Retiming and Resynthesis with Sweep Are Complete for Sequential Transformation

Hai Zhou
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

***Abstract*— There is a long history of investigations and debates on whether a sequence of retiming and resynthesis is complete for all sequential transformations (on steady states). It has been shown that the sweep operation, which adds or removes registers not used by any output, is necessary for some sequential transformations. However, it is an open question whether retiming and resynthesis with sweep are complete. This paper proves that the operations are complete, but with one caveat: at least one resynthesis operation needs to look through the register boundary into the logic of previous cycle. We showed that this one-cycle reachability is required for retiming and resynthesis to be complete for re-encodings with different code length. This requirement comes from the fact that Boolean circuit is used for a discrete function thus its range needs to be computed by a traversal of the circuit. In theory, five operations in the order of sweep, resynthesis, retiming, resynthesis, and sweep are already complete. However, some practical limitations on resynthesis must be considered. The complexity of retiming and resynthesis verification is also discussed.**

I. Introduction

Logic synthesis algorithms originally targeted the optimization of PLA implementations; this was followed by research in synthesizing more general multilevel logic implementations. Currently, the central thrust in logic synthesis is sequential synthesis, i.e., the automatic optimization of the entire system. This is for designs specified at the structural level in the form of netlists, or at the behavioral level, i.e., in the form of finite state machines. DeMicheli [12] gives an excellent introduction to logic synthesis.

In this paper, we will be concerned with sequential designs. These can be specified at the behavioral level, as *finite state machines* (FSMs), or at the structural level, as *netlists* of *gates* and *registers*.

Retiming is a powerful sequential optimization step that can be applied to sequential designs described at the netlist level. It can be used to optimize the clock period or the registers area of a design. Logic synthesis is an operation that changes the circuit structure without changing the function of the combinational logic. It has been shown that given two designs, one of netlists has been derived from the other by a sequence of retiming and resynthesis, a certain equivalence relation (namely, steady-state equivalence) exists between them. However, the converse is not well understood, and there is a long history of investigations and debates on whether a sequence of retiming and resynthesis is complete for any sequentially equivalent transformation.

Malik [11] gave the first (partial) positive answer to this question. He proved that retiming and resynthesis are complete for any state re-encoding, and for some other transformations. Zhou et al. [18] provided the first negative answer by proving that some sequentially equivalent transformations cannot be done by retiming and resynthesis, which also helped to discover and fix an error in Malik's result [13]. The sweep operation, which adds or removes registers not used by any output, is needed for these transformations. However, it is an open question whether retiming and resynthesis with sweep are complete for general sequential transformations.

In this paper, we provide a complete answer to the open question. We proved that retiming and resynthesis with sweep are complete, but with one caveat: at least one resynthesis operation needs to look through the register boundary into the logic of previous cycle. We even showed that this one-cycle reachability is required for retiming and resynthesis to be complete for re-encodings with different code length, an extension to Malik et al. [10]. It also demonstrates that reachability information cannot be captured by these structural operations. Therefore, they are complete for transformations based on all steady states unless reachability information is provided. Our completeness proof is a constructive one that applies five operations in the order of sweep, resynthesis, retiming, resynthesis, and sweep. We will discuss the implications of such a result and some practical limitations on resynthesis.

Zhou et al. [18] also started an investigation on the complexity of retiming and resynthesis verification problem. Since the general sequential equivalence verification is PSPACE-complete, a different complexity category may indicate that the gap between retiming and resynthesis and sequential transformation is big. Jiang and Brayton [6] later showed that the complexity of retiming and resynthesis verification is also PSPACE-complete. We examine their proof and point out parts that are unclear. Based on those we consider the membership of retiming and resynthesis verification an open question.

Our results have very important practical implications. Since retiming and resynthesis with sweep are complete, sequential optimization tools can be centered around them. If any reachability information is provided to the optimization, it is also critical to be supplied to the verification. Our completeness proof also indicates that the resynthesis needs to generate exponential-size circuits to complete some transformations

This work is supported by the NSF under CNS-0613967 and CCR-0811270.

