How Powerful is Retiming?*

Hai Zhou† haizhou@cs.utexas.edu

Vigyan Singhal‡ vigyan@cadence.com

Adnan Aziz§ adnan@ece.utexas.edu

Abstract

This paper is about exploring the power of retiming and resynthesis. We show that there exists a pair of "sequentially equivalent" designs so that one cannot be obtained from another by a sequence of retiming and resynthesis operations; the notion of sequential equivalence itself is predicated on whether or not the designs are associated with designated initial states. Categorizing the power of retiming and resynthesis operations is closely related to the problem of verifying sequential equivalence. We show the complexity results for the sequential verification problem. However, our primary goal, namely determining sharp bounds on the power of retiming and resynthesis, and determining the complexity of the associated verification problem remains open, and we would like to present that as a challenge to the community.

1 Introduction

The advent of modern VLSI CAD tools has radically changed the process of designing digital systems. The first CAD tools automated the final stages of design, such as placement and routing. As the low level steps became better understood, the focus shifted to the higher stages. In particular logic synthesis, the science of optimizing gate level designs for measures such as area, speed, or power, has shifted to the forefront of CAD research.

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 [Mic94] gives an excellent introduction to logic synthesis.

*Supported in part by an IBM Research Partnership Award

†Department of Computer Sciences, University of Texas, Austin, TX 78712

‡Cadence Berkeley Labs, 2001 Addison St 3rd Floor, Berkeley, CA 94704

§Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712

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 *latches*.

Retiming is a powerful sequential optimization step that can be applied to sequential designs described at the netlist level. It is used to optimize the clock-cycle timing of a design. 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. In this papers, we summarize our progress in answering the question.

We are also interested in the associated verification question: if two designs are sequentially equivalent, what is the computational complexity of checking equivalence? And, if it turns out that retiming and resynthesis operations cannot cover the entire space of sequentially equivalent circuits, we would like to know exactly how hard it is to decide if one circuit *can be* be obtained from the other by a sequence of retiming and resynthesis moves.

The verification question has important practical repercussions. For example, if it is easier to decide retiming and resynthesis equivalence, we may obtain more efficient solutions for equivalence checking when it is known that retiming is only permitted sequential transformation. Indeed, the recent sequential equivalence checking algorithms [SK97, vE98] effectively try to show that two circuits are equivalent by deriving the retiming relationships between the circuits. While we are unable to determine the complexity of retiming and resynthesis equivalence, we are able to show that the general sequential equivalence checking problem is PSPACE-complete in the size of the design.

We begin in Section 2 by describing the background: 2.1 discusses our two designs models– netlists and FSMs, Section 2.2 references the paradigms of retiming and resynthesis (we will use the term "RnR") and Section 2.3 revisits the notions of sequential equivalence. Section 3 briefly visits the case when designs have initial state. It is clear for such designs that RnR is not powerful enough in absence of unreachability information. Section 4 presents the examples which motivate our search for the gap between RnR equivalence and sequential equivalence. Section 5, then, presents our progress in showing that the gap is significant; however, our work is not complete and is predicated on a conjecture which we have been unable to prove.

2 Background

2.1 Netlists and FSMs

In order to be able to formally reason about hardware, we need to develop mathematical models for digital systems. In this section we develop two formalisms for expressing designs, namely finite state machines and netlists. Finite state machines are more abstract; they correspond to the behavioral

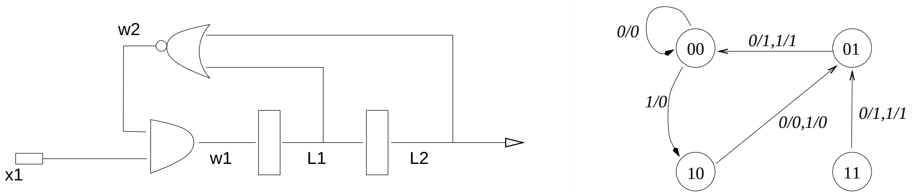

Figure 1: Netlist and corresponding FSM.

specification as given by the designer. Netlists are "structural"; they consist of an interconnection of gates and latches, and are closer to the actual implementation.

