a 1-1 correspondence between the FFs of A and B.

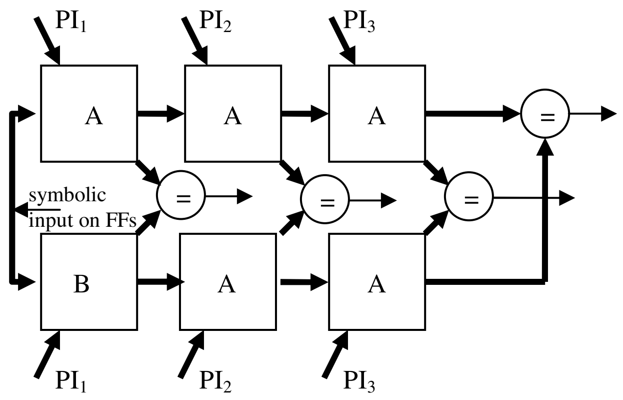

Figure 1. Unrolled sequential equivalence checking.

More generally, assume that two sequential circuits, A and B, are given where their FFs can be matched one-to-one across the two circuits.

Theorem 1: $[(B^n, A^k) = A^{n+k}] \Rightarrow A = B$ *for any initial state.*

Proof: Assume that $(B^n, A^k) = A^{n+k}$. Consider the following infinite sequence of lines.

[1] Although we should denote the combinational part of a sequential circuit, A, by a different notation, say A_c, for simplicity we choose to overload notation by using A to denote both circuits.

[2] We overload the equal sign, so that if there is only one copy of each machine, e.g. $A = B$, this means sequential equivalence.

$\rightarrow$ frame number $\rightarrow$

$$\begin{tikzcd}
{} \arrow[r, "S^*"] & A^n \arrow[r, "S_0(n)"] & A^n \arrow[r, "S_0(2n)"] & A^n \cdots \\
{} \arrow[r, "S^*"] & B^n \arrow[r, "S_1(n)"] & A^n \arrow[r, "S_1(2n)"] & A^n \arrow[r, "S_1(3n)"] & \cdots \\
{} \arrow[r, "S^*"] & B^n \arrow[r, "S_2(n)"] & B^n \arrow[r, "S_2(2n)"] & A^n \arrow[r, "S_2(3n)"] & \cdots \\
 & & \vdots & & \\
{} \arrow[r, "S^*"] & B^n \arrow[r, "S_j(n)"] & B^n \arrow[r, "S_j(2n)"] & B^n \arrow[r, "S_j(3n)"] & B^n \cdots
\end{tikzcd}$$

It is assumed that all lines at each time-frame receive the same sequence of common PI inputs. Denote the POs of line *j* by $PO_j(t)$, where $t \in \{1,2,3,\cdots\}$ and $j \in \{0,1,2,3,\cdots\}$. Similarly for the states; $S_j(t)$ denotes the state of line *j* at time *t*, $t \in \{0,1,2,3,\cdots\}$ where $S_j(0) = S^*$, the set of all states. Since $(B^n, A^k) = A^{n+k}$, then $PO_0(t) = PO_1(t)$ for $t \in \{1,2,3,\cdots,n+k\}$. Note that for all $t > n+k$, this is also true because $S_0(n+k) = S_1(n+k)$ and the copies in both lines are *A* from then on. Now compare lines 1 and 2. Clearly $PO_1(t) = PO_2(t), t = 1,...,n$ since in both lines the inputs are to *n* copies of *B*, and by using the template, $(B^n, A^k) = A^{n+k}$, but applying it starting at the end of frame *n*, we have $PO_1(t) = PO_2(t)$ for all *t*, by the same argument that established that $PO_0(t) = PO_1(t)$ for all *t*. Thus by transitivity, $PO_0(t) = PO_2(t)$. Continuing, we get $PO_0(t) = PO_j(t)$ for all lines $j \in \{1,2,3,\cdots\}$. Thus line 0 and line ∞ always produce the same sequence of POs for all time no matter what is the initial state. Since the miter for $A \oplus B$ is proven to be UNSAT, we have $A = B$. **QED**.

Note that nothing is assumed about if or how *B* might have been obtained from *A*. Also, the implication is only one way, so that if $(B^n, A^k) \neq A^{n+k}$**,** it does not mean that *B* is sequentially inequivalent to *A*. In such a case when the premise fails, one can try to strengthen it by increasing *k* or *n*, and a false negative may go away.