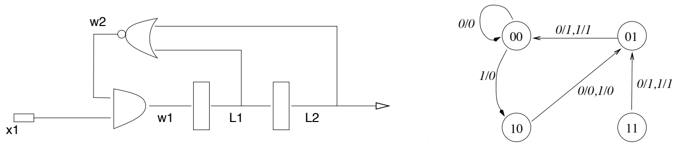

Fig. 1. Netlist and corresponding FSM.

(including some re-encoding ones). However, no practical resynthesis is so powerful. Under realistic limitations, retiming and resynthesis verification is much simpler. Indeed, the recent sequential equivalence checking algorithms [15], [17], [7] effectively try to show that two circuits are equivalent by deriving the retiming relationships between them.

II. Background

Netlists and FSMs: We introduce two formalisms for representing designs, namely netlists and finite state machines (FSMs). Netlists are structural and consist of an interconnection of gates and registers. Finite state machines are behavioral and specify how the system changes its states and produces outputs responding to inputs. We leave for the readers to ponder which representation is more abstract.

Before giving formal definitions to netlists and FSMs, we illustrate them by means of examples. The netlist in Figure 1 has one primary input, one primary output, two registers, and two gates. The FSM for the same design is shown beside it; it consists of 4 states.

Definition 1: A *Finite State Machine (FSM)* is a quintuple $(Q, I, O, \lambda, \delta)$ where Q is a finite set referred to as the *states*, I, and O are finite sets referred to as the set of *inputs* and *outputs* respectively, $\delta : Q \times I \to Q$ is the *next-state function*, and $\lambda : Q \times I \to O$ is the *output function.*

The output and next state functions can be inductively extended to the domains $Q \times I^+ \to O^+$ and $Q \times I^+ \to Q$, respectively; we continue to use λ and δ to denote these extended functions. For example, for the FSM in Figure 1, $\lambda(10, 1 \cdot 0) = 0 \cdot 1$ and $\delta(01, 0 \cdot 1 \cdot 0) = 01$.

Definition 2: A *netlist* is a directed graph, where the nodes correspond to elementary circuit elements, and the edges correspond to wires connecting these elements. Each node is labeled with a distinct variable w_i. For simplicity, we will assume that the netlist is *Boolean*, i.e. all variables take values in $B = \{0, 1\}$. The three basic circuit elements are *primary inputs, registers*, and *gates*. Primary input nodes have no fanins; registers have a single input. Associated with a gate g on n-inputs $w_1, w_2, \ldots, w_n$ is a function from B^n to B. Some nodes are designated as being *primary outputs*.

Given a value to each input and a state (an assignment of values to registers), one can uniquely compute the value of each node in the netlist by evaluating the functions at gates. A netlist η on inputs $i_1, i_2, \ldots, i_n$, outputs $o_1, o_2, \ldots, o_m$ and registers $r_1, r_2, \ldots, r_k$ bears a natural correspondence to an FSM M_η on inputs $X = B^n$, outputs $Y = B^m$, and state space $Q = B^k$. The next-state function of M_η is defined by the composed logic gates in the following manner: for each register r_i we can find a function $f_i : Q \times X \to B$ by composing the functions of the gates from the inputs and register outputs to the input of the register. We will refer to f_i as the next-state function of the register i. Then $\delta_{M_\eta} : Q \times X \to Q$ is simply $(f_1, f_2, \ldots, f_k)$. Similarly, the output function is defined by composing the functions of gates from inputs and registers to output nodes.