Before we give a formal syntax and semantics to netlists and FSMs, we illustrate them by means of examples. The netlist in Figure 1 has one primary input, one primary output, two latches, and two gates. The FSM derived from this netlist is shown adjacent to 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 will abuse notation and 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 $\delta(01, 0 \cdot 1 \cdot 0) = 01$.

A netlist is a representation of a design at the *structural level* which is closer to the actual implementation of the design than FSMs, which can be viewed as behavioral level descriptions of the design. In this paper, it will be clear from the context whether a design refers to an FSM or a netlist or both.

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, latches*, and *gates*. Primary input nodes have no fanins; latches have a single input. Associated with a gate g on n-inputs $w_1, w_2, \ldots, w_n$ is a function from W^n to W. (When $n = 0$ the gate is a constant). Some nodes are designated as being *primary outputs*.

Given a set of assignments to each input node and a state (an assignment of values to latches), one can uniquely compute the values of each node in the netlist by evaluating the functions at gates. In this way, a netlist η on inputs $a_1, a_2, \ldots, a_n$, outputs $b_1, b_2, \ldots, b_m$ and latches $r_1, r_2, \ldots, r_k$ 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 latch r_i we can find a function $f_i : Q \times R_i$ by composing the functions of the gates from the inputs and latch outputs to the input of the latch. We will refer to f_i as the next-state function of the latch i. Then $\delta_{M_\eta}(w_1, w_2, \ldots, w_n, r_1, r_2, \ldots, r_k) : Q \times X \rightarrow Q$ is simply $[f_1 f_2 \ldots f_k]$. Similarly, the output function is defined by composing the functions of gates from inputs and latches to output nodes.

2.2 Retiming and Resynthesis Operations

Retiming is a synthesis step which is applied to netlists. It consists of removing a fixed number of latches from each output of a combinational node and adding the same number to each input of the node[1].

The operation of retiming can be described mathematically in terms of "lag function", which indicate for each combinational node, the number of latches which are taken from each fanout, and moved to the fanins. (Negative lag means that latches are moved forward across the node.)

Retiming becomes very powerful when it is interspersed with resynthesis of the combinational logic present in the netlist (combinational resynthesis has to keep all the latches in place, and is free to resynthesize any combinational logic around the latches; thus the combinational synthesis leaves the FSM of the design unchanged). This is the basis for the retiming and resynthesis (RnR) paradigm proposed in [DeM91, MSBSV91]. 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: $r(w_1) = 1, r(w_3) = r(w_4) = -1, r(w_2) = r(w_5) = 0$.

2.3 Sequential Equivalence

Definition 3 Two states s and t are equivalent 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.*

[1] In order to avoid ambiguity, every "stem" (i.e., multi-fanout junction) is replaced by a buffer; see [SPRB95] for details.