Note also that no initial state information was used in proving this theorem, i.e. $S_j(0) = S^*$ is the set of all states. However, we could use a subset $\hat{S} \subset S^*$ as long it is guaranteed that $S_0(n), S_1(2n), S_2(3n), \cdots$ are all **subsets** of $\hat{S}$. Thus a corollary of the theorem would be that $[(B^n, A^k) = A^{n+k}]_{\hat{S}} \Rightarrow A = B$ for any state $s \in \hat{S}$, where $[(B^n, A^k) = A^{n+k}]_{\hat{S}}$ denotes that combinational equivalence need only hold on state inputs in $\hat{S}$.

A companion of Theorem 1 is the following which states the sequential equivalence of *A* and *B* after *n* cycles of *A*.

Theorem 2: $[(A^n, B^k) = A^{n+k}] \Rightarrow A = B$ *on the subset of states that can be reached by A after n cycles, denoted* S_n^A.

Proof: We use that fact that the state space is finite, and therefore its diameter, *D*, is bounded. Thus after *D* time frames, every possible state has been seen under all possible inputs. The proof is similar to that of Theorem 1, except we proceed backwards from time-frame $T = D + (k\lceil D \div k \rceil)$. We first apply the template, $(A^n, B^k) = A^{n+k}$, to insert *k B*'s just before $t = T$. This is iterated $\lceil D \div k \rceil$ times until we arrive at a line with *n A*'s followed by all *B*'s up to *t* = *T*. At each iteration *j*, we are assured that $PO_1(t) = PO_j(t)$ for all time. At this point we know that all states and all PIs for these states have been seen and for all of these the PO's agree. Thus starting at the states S_n^A, $A = B$. **QED.**

To illustrate the need to start only on the states reachable by an initial sequence of *A*'s, consider the example Figure 2.

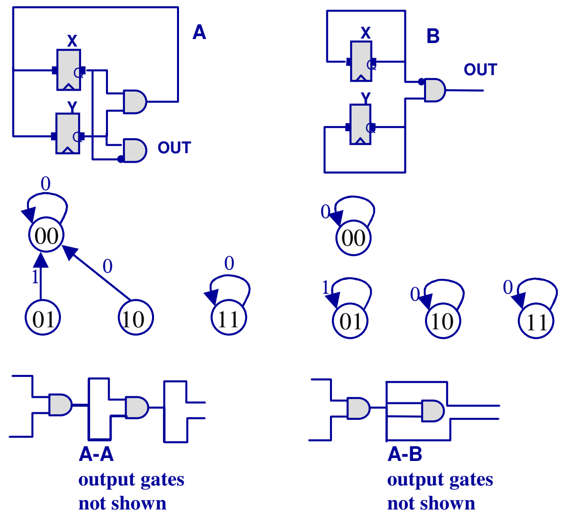

Figure 2. $A, B = A^2$, but sequential equivalence occurs only after one cycle of *A*.

One can check easily that (*A*,*A*)=(*A*,*B*) from the STGs shown, but it is not true that *A*=*B* sequentially. The counterexample is that if *A* and *B* start in State 01, the PO sequences for *A* and *B* are not the same. However, note that on the states that can be reached after one clock cycle of *A* (i.e. States 00 and 11), then *A* and *B* are sequentially equivalent.

The first theorem is an observability theorem and the second a controllability theorem. One might conjecture that an analgous combined controllability and observability theorem also holds, but that is not the case.

To see this, consider the STG example shown in Figure 3, which has no inputs, and the label on the edges denotes the output value. Although $(A, A, A) = (A, B, A)$, one can check that $A \neq B$, even on the states that *A* can reach after one cycle. For example, start at State 01. The output sequence for *A* is 110… and for *B* it is 111….

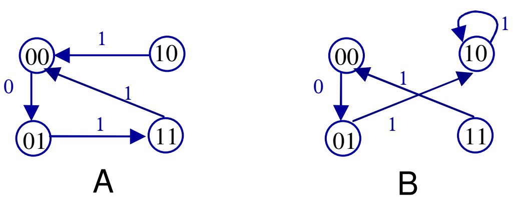

Figure 3. Although $(A, A, A) = (A, B, A)$ combinationally, *A* and *B* are not sequentially equivalent.