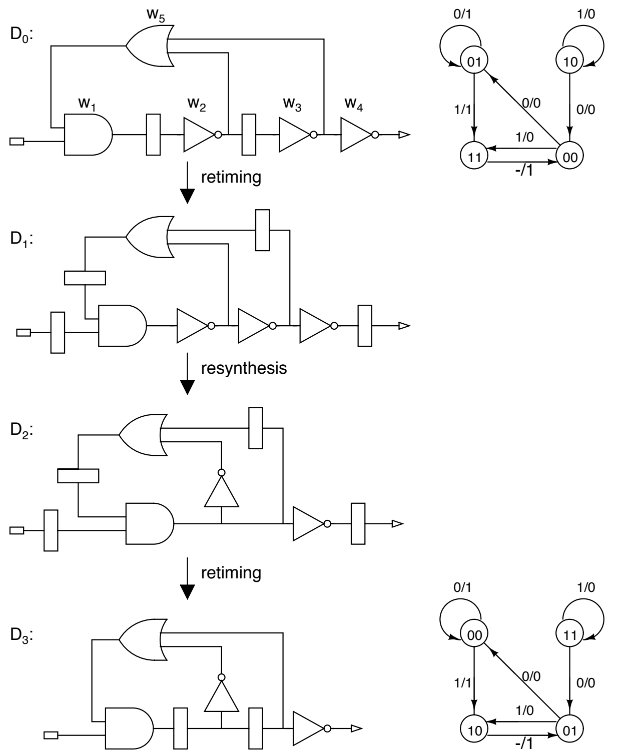

Fig. 2. Example of retiming and resynthesis.

Retiming, Resynthesis, and Sweep: Retiming, resynthesis, and sweep are structural operations applied on netlists.

Retiming consists of moving a given number of registers between the inputs and outputs of each combinational node. A retiming can be described mathematically by a lag function, which gives for each combinational node, the number of registers that are moved from each fanout to each fanin.

Combinational synthesis restructures the netlist within the register boundaries without changing its functionality. It leaves the FSM of the design unchanged. Retiming becomes very powerful when it is interspersed with resynthesis of the netlist within the changed register boundaries. Resynthesis provides new signals for retiming to place registers on while retiming provides new combinational blocks for resynthesis to manipulate. This is the basis for the retiming and resynthesis (RnR) paradigm proposed in [4], [10].

Sweep, the simplest among the operations, adds or removes registers not used by any output. Since synthesis normally simplifies the circuit structure, sweep is usually met as an operation removing redundant registers and logic.

An example of a design transformation using this RnR paradigm is shown in Figure 2. At the first retiming step, we have the following lag function: $r(w_1) = 1, r(w_3) = r(w_4) = -1, r(w_2) = r(w_5) = 0$.

Sequential Equivalence:

Definition 3: Two states s and t are *equivalent*, denoted as $s \cong t$, if and only if for every finite input sequence π, the outputs resulting on applying π are equal.
For example, in circuits D_0 and D_3 of Figure 2, state 00 in D_0 is equivalent to state 01 in D_3.

Definition 4: Two netlists C and D are *FSM-equivalent* if and only if every state $c \in C$ is equivalent to some state $d \in D$, and every state $d \in D$ is equivalent to some state $c \in C$.
Thus the two designs D_0 and D_3 in Figure 2 are FSM-equivalent.

Definition 5: The *steady state set* of a design D, denoted by D^∞ is the subset of states such that for each state s there is an input sequence π which drives this state to itself, i.e., $\lambda_D(s, \pi) = s$. The remaining set of states is called the *transient state set*.
For example, the steady state set for the design in Figure 1 is $\{00, 01, 10\}$ and the transient state set is $\{11\}$. Notice that once a design starts up in any state, it will eventually be and remain in steady states.

Theorem 1 ([8]): If design C has been obtained from design D by a sequence of retiming moves, the steady state set of C is FSM-equivalent to the steady state set of D.

Retiming becomes very powerful when it is combined with (combinational) resynthesis operations (the RnR paradigm). However, since resynthesis itself does not change the state transition graph of a design, we have the following corollary.

Corollary 1: If design C has been obtained from design D by a sequence of retiming and combinational resynthesis moves, the steady state set of C is FSM-equivalent to the steady state set of D.

Designated Initial State: We will not assume a designated initial state for our circuits. If we do want to force a circuits into a designated initial state we can explicitly model the reset circuitry along with the registers: indeed, this is the approach suggested for retiming initial states in [14], as opposed to the approach in [16], [5], where the implicit initial state values have to be retimed across gates.

