The Validity of Retiming Sequential Circuits

Vigyan Singhal* Carl Pixley† Richard L. Rudell‡ Robert K. Brayton*

Abstract

Retiming has been proposed as an optimization step for sequential circuits represented at the net-list level. Retiming moves the latches across the logic gates and in doing so changes the number of latches and the longest path delay between the latches. In this paper we show by example that retiming a design may lead to differing simulation results when the retimed design replaces the original design. We also show, by example, that retiming may not preserve the testability of a sequential test sequence for a given stuck-at fault as measured by a simulator. We identify the cause of the problem as forward retiming moves across multiple-fanout points in the circuit. The primary contribution of this paper is to show that, while an accurate logic simulation may distinguish the retimed circuit from the original circuit, a conservative three-valued simulator cannot do so. Hence, retiming is a safe operation when used in a design methodology based on conservative three-valued simulation starting each latch with the unknown value.

1 Introduction

We are interested in the optimization of synchronous digital circuits represented at the so-called net-list level. A synchronous circuit is defined loosely as an interconnection of combinational logic gates (*gates*) and synchronizing memory elements (*latches*) where where each cycle contains at least one latch. For simplicity, we consider circuits consisting of edge-triggered latches clocked directly by a single input signal.

Many sequential optimization techniques further assume the existence of a single input signal connected to each latch to set the state of the latch before the circuit begins operation. The corresponding pin on the latch is called a *synchronous reset* (or *synchronous set*) pin. However, very few designs satisfy the requirement that every latch is reset in this manner. For example, consider a digital signal processing circuit consisting of a controller connected to a datapath. The datapath consists of e.g., adders, multiplexors, and registers (latches). The controller is typically reset during the first cycle by an input signal which is connected to each latch in the controller. However, it is unnecessary for the correct operation of the circuit to force the datapath latches to reset on power-up with the same global reset line. Once the controller is reset, it simply applies a sequence of inputs to the datapath to ensure that any datapath registers (e.g., an accumulator) are initialized properly. For many designs of this style, the controller contributes less than 10% of the total latches in the design. Also, the size of a latch with a reset signal is larger than the size of latch without a reset signal and a significant area penalty arises from the routing of the reset signal. Therefore, to reduce the size of the design, none of the datapath latches is directly reset. As an extreme example, consider a pipelined 32-bit multiplier with 4 pipeline stages; to force the latches in this finite memory subcircuit to a defined value with a global reset signal is costly and totally unnecessary.

The above observation has sparked interest in algorithms for analyzing and optimizing designs where none of the latches have reset signals. In these algorithms, latches in the design which have synchronous control pins (e.g., set, reset, load enable) are modelled as simple latches surrounded by additional gates. For example, a synchronous reset latch with positive logic reset signal R and data input signal D is modelled by a simple latch and an AND gate with the AND gate fed by NOT(R) and D. Modelling latches with reset signals this way may restrict the space of retiming moves allowed because this additional logic might come in the away of latches being retimed. However, we can take care of this problem by treating each latch along with its reset logic as one block and then retiming this block as one unit. Also, we can choose to split this block; this will allow up get even more retimings which we could not have obtained had we not modelled the reset logic explicitly.

Logic optimization modifies a circuit to improve a cost function on the circuit (such as minimizing the area required to meet a specified clock period) while preserving the behavior of the circuit. It is important to note that these modifications are done at the net-list level with no knowledge of the environment in which the circuit is used;[1] hence, it is important that the op-

*Department of EECS, University of California at Berkeley, Berkeley, CA 94720

†Motorola Inc., Bridgepoint Plaza I, 5918 W. Courtyard Dr., Suite 200, Austin, TX 78730

‡Synopsys Inc., 700 East Middlefield Road, Mountain View, CA 94043

[1] Sometimes limited knowledge of the environment, such as output don't-cares or an initializing sequence, may be provided by the designer; however, our goal is to develop optimization algorithms which do not require such information to be provided.

32nd ACM/IEEE Design Automation Conference ®
Permission to copy without fee all or part of this material is granted, provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission. © 1995 ACM 0-89791-756-1/95/0006 $3.50

timized circuit be able to replace the original circuit for any environment. As a simple example, combinational logic optimization restricts modifications to the gates in the circuit. As long as the Boolean functions produced at the outputs of the combinational portion of the circuit remains unchanged, the optimized circuit can safely replace the original circuit.

A sufficient condition which allows a design to be replaced independent of its environment is the classical notion of equivalent FSMs (i.e., for every state in the first machine there is an equivalent state in the second machine, and vice-versa). A weaker condition, which is both necessary and sufficient for replacement was described[2] by Pixley *et al.* and is called *safe-replacement* [7].