Even though such a theorem does not hold, we may still be able to easily verify equivalence where both observability and controllability are used in synthesis, e.g. as in [3]. It is possible, in the synthesis, that the don't cares actually used would be produced also by *B* as well as *A*. For example, instead of trying to verify $A^3 = (A, B, A)$, we can try $A^3 = (B^2, A)$ or $A^3 = (A, B^2)$

using Theorems 1 or 2. If either case works, we have proved sequential equivalence more easily.

Another possibility is given in Theorem 3.

Theorem 3: $[(B, A^2) = (B^2, A)] \Rightarrow A = B$ *on the subset of states that can be reached by B after one cycle, denoted* S_1^B.

Proof: Consider the sequence of transformations shown below.

$$\begin{array}{l} B, A, A, A, A, A, \cdots \\ B, B, A, A, A, A, \cdots \\ B, B, B, A, A, A, \cdots \\ B, B, B, B, A, A, \cdots \\ \cdots \end{array}$$

Each new line is obtained by using $(B, A^2) = (B^2, A)$. At each line note that $P_j(t) = P_0(t), \forall t$. Thus after the first time frame, the set of states that can exist is S_1^B and after that, we have

$$A, A, A, A, A, A, \cdots = B, B, B, B, B, B, \cdots$$

Thus, $A = B$ on S_1^B. **QED**

Note that for the example in Figure 2, $(B^2, A) \neq (B, A^2)$, which can be seen by starting at State (01), so it is not a counterexample to Theorem 3.

3 Relations to Previous Work

Synthesis transformations are either combination or sequential. Combinational ones extract the flip-flops and treat their inputs and outputs as additional POs and PIs respectively. Synthesis done on this combinational part obviously preserves sequential equivalence.

One of the pragmatic aspects of synthesis, often ignored, is that it is insufficient to provide synthesis software, which may even use formally-proved[3] transforms. These transformations are embodied in software, which may have bugs; it is very rare that a software program is proved formally. In addition, even if the software has withstood the test of time and was applied to many examples, many hardware designers and systems companies insist on formally verifying the result against the original design. Equivalence checking of combinational netlists (CEC) is doable for most industrial designs and because of this, combinational synthesis is readily accepted. Additional confidence is achieved by using CEC software that is provided by third parties. Also, resolution proofs [7], which exist for CEC[4], can be used.

However, the complexity of SEC, which is PSPACE- complete, often discourages the use of sequential synthesis. In special cases, the complexity of SEC is simpler, e.g. if synthesis is restricted to one set of combinational transformations followed by one retiming (a sequential synthesis step) or vice versa, the problem is provably simpler - *only* NP-complete. If retiming and resynthesis are iterated, the problem is again PSPACE complete [2]. Like CEC, SEC becomes simpler in practice if there are many structural or functional similarities (cut-points) between the two circuits being compared.

There are several known instances where SEC can be transformed into a CEC problem on which today's commercial CEC engines usually can be successful, even on very large problems. One is where *sequential signal* equivalences (signals that are equivalent on the reachable state set) are derived using induction [4] and used in the synthesis process. If equivalence checking is done after this without other transformations intervening, SEC can be proved by CEC methods.

Another example is when a history of synthesis is recorded by a sequence of circuit differences seen after incremental synthesis steps [5]. In most cases, this history provides a set of intermediate equivalences, which can be used to prove sequential equivalence using CEC methods. Also, the concept of speculative reduction [6] can be used to make the equivalence checking problem even easier in this case.

Several papers have used an (explicitly or implicitly) unrolled version of the circuit to derive redundancies for synthesizing an improved sequential circuit. These papers do not address the formal equivalence checking of the synthesized result. All deal with the case when a machine can power up in any state, so the redundancies derived are independent of any initial state. Most of these types of results come from the testing community, where the notion of the redundancy of a signal is stated in terms of a *stuck-at-u,* $u \in \{0,1\}$, fault. A signal is redundant if the good and faulty (fault inserted) machines can not be distinguished for any initial state after power-up.