One optimization advantage of considering designated initial states is that the synthesis algorithms have greater flexibility since the synthesis tool can potentially take advantage of don't cares arising from the set of states unreachable from the initial state. However, it is easy to show that for designs which have designated initial state, retiming and resynthesis is strictly weaker than a sequential optimization algorithm which uses unreachability don't cares (for example, [9]).

Consider circuits C and D in Figure 3 with 00 as the designated initial state for both C and D. Clearly the two circuits C and D are equivalent from the initial state 00. However, from Corollary 1, it is clear that C and D are not RnR equivalent (since state $10 \in C^\infty$, the steady state set of C, but there is no equivalent state in D^∞).

However, in general, commercial synthesis tools do not use unreachability don't cares. This is simply because computing the set of unreachable states is computationally very expensive on real designs; the theoretical complexity of this problem is PSPACE-complete:

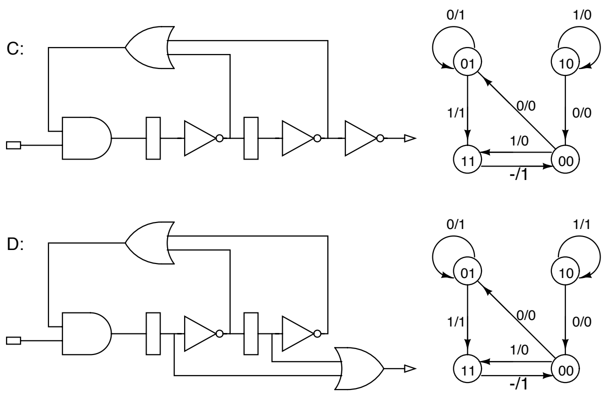

Fig. 3. Circuits C and D are equivalent from designated initial state 00.

Theorem 2 ([2]): Given two netlists C and D, and two states s from M_C, t from M_D, checking whether s and t are equivalent is PSPACE-complete in the size of the netlists.

III. Sweep is Necessary

Even though it is commonly suspected that retiming and resynthesis are not complete for all steady state equivalent transformations, Zhou et al. [18] was the first giving such a proof. They designed two pairs of circuits and proved that the first pair cannot be transformed to each other even though they are FSM-equivalent. The second pair was also conjectured so. We show here that both pairs are incomplete, using the same reasoning they used for the first pair. The two pairs of circuits are shown in Figure 4.

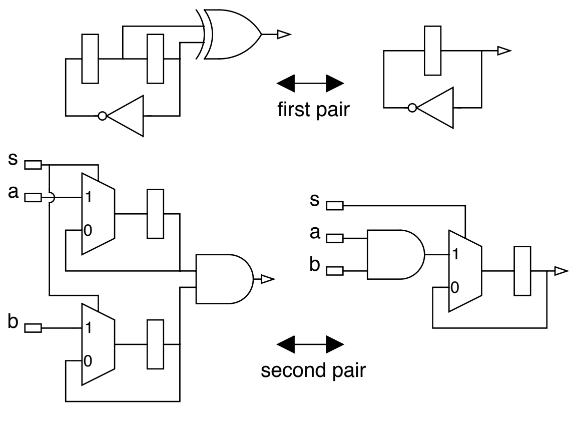

Fig. 4. Examples showing incompleteness of retiming and resynthesis.

Lemma 3: Retiming and resynthesis cannot transform one circuit to the other for either pair in Figure 4.