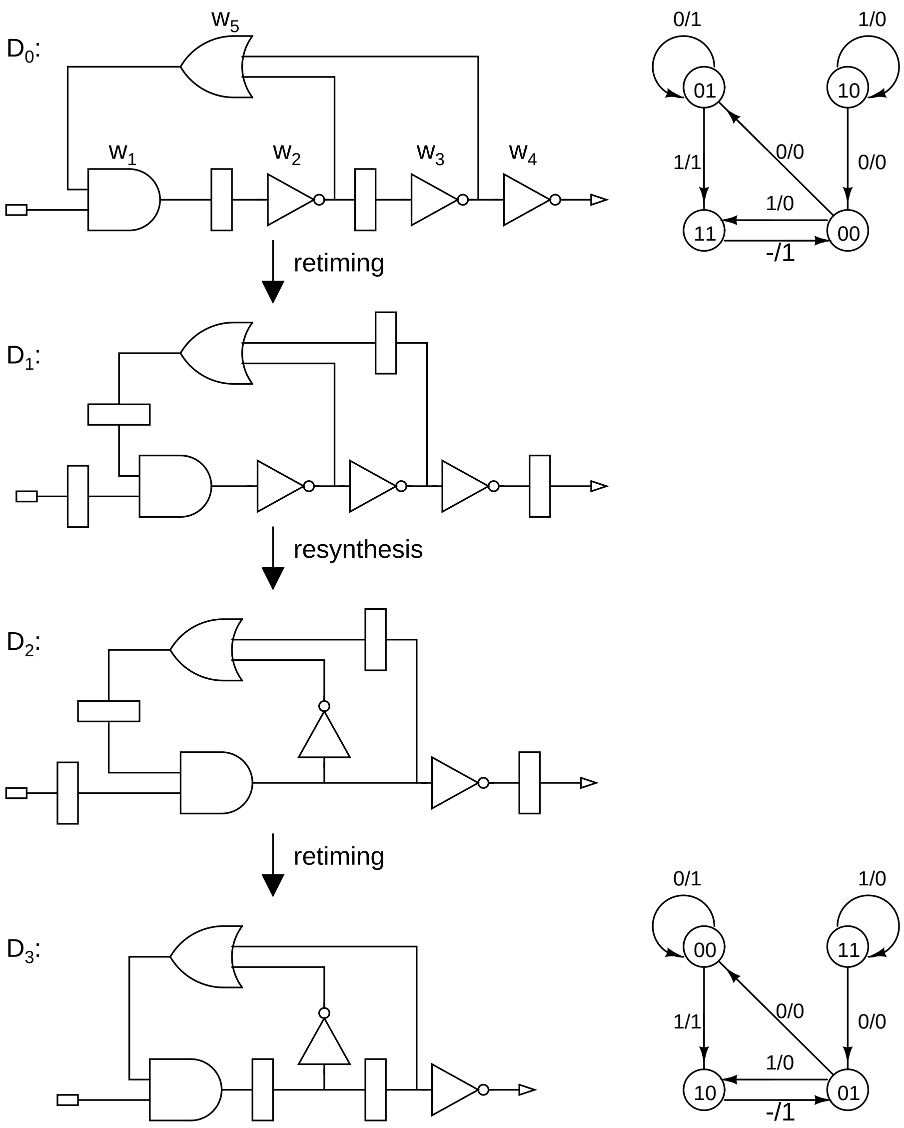

Figure 2: Example of retiming and resynthesis.

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 can not exist in a transient state if we are prepared to wait arbitrarily long.

Theorem 1 ([LS83]) *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 2 *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.*

The primary question of interest for this paper is whether the converse of the above corollary is true.

3 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 latches: indeed, this is the approach suggested for retiming initial states in [SMB96], as opposed to the approach in [TB93, ESS96], 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, RnR is strictly weaker than a sequential optimization algorithm which uses unreachability don't cares (for example, [LTN90]). 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 2, 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:

Theorem 3 ([ASB93]) *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.*

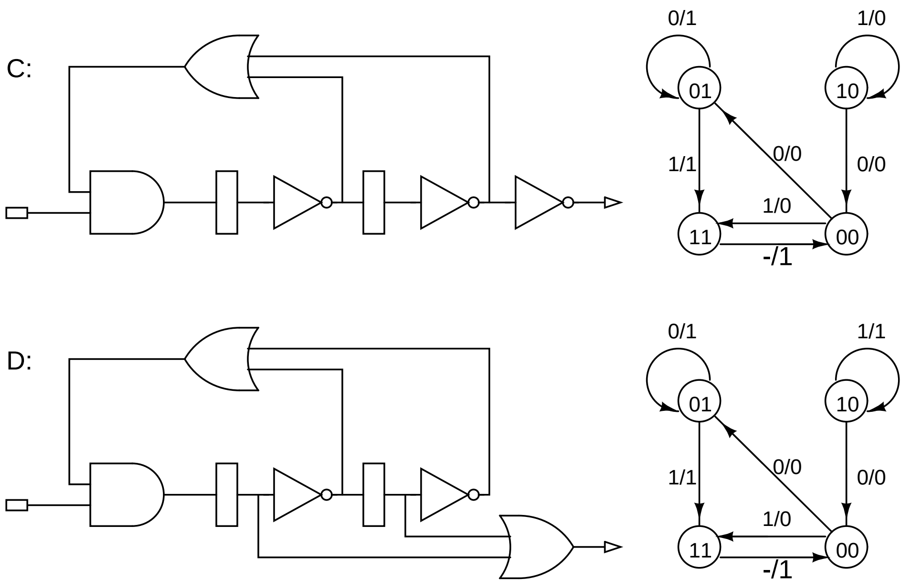

Figure 3: Circuits C and D are equivalent from designated initial state 00.