There is a subtle distinction between untestable faults and redundant faults. If s_f is the initial power-up state of the faulty machine and *s* is the one for the good machine, then a fault is *untestable* if $\forall(I)\exists(s, s_f)[Z(I,s) = Z^f(I, s_f)]$ and it is *redundant* if $\forall(I, s_f)\exists(s)[Z(I,s) = Z^f(I, s_f)]$. $Z(I,s)$ is the trace of POs under the sequence *I* of PI inputs and starting at state *s*. Using redundancy in synthesis means that when the good machine is replaced with the "faulty" (redundancy removed) machine, no difference can be observed externally because no matter what state s_f the faulty machine powers up in, there is an equivalent state in which the good machine could have powered up. Such a replacement is *safe*[5] [8] and *compositional*. In contrast, if the fault is merely untestable, then there could exist at least one *pair* of states in which the two machines could power up, and difference between the two machines could not be detected (externally). However, there could be a state in either machine which has no match in the other and if one of the machines happened to power-up in such a state, the two machines would have different behaviors. Such a (untestable) replacement is not safe and is not compositional, and its use in synthesis is problematic. A good discussion on the difference between undetectable faults and redundant faults can be found in [2].

From Theorem 1, if $(B^n, A^k) = A^{n+k}$, then the synthesized circuit is a safe replacement for the original one. Safe replacements are useful because safety implies that every synchronization sequence for the original design also synchronizes the replacement. This is often desired because it is not necessary to re-derive a new synchronizing sequence for initializing the synthesized machine.

A useful notion is *c*-cycle redundancy [1] where the two circuits' outputs need not match for the first *c* cycles after power-up. This allows more flexibility in synthesizing a circuit because the behavior of the machine need only be preserved on states that can be reached after *c* cycles as long as initialization is preserved. Several papers make use of this and determine a bound *k* and a new circuit with the redundancies removed (called a *k*-delayed replacement) [3]. In [1] such redundancies are identified, one is

[3] There are cases where "proved" methods in the literature have been shown to have counterexamples.

[4] We know of no equivalent capability for sequential equivalence checking.

[5] A safe replacement is one for which there is no possibility of externally detecting any difference from the original.

then removed, and new ones identified. This is repeated until no more can be found. In [3], a set of "compatible" redundancies is found and removed simultaneously.

The method of [3] derives a constant n which is the difference between the time frame of an identified redundancy and the least time frame needed to infer this redundancy. Their theorem states that if the redundancy is used to create the new circuit, then it is a n-delayed replacement of the original. Note the difference between n-delayed replacement and Theorem 2. In n-delayed replacement, it is B that is delayed for n cycles before equivalence can be guaranteed, whereas in Theorem 2 it is A that is delayed n cycles.

A sequential ATPG engine can be used to determine if a test vector sequence can be found which justifies a state that activates the fault in n cycles and then propagates the fault effect to a PO in k cycles. If none can be found, the fault might be conjectured to be redundant. However, there are three things that can go wrong here; (i) undetectable faults are not necessarily redundant, (ii) the justification and propagation conditions are usually done on the good machine, and (iii) finite values for n and k were used. Nevertheless, such a fault is a good candidate for redundancy removal, but the result must be sequentially verified. Such a situation is a good candidate for applying Theorems 1 or 2, which may still work if A or B are supplying a sufficient set of SDCs or ODCs. A discussion of some incorrect "proofs" in the literature related to the use of ATPG for redundancy removal can be found in [2], as well as limitations on some other methods.

4 Experimental Results

Experimental results are shown in Table 1. Six large industrial benchmarks were synthesized using sequential clock-gating transforms, based on sequential observability don't care arguments. The synthesized versions are denoted by B and the originals by A. Columns 1-5 give information about the sizes of the circuits. The columns labeled *New* denote the use of the new method which uses Theorem 1 and ABC's CEC algorithm to prove SEC. The column *ABC general* denotes that a general SEC algorithm (ABC command *dsec*) was used. The columns *seq-j* denote experiments where (B, A^j) was compared combinationally with A^{j+1} to produce more difficult CEC problems to illustrate how CEC run-times might scale.

Note that, except those items marked with * or **, all equivalence checking problems succeeded. Some observations about the experiments are the following:

1. In general, *New* is significantly faster than *General* as expected.
2. It was surprising that *General* could actually complete on three out of six large problems and prove SEC.
3. Except for Design 4, CEC scales nicely as more copies of A are used in the CEC problem.