Proof: The next state function of the left circuit in each pair contains a permutation on the set $\{0, 1\}^2$, which has cardinality of 4. No matter what resynthesis does, the smallest cut size, in terms of the number of signals, on the combinational part must be at least 2, in order to encode all the information. Therefore, the next retiming step cannot reduce the number of registers. Since the next state function of the new circuit still has the property as the old one, any later retiming and resynthesis steps cannot reduce the number of

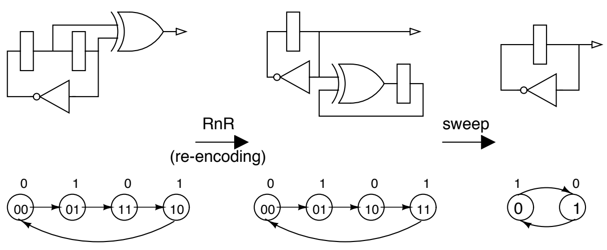

Fig. 5. Retiming and resynthesis are more powerful with sweep.

registers, either. This means that no sequence of retiming and resynthesis can transform the left circuits to the right ones. ■

However, it was also noted in [18] that with the sweep operation, the first pair of circuits are transformable to each other, as shown in Figure 5. We investigate whether the sweep is also of help in the second pair of circuits and find that, with re-encoding and sweep, they can be transformed, as shown in Figure 6. However, when trying to design a sequence of retiming and resynthesis to do the re-encoding, instead of direct applying Malik's theorem, we found that re-encoding is harder than previous thought and resynthesis needs to be slightly enhanced for retiming to be complete for re-encoding. The details will be presented in the next section.

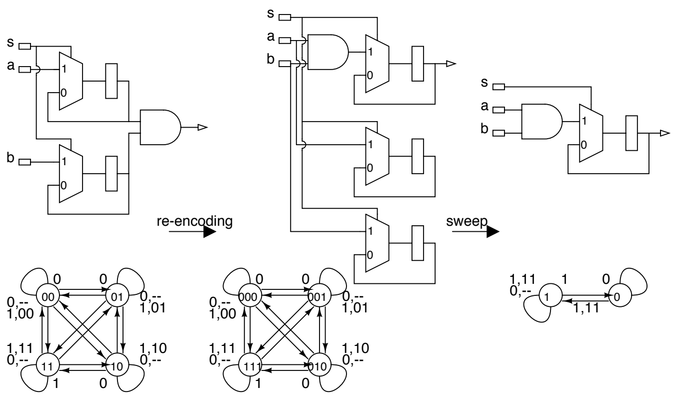

Fig. 6. Second pair is completed by re-encoding and sweep.

IV. Re-encoding is Hard

The first attempt to relate re-encoding and retiming-and-resynthesis was made by Malik et al. [10] via the following result which relates designs with different state encoding:

Theorem 4 ([10]): If two circuits have the same symbolic FSM, then one circuit can be obtained from another by a sequence of retiming and resynthesis.

However, the above theorem cannot be applied to re-encodings with different code length. we have the following result. This result also shows a sharp difference between reachability and retiming-and-resynthesis.

Lemma 5: Without any reachability information, some re-encodings with different code length cannot be completed by any sequence of retiming and resynthesis.

Proof: As we already mentioned in previous section, in Figure 6, even though the second circuit is a re-encoding of the first one, it cannot be transformed from the first one by a sequence of retiming and resynthesis. This can be proved by considering all the states in the second circuit, including all the ignored unreachable states, as shown in Figure 7. Since new cycles are created in this STG of the second circuit, retiming an resynthesis simply cannot produce such a circuit from the first one. This is based on Jiang and Brayton [6]'s characterization of STG transformations by retiming, in which no cycle can be generated or canceled. ■

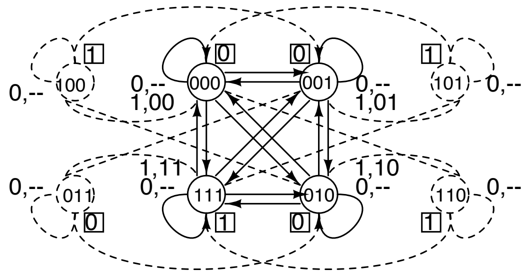

Fig. 7. Unreachable states need to be considered in re-encoding of different length.

Although re-encodings with same code length can be done by retiming and resynthesis, their verification problem is not easy. The following theorem shows that the problem is PSPACE-hard by reducing reachability problem to it.