A complete proof is given in [ASB93]; the idea is to use the "generic Turing machine reduction" [GJ79]: the halting problem for a PSPACE bounded Turing machine can be modeled using a set of linearly connected FSMs, one for each tape square used, which mimic the tape contents, the finite state control, and the head location.

4 RnR equivalence vs steady-state equivalence

The big question we are interested in is whether the converse of Corollary 2 is true, namely:

Conjecture 4 *If the steady state set of design C is FSM-equivalent to the steady state set of design D, C can be obtained from D by a sequence of retiming and combinational resynthesis moves.*

The first attempt to answer this question was made by Malik [Mal90] via the following result which relates designs with different state encoding:

Theorem 5 ([Mal90]) *If two circuits have the same symbolic FSM, then one circuit can be obtained from another by an RnR sequence.*

Notice that the FSMs can have different state encoding for the states (the RnR transformation in Figure 2 is an example of this theorem). Malik gave a constructive proof for the above result. However, notice that the result is applicable only if the two designs have the *same* FSM (and not when the designs have different FSMs but are FSM-equivalent).

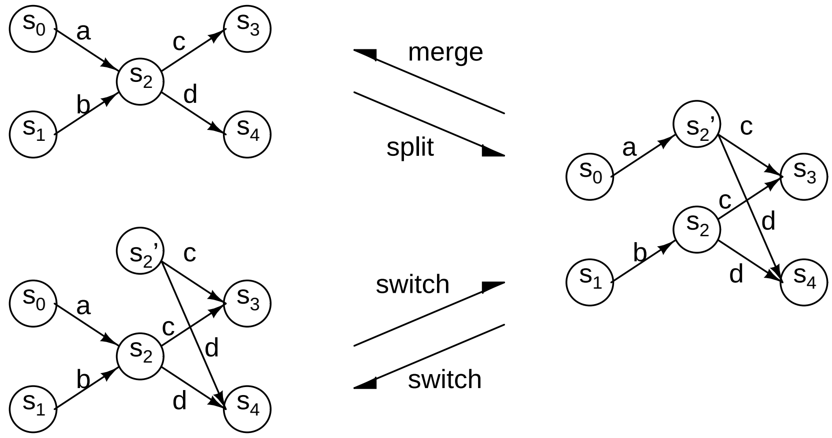

Figure 4: 1-step equivalent transformations: split, merge, and switch.

Malik also attempted to extend the above result by allowing FSM transformations. However, there is a bug in his result. The corrected result is provided by Ranjan [Ran97]; it needs the two following definitions:

Definition 6 States s and t are *1-step equivalent* if and only if, for any input x, the next states $\delta(s,x)$ and $\delta(t,x)$ are identical.

Definition 7 A *1-step equivalent transformation* is one of the following:

1. splitting a state into multiple 1-step equivalent states.
2. merging multiple 1-step equivalent states into one state.
3. switching a next state from one state to another state which is 1-step equivalent to it.

The operations split, merge and switch are shown pictorially in Figure 4.

Theorem 6 ([Ran97]) *If design D can be obtained from design C by a sequence of 1-step equivalent transformations, then D can be obtained from C by an RnR sequence.*

Notice that in order to make 1-step equivalent transformations, a sequential optimization tool needs exactly one step of reachability information. This is much cheaper than doing more general sequential optimizations, e.g. extracting the set of equivalent state pairs and using them as don't cares [LTN90, CHS91], extracting the set of unreachable states from an initial state and using them as don't cares [LTN90]. In hindsight, it may not be so surprising, that the power of RnR transformations has only been shown to be equivalent to 1-step equivalent transformations which are optimizations based on local information (1-step equivalence).