5 Conclusions and Future Work

Some sequential synthesis transforms do not use the initial state information but preserve a circuit's behavior starting from any initial state. Such transforms may use sequential observability [1] [3] and can be practical because they do not use state space search or can be argued using structural information as in the case of many clock-gating methods. These are in contrast to transforms which extract ODCs using exact BDD-based reachability analysis or SAT-based induction or interpolation to obtain an over-approximation to the set of reached states, such as in a recent work [4].

In the sequential observability case, it may be possible that sequential equivalence can be proved by combinational equivalence checking methods, making SEC much easier. This can have a significant impact in such applications where parts of the circuit are changed using a local view of the circuit.

In this paper, we have given a method for SEC, which can be effective in certain special cases, leading to considerable reduction in computation effort. The method is conservative, so when it fails, no information is obtained. We discussed some conditions under which it can be expected to succeed. These include some sequential clock-gating methods and methods that alter pipeline behavior. Experimental results were given on some large industrial applications, comparing a sequentially synthesized design against the original design, with the expected benefit of achieving very efficient SEC.

Our results are stated in terms of having a one-to-one correspondence between the FFs of A and B. This was so that combinational circuits (A,B) or (B,A) can be formed, which requires that signals in the first circuit to be wired to their corresponding signal in the second circuit. However, some transforms require that a signal be delayed one or more time frames. In such a case, FFs must be introduced in B that have no correspondence in A. This can be handled by introducing dummy FFs in A that have no function.

We conjecture that, more generally, it is sufficient to find two cuts of the same size, one in A and the other in B. The signals in the cuts can be a mixture of internal wires and FFs. It may be that the only requirement is that the cuts are feedback arc sets, i.e. cutting them makes each circuit acyclic. This would allow applications of the theorems to circuits where one of them has been retimed.

Also, it would be desirable to have a practical method to check general k-delayed equivalence, such as designs produced by the methods of [1][3]. These situations are cases of local sequential synthesis and it might be hypothesized that the associated SEC problems can be reduced to CEC problems. It is possible that Theorem 3 can be used in such cases, although at the moment, we have no experimental results on this.

Acknowledgements

This work was supported by SRC contract 1444.001, NSF grant CCF-0702668, and the (now-defunct) California Micro program.

References

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

[2] M. A. Iyer, D. E. Long, and M. Abramovici, ''Surprises in Sequential Redundancy Identification,'' *Proc. EDTC'96*.

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

[4] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *Proc. ICCAD'08*. http://www.eecs.berkeley.edu/~alanmi/publications/2008/iccad08_seq.pdf

[5] A. Mishchenko and R. K. Brayton, "Recording synthesis history for sequential verification", *Proc. FMCAD'08*, pp. 27-34. http://www.eecs.berkeley.edu/~alanmi/publications/2008/fmcad08_haig.pdf

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

[7] S. Chatterjee, A. Mishchenko, R. Brayton, and A. Kuehlmann. "On Resolution proofs for combinational equivalence", *Proc. DAC'07*.

[8] V Singhal and C. Pixley. "The verification problem for safe replaceability," *Proc. CAV'94*, LNCS, Vol. 818, pp. 311-323.

Table 1. Experimental results.

Design	Statistics				Seq-1		Seq-2	Seq-3	Seq-4	Seq-5
	Ands	Flops	PI	PO	New	General	New	New	New	New
1	39282	6506	51	83	0.68	15.16	0.98	1.34	1.55	1.78
2	18932	10544	96	115	0.51	18.88	0.7	0.88	1.06	1.25
3	31103	7276	105	79	0.78	*60.55	1.29	1.51	1.69	1.63
4	81782	13822	394	703	1.61	*152.21	2.34	17.22	72.93	267.83
5	45241	11595	1741	301	0.94	25.63	1.26	1.63	2.37	**6.18
6	114824	15284	857	804	2.05	*112.83	3.26	4.09	4.79	6.22

Notes: * General sequence equivalence in ABC timed out.
** Unresolved by ABC combinational equivalence checking
Entries in columns 6-11 denote run times in minutes.
Seq-*j* denotes the CEC problem where *j* copies of *A* are used, i.e. (B,A^j) is compared to A^{j+1}.