Theorem 6 ([3]): Checking whether two circuits with the same number of registers are re-encoding of each other is PSPACE-hard.

In [6] an answer about the membership of retiming and resynthesis equivalence in PSPACE is explored. Retiming and resynthesis equivalence is reduced to immediate equivalent state minimization of the two machines and then graph isomorphism starting from known initial states. It is unclear though how graph isomorphism can be checked in PSPACE. Moreover, the proof for completeness is based on the reduction of reachability to checking whether the State Transition Graphs of the two circuits are isomorphic including the transient states. The assumption is that all dangling[1] states can be merged to non-dangling states. However, due to the binary representation of the FSM, this is not always possible. An example can be seen in Figure 8 in which no retiming and resynthesis transformation can merge the immediate equivalent states s_1 and s_3 . The reason is that for n registers the number of states in the State Transition Graph must be 2^n. When binary representation is used and the dangling states cannot be ignored, the State Transition Graphs of two retiming and resynthesis equivalent circuits may not be transformable to isomorphic graphs.

V. Completeness under Reachability

We first show a revised result for re-encoding transformation.

[1]Dangling states are inductively defined as states that have no predecessors or states whose predecessors are all dangling. All other states are considered non-dangling.

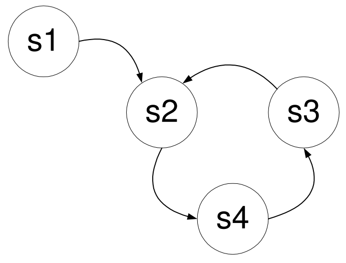

Fig. 8. State transition graph. Immediate equivalent states s_1 and s_3 cannot be merged using retiming and resynthesis in a circuit with binary representation.

Lemma 7: When the resynthesis is allowed to use the reachability information generated from one cycle, retiming and resynthesis are complete for all re-encoding transformations, including those with different coding lengths.

Proof: The proof is similar to Malik et al. [10], using schematics for circuits in Figure 9. Starting with a circuit C with the smaller encoding length n, the identity function at the register outputs is resynthesized to $f \cdot f^{-1}$ where f is the one-to-one mapping from states of C to the target states of circuit D. Please note that f may not be an onto function, thus f^{-1} may be different depending on how to map the don't care states in D. Then retiming moves the registers forward over f. The third step resynthesizes $f^{-1} \cdot C \cdot f$ into D. However, if D is encoded on a longer length m, the one-cycle reachability information needs to be used to identify the target states corresponding to states in C, which has to be used for generating D in the last step. ∎

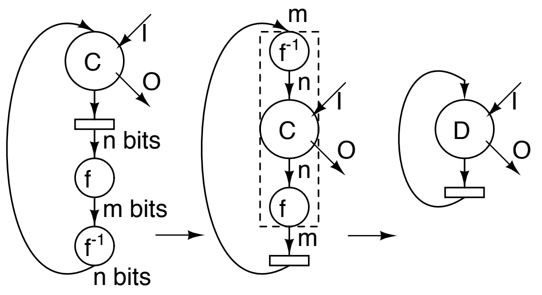

Fig. 9. One-cycle reachability makes retiming and resynthesis complete for re-encodings.

The key to the completeness of retiming and resynthesis for re-encodings is the existence of a mapping from the states of one machine to those of the other that preserves the transitions. Such a mapping is called *refinement mapping* [1].

Definition 6: For two equivalent finite state machines $(Q_1, I, O, \lambda_1, \delta_1)$ and $(Q_2, I, O, \lambda_2, \delta_2)$, a *refinement mapping* is a function $f : Q_1 \rightarrow Q_2$ such that for any $s \in Q_1$, s and $f(s)$ are equivalent, and further for any $i \in I$,

$$f(\delta(s,i)) = \delta(f(s), i).$$

Abadi and Lamport [1], studying the verification of one system implementing another, proved that, if S_1 implements S_2, then one can add auxiliary history and prophecy variables to S_1 to form an equivalent system S_1^{hp} and find a refinement mapping from S_1^{hp} to S_2 under three very general hypotheses: S_1 is machine closed, S_2 has finite invisible nondeterminism, and S_2 is internally continuous. For deterministic finite state machines, they are always true. The following result is simply a corollary of the main theorem of Abadi and Lamport [1]. But we will give a direct proof to avoid detouring via general (infinite nondeterministic) system models.