In the next two subsections, we present two examples of pairs of equivalent circuits: we prove that the first pair is not RnR equivalent, even though it is FSM-equivalent; for the second circuit, we have no proof, but we simply conjecture that it is not RnR equivalent.

4.1 First example

In this section, we will show that RnR is not complete for FSM-equivalence. An example is shown in Figure 5, where C and D are two circuits. Their FSMs are shown in Figure 6. It can be easily checked that C and D are FSM-equivalent. But we can prove that RnR can not transform one of them to the other.

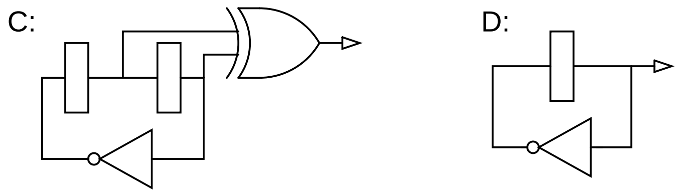

Figure 5: Example showing incompleteness of RnR

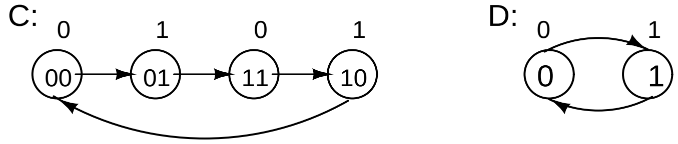

Figure 6: FSMs of the example

Theorem 7 *RnR can not transform circuit C to D in Figure 5.*

Proof: The next state function of C is a permutation on the set $\{0,1\}^2$, which has cardinality of 4. No matter what resynthesis has been done, the smallest cut size on the combinational part must be at least 2. Therefore, the next retiming step can not reduce the number of latches. Since the next state function of the new circuit still has the property as the old one, any later retiming and resynthesis steps can not reduce the number of latches, either. This means RnR can not transform circuit C to D. ■

Notice that this means that C cannot be transformed to D using 1-step equivalent transformations.

4.2 Second example

Consider the pair of circuits in Figure 7 which are FSM-equivalent. We conjecture that the two circuits are not RnR equivalent. We have tried hard but been unable to come up with a proof or a transformation which contradicts this conjecture.

Notice that if we consider input s as a load-enable, the three latches might be seen as modeling load-enable latches. This particular circuit transformation is very important because it shows retiming of load-enable latches [LVW97] can be obtained by simple RnR transformations; on the other

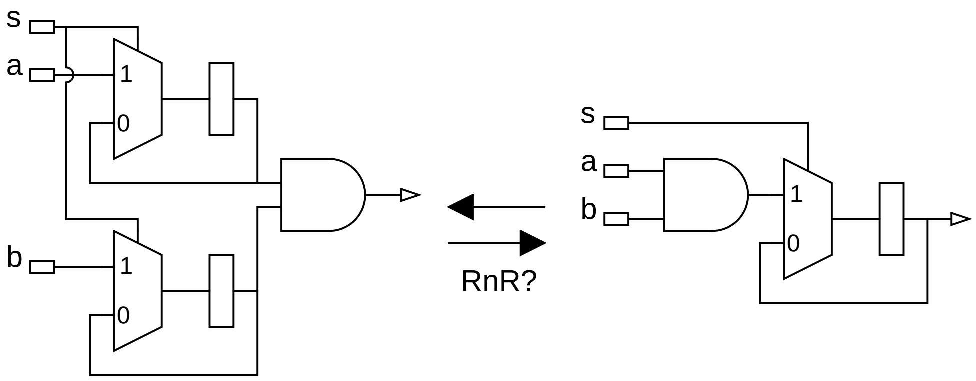

Figure 7: Example of equivalent circuits

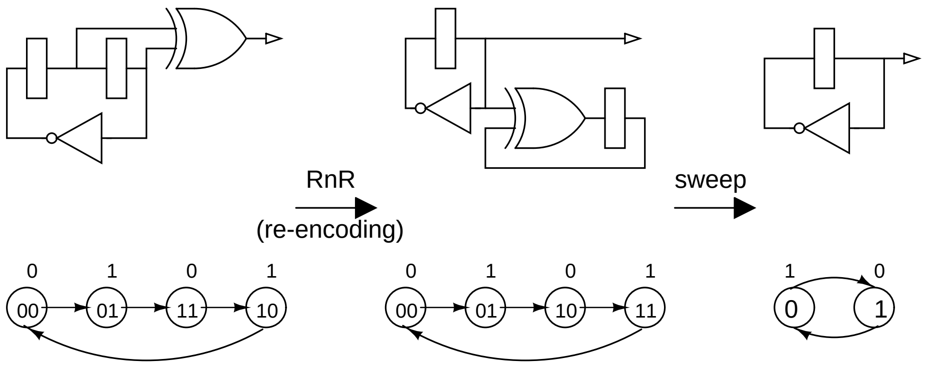

Figure 8: RnR is more powerful with `sweep`

hand, Legl et al. [LVW97] enhance their set of transformations by directly retiming load enable latches.

4.3 RnR with sweep

Actually, the two circuits in our first example (in Section 4.1) are related by a more general sequence of transformations (this has been shown by Somenzi [Som97] and also appears in [Ran97]). The new transformation we need is `sweep`: this allows removing circuitry which does not reach any primary output (notice that `sweep` is a not a combinational synthesis step since it can also remove latches from a circuit). Using this extra transformation we can relate the circuits, as shown in Figure 8. Notice that the re-encoding step can be obtained by an RnR sequence, via Theorem 5. This shows that, surprisingly, `sweep` itself is a powerful transformation which can modify FSMs.

5 Gap between RnR equivalence and FSM-equivalence

In this section we explore whether or not there is a significant gap between RnR equivalence and FSM-equivalence. The strategy is to show that the verification problem for RnR equivalence is theoretically simpler than the verification problem for FSM-equivalence. If we show this, this establishes that the gap is significant and is not just restricted to a small number of examples, as described in Sections 4.1 and 4.2. If, on the other hand, the verification problems have the same complexity, the gap is probably not significant. It is also important to note that even if the verification problem for RnR were theoretically *harder* than the verification problem for FSM-equivalence, the gap could still be significant. This is because if the only difference between RnR and FSM-equivalence is restricted to a small number of examples, we can "hard-wire" them into the simpler decision procedure and get a procedure for the harder problem. This will contradict with the complexity result.

5.1 Checking steady state equivalence is PSPACE-hard

First, we show that the verification problem for steady state equivalence is PSPACE-hard.

Actually, given two netlists with designated initial states, we can always combine their inputs together, and construct one output to indicate whether their outputs are different. Hence, we have the following corollary of Theorem 3.

Corollary 8 *Given a netlist D with only one output and an initial state s, checking whether there is an input sequence leading to output 1 is PSPACE-complete.*

Our theorem can be stated as follows.

Theorem 9 *Given two netlists D_1 and D_2, checking whether they are FSM-equivalent is PSPACE-complete.*

Proof: Membership in PSPACE follows from the fact that equivalence is in PSPACE, and that we can enumerate all pairs of states from D_1 and D_2 in PSPACE.

In the sequel, we will show it is PSPACE-hard by transforming the problem of checking whether a state in a netlist can leading to output 1 to our problem.

Given a netlist D as shown in Figure 9, and a initial state s, we construct two netlists D_1 andD_2, as shown in Figure 10. The FSMs of D_1 and D_2 are drawn in Figure 11, where each of x, y, z and t represents a set of equivalent states. It is easy to see that D_1 and D_2 are FSM-equivalent if and only if states t and z in M_2 have equivalent states in D_1. If there is no input sequence leading (D, s) to output 1, then states t in D_2 will be equivalent to state s in D_1. Therefore, states z in D_2 will be equivalent to states x in D_1. Since the only states in D_1 which can have $O_2 = 1$ are those in x, the fact that D_1 and D_2 are FSM-equivalent means states z must be equivalent to states x. This also means that s must be equivalent to t, that is, any input sequence can only lead (D, s) to 0.