Theorem 8: If two deterministic FSMs C and D are equivalent, then one can add history variables to C to form an equivalent FSM C', and find an onto refinement mapping from C' to D.

Proof: For $C = (Q_C, I, O, \lambda_C, \delta_C)$ and $D = (Q_D, I, O, \lambda_D, \delta_D)$, we can have

$$\begin{aligned} Q_{C'} &\triangleq \{(c,d) \in Q_C \times Q_D : c \cong d\} \\ \lambda_{C'}((c,d),i) &\triangleq \lambda_C(c,i) \\ \delta_{C'}((c,d),i) &\triangleq (\delta_C(c,i), \delta_D(d,i)) \end{aligned}$$

It is straight-forward to check that $f(c,d) = d$ for any $(c,d) \in Q_{C'}$ is a refinement mapping from $C' \triangleq (Q_{C'}, I, O, \lambda_{C'}, \delta_{C'})$ to D. ∎

The proof only gives a simple construction without considering efficiency; for any states c and d such that $c \cong d$, we need only add a history variable to record the part of d that is independent of c, instead of the whole d. In the special case where each d is totally dependent on c, no history variable is needed, and the refinement mapping is the generating function of d from c.

With the refinement mapping, a completeness result can be given as follows.

Theorem 9: If two circuits are equivalent, then one of them can be transformed to the other by a sequence of sweep (inverse), resynthesis, retiming, resynthesis, and sweep, given that the second resynthesis operation is allowed to use one-cycle reachability.

Proof: For two equivalent circuits C and D, their corresponding FSMs are deterministic and equivalent. Based on Theorem 8, a set of history registers and their next state functions can be added to C to make it C', and an onto refinement mapping can be found from C' to D. Denote the mapping by F. Adding unobservable registers and their next state functions is just an inverse of the sweep operation.

If F is an one-to-one mapping, then F^{-1} exists. Otherwise, we expand F with the register outputs of C and denote by F^{-1} the function that generate the state of C' from the output of F. Resynthesis can generate F and F^{-1} connected at the register output of circuit C'. Then retiming moves the registers to the outputs of F. Since F is a refinement mapping from C' to D, the relocated registers give the states of circuit D in parallel with (possibly partial) states of circuit C. The circuit composed of F^{-1}, H (the history transition), and F can be re-synthesized into the circuit D in parallel with another circuit (partial C). Then a sweep operation will remove all unobservable part to produce circuit D. The sequence of the five operations are shown in Figure 10. A key operation in the second re-synthesis operation is to have the output from D, instead of C. This cannot be done if the register vectors V_c and V_d are assumed to be independent (as in pure combinational