Notice we actually prove FSM-equivalent is coPSPACE-complete, but coPSPACE is equal to PSPACE [GJ79]. ■

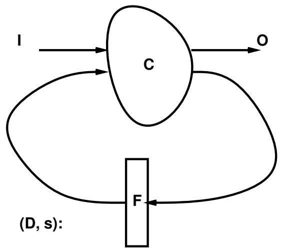

Figure 9: A sequential design with only one output.

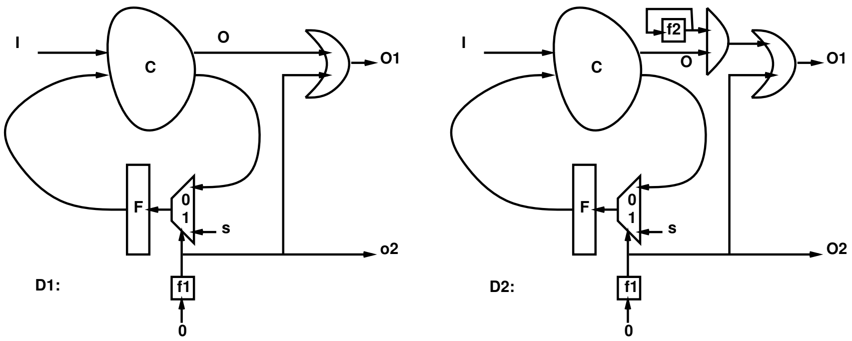

Figure 10: Two sequential designs derived from Figure 9.

In fact, exactly the same construction in the above proof can be used to prove the following result:

Theorem 10 *Given two netlists D_1 and D_2, checking whether their steady state sets are FSM-equivalent is PSPACE-complete.*

5.2 Checking RnR equivalence is probably in $\mathbf{NP^{coNP}}$

We have strong reason to believe that RnR equivalence checking problem is in $\mathrm{NP^{coNP}}$ (the complexity class $\mathrm{NP^{coNP}}$, also known an Π_2^p, lies strictly between $\mathrm{P} \cup \mathrm{NP}$ and PSPACE in the polynomial hierarchy [GJ79]), thus indicating that RnR does *not* yield the entire space of steady-state-equivalent transformations (since verification for steady-state equivalence is PSPACE-complete, by Theorem 10)[2] . However, we could not establish the complexity of RnR equivalence. Our belief of RnR equivalence lying in NP^{CoNP} is based on the following conjecture.

[2]This is predicated on the widely held conjecture that the polynomial hierarchy is strict.

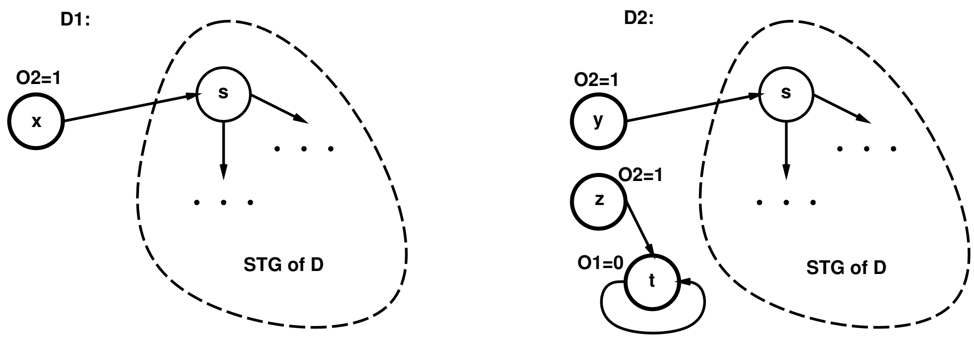

Figure 11: FSMs of the designs in Figure 10 (a thick circle represents a set of equivalent states).

Conjecture 11 *If design C can be obtained from design D via any RnR sequence, there exists an RnR sequence such that the sequence has polynomial (in the number of gates of C and D) steps, and each intermediate circuit is polynomial in the size of C and D.*

We feel that it is unlikely that the above conjecture is false. But we could not find a proof. However, *if* the conjecture is true, we can show that RnR is in $\mathrm{NP}^{\mathrm{coNP}}$ by using the following two results:

Theorem 12 *Combinational equivalence checking is in coNP.*

Proof: Follows from transforming tautology checking to combinational equivalence (the classic miter construction), and that tautology checking is coNP-complete [GJ79]. ■

Theorem 13 *Retiming verification is in coNP.*

Proof: Verifying whether a circuit is a legal transformation of another can be done in two substeps [SSBSV92]:

- Verifying directed graph isomorphism.
- Verifying that there is a consistent assignment of lags to each gate in the circuit.