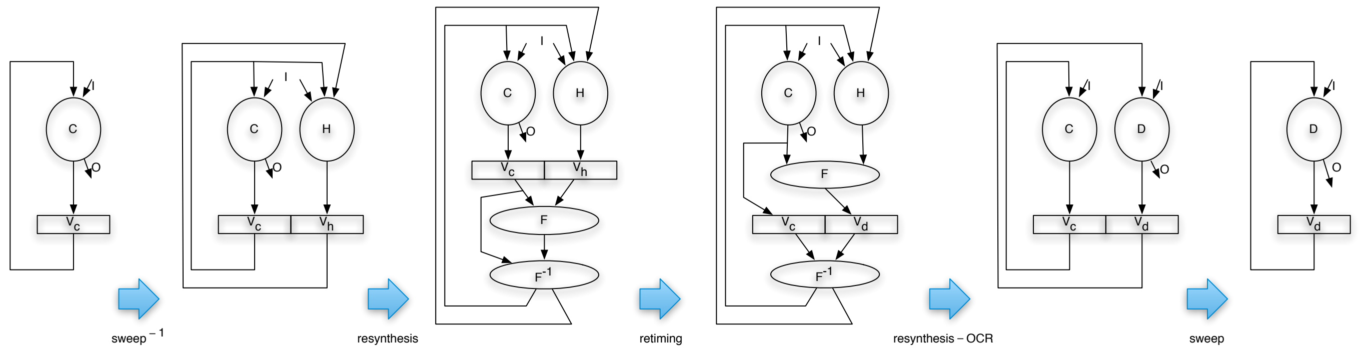

Fig. 10. Transformation from a circuit to an equivalent one by retiming and resynthesis with sweep.

synthesis). However, with the observation that $V_d = F(V_c)$ from the previous cycle, the output O can be synthesized out solely from V_d. ■

VI. Conclusions

We have shown in this paper that retiming and resynthesis with sweep are almost complete for all steady state equivalent transformations, in the sense that resynthesis needs to get one-cycle reachability information by looking into previous phase. Without such information, they cannot even complete re-encodings with different code length. It suggests that a powerful sequential optimization tool can be built around retiming, resynthesis, and sweep, and also suggests to enhance each resynthesis step to employ one-cycle reachability by looking into previous phase. In practice, resynthesis may not generate exponential-size circuits and may have other restrictions. Those restrictions can make the retiming and resynthesis equivalence checking easier.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2), 1991.

[2] A. Aziz, V. Singhal, and R. K. Brayton. Verifying Interacting Finite State Machines. Technical Report UCB/ERL M93/52, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, July 1993.

[3] José L. Balcázar, Antoni Lozano, and Jacobo Torán. The complexity of algorithmic problems on succinct instances. *Computer science: research and applications*, pages 351–377, 1992.

[4] G. DeMicheli. Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization. *IEEE TCAD*, 10(1):63–73, January 1991.

[5] G. Even, I. Y. Spillinger, and L. Stok. Retiming Revisited and Reversed. *IEEE TCAD*, 15(3):348–357, March 1996.

[6] J.-H. Jiang and R. Brayton. Retiming and resynthesis: A complexity perspective. *IEEE TCAD*, 25:2674–2686, December 2006.

[7] J.-H. Jiang and W.-L. Hung. Inductive equivalence checking under retiming and resynthesis. In *ICCAD*, San Jose, CA, November 2007.

[8] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.

[9] B. Lin, H. J. Touati, and A. R. Newton. Don't Care Minimization of Multi-level Sequential Logic Networks. In *ICCAD*, pages 414–417, November 1990.

[10] S. Malik, E. M. Sentovich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and Resynthesis: Optimization of Sequential Networks with Combinational Techniques. *IEEE TCAD*, 10(1):74–84, January 1991.

[11] Sharad Malik. *Combinational Logic Optimization Techniques in Sequential Logic Synthesis*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, November 1990. Memorandum No. UCB/ERL M90/115.

[12] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[13] R. K. Ranjan, V. Singhal, F. Somenzi, and R. K. Brayton. On the optimization power of retiming and resynthesis transformations. In *ICCAD*, pages 402–407, San Jose, CA, November 1998.

[14] V. Singhal, S. Malik, and R. K. Brayton. The Case for Retiming with Explicit Reset Circuitry. In *ICCAD*, pages 618–625, November 1996.

[15] D. Stoffel and W. Kunz. Record and Play: a Structural Fixed Point Iteration for Sequential Circuit Verification. In *ICCAD*, pages 394–399, 1997.

[16] H. J. Touati and R. K. Brayton. Computing the Initial States of Retimed Circuits. *IEEE TCAD*, 12(1):157–162, January 1993.

[17] C. A. J. van Eijk. Sequential Equivalence Checking without State Space Traversal . In *DATE*, pages 618–623, Paris, France, 1998.

[18] H. Zhou, V. Singhal, and A. Aziz. How powerful is retiming? In *Workshop Notes of Intl. Workshop on Logic Synthesis*, Lake Tahoe, CA, June 1998.