Directed graph isomorphism is in coNP [GJ79], and Shenoy et al. [SSBSV92] show a polynomial algorithm for the second step. ■

Conjecture 14 *RnR equivalence is in NP^{coNP}.*

Proof: If two circuits C and D are RnR equivalent, there exists a polynomial RnR transformation (Conjecture 11). To show the equivalence between C and D, we first guess candidates for this transformation non-deterministically. From Theorems 12 and 13, checking whether each candidate is a valid transformation is in co-NP. Hence, proved. ■

6 Discussion

We have inferred a very strong conclusion. If Conjecture 11 is true, as we believe, RnR equivalence is in $\mathrm{NP}^{\mathrm{coNP}}$ and steady state equivalence is PSPACE-hard (Theorem 10). This shows a very significant gap between what RnR can achieve and steady state equivalence, showing that RnR is not as powerful as thought. On the other hand, if Conjecture 11 is false, that itself also has a great practical significance. This would show that in order to obtain transformation a circuit C to another circuit D, which is steady state equivalent to it, we will need either exponentially many RnR transformations or visit an intermediate circuit which is exponentially larger than either C or D. This clearly says that any practical RnR-based sequential optimization algorithm will be unable to transform C to D (or vice-versa).

References

[ASB93] 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.

[CHS91] H. Cho, G. D. Hachtel, and F. Somenzi. Redundancy Identification and Removal Based on Implicit State Enumeration. In *Proc. Intl. Conf. on Computer Design*, pages 77–80, Cambridge, MA, October 1991.

[DeM91] G. DeMicheli. Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(1):63–73, January 1991.

[ESS96] G. Even, I. Y. Spillinger, and L. Stok. Retiming Revisited and Reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 15(3):348–357, March 1996.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979.

[LS83] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.

[LTN90] B. Lin, H. J. Touati, and A. R. Newton. Don't Care Minimization of Multi-level Sequential Logic Networks. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 414–417, November 1990.

[LVW97] C. Legl, P. Vanbekbergen, and A. Wang. Retiming of Edge-Triggerred Circuits with Multiple Clocks and Load Enables. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, May 1997.

[Mal90] 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.

[Mic94] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw Hill, 1994.

[MSBSV91] S. Malik, E. M. Sentovich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and Resynthesis: Optimization of Sequential Networks with Combinational Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(1):74–84, January 1991.

[Ran97] Rajeev K. Ranjan. *Design and Implementation Verification of Finite State Systems.* PhD thesis, Electronics Research Laboratory, University of California, Berkeley, CA 94720, 1997. Memorandum No. UCB/ERL M97/99.

[SK97] D. Stoffel and W. Kunz. Record and Play: a Structural Fixed Point Iteration for Sequential Circuit Verification. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 394–399, 1997.

[SMB96] V. Singhal, S. Malik, and R. K. Brayton. The Case for Retiming with Explicit Reset Circuitry. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 618–625, November 1996.

[Som97] Fabio Somenzi. Personal communication, 1997.

[SPRB95] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The Validity of Retiming Sequential Circuits. In *Proc. of the Design Automation Conf.*, pages 316–321, San Francisco, CA, June 1995.

[SSBSV92] N. Shenoy, K. J. Singh, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. On the Temporal Equivalence of Sequential Circuits. In *Proc. of the Design Automation Conf.*, pages 405–409, Anaheim, CA, June 1992.

[TB93] H. J. Touati and R. K. Brayton. Computing the Initial States of Retimed Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 12(1):157–162, January 1993.

[vE98] C. A. J. van Eijk. Sequential Equivalence Checking without State Space Traversal . In *Proc. DATE: Design Automation and Test in Europe*, pages 618–623, Paris, France, 1998.