Pixley introduced *Sequential Hardware Equivalence* (SHE) [6] to consider the equivalence of digital circuits under the assumption that digital circuits must operate correctly starting from a random power-up state. Consider the state transition graph of a digital circuit, which, by his definition, is a completely-specified machine with 2^n states given n latches. Collapse this machine by merging equivalent states. Strongly connected component analysis of any directed graph yields a directed acyclic graph of strongly connected components (SCC). Pixley argued that for the behavior of the circuit to be well-defined under the assumption of a random power-up state, the state-minimal graph of the circuit must have a single sink SCC (i.e., a single terminal SCC or TSCC). All interesting notions of replacement require equivalence of the TSCC of the two designs, as the TSCC defines the steady-state behavior of the machine. The subtlety which distinguishes SHE and various notions of replacement is in dealing with the transient behavior of the machine (all states outside the TSCC). Replacement requires that the environment be able to drive a replacement machine into its steady-state behavior (i.e., reset the machine) with the same sequence used to reset the original machine.

Retiming was first formulated by Leiserson and Saxe [4] in the context of systolic systems. When applied to digital circuits, retiming is an optimization step which moves the latches across the logic gates and in doing so changes the number of latches and the longest path delay between the latches. In this manner, the number of latches can be reduced, and/or the cycle time of the circuit can be improved. Recent results by Shenoy and Rudell [9] have improved the efficiency of retiming so that circuits up to 50,000 equivalent gates can be retimed for minimum area under a delay constraint. This has sparked further interest in exploring the application of retiming as a general optimization step during logic synthesis.

Prior literature has assumed that retiming can be directly applied to sequential circuit optimization. However, it has been recently pointed out by Pixley *et al.* [7] that retiming does not satisfy the safe-replacement condition. Note that the theorem of Leiserson and Saxe on the validity of retiming is not in doubt. They simply made the assumption that the environment of the circuit could be modified to wait a fixed number of cycles (dependent upon the retiming) before applying its inputs. It is this requirement which violates safe-replacement and casts a doubt on whether retiming is valid as part of a synthesis methodology for net-list level sequential circuits.

In this paper, we first show in Section 2 how a designer who applies retiming to a circuit could be surprised during simulation when the retimed design replaces the original design. Specifically, it is possible that a logic simulator could produce a different output sequence when simulating the retimed circuit. We also show on the same example that an input sequence which tests a fault in a circuit cannot test the same fault in the retimed circuit. In Section 3 we formalize our model of retiming on sequential circuits, which is slightly different from that of Leiserson and Saxe in that we distinguish the case where a latch moves from the output of one gate to the inputs of each of its fanouts. We then re-prove in Section 4 the Leiserson and Saxe result that retiming is a safe operation as long as one can wait long enough after power-up before applying the input sequence. As a consequence of our proof, we identify that the only incorrect retiming transformation is one which moves a latch forward across a multiple-fanout junction; i.e., if we limit the retiming transformations, then retiming satisfies the condition of safe-replacement. Section 5 is the key to the paper. We define a *conservative three-valued logic simulator* (CLS) as a three-valued simulator using the values 0, 1, and X which performs only local propagation of the X values (i.e., AND$(0, X) = 0$ but AND$(1, X) = X$).[3] Further, the CLS begins operation with all latches in the X state. In Section 5 we show that while a simulator could distinguish a retimed circuit from the original, in fact, a conservative three-valued logic simulator cannot. In other words, retiming retains an invariant on the output sequences produced by conservative three-valued simulation.

As a final comment, note that our model of a synchronous circuit does not require a latch to have a set or reset line and does not require any notion of the initial state of the circuit. Hence, we avoid the problem pursued by Touati and Brayton [10] in *retiming the initial state*.

2 Retiming Violates Safe-Replacement

2.1 Simulation Example

Here we show how a simple retiming move might change the behavior of a design. Consider the circuit D and the retimed version C in Figure 1. The STGs for these circuits are shown in Figure 2. Design D has two states and is initialized to state 0 on the length-1 input sequence 0, whereas C is not initialized with this input sequence.

If we simulate the two design D and C with an input sequence, we may get different results. Consider the input sequence $0 \cdot 1 \cdot 1 \cdot 1$. The simulation results for this input sequence

[2] In the context of fault detection, Pomeranz and Reddy [8] have presented an equivalent condition which is used to identify sequentially redundant faults.

[3] In contrast, an exact three-valued simulator will output an X only in the case that some assignment of values to X on the input yield differing outputs; for a conservative simulator the local propagation of X's may cause an X on the output even though the output is a 0 (or a 1) for every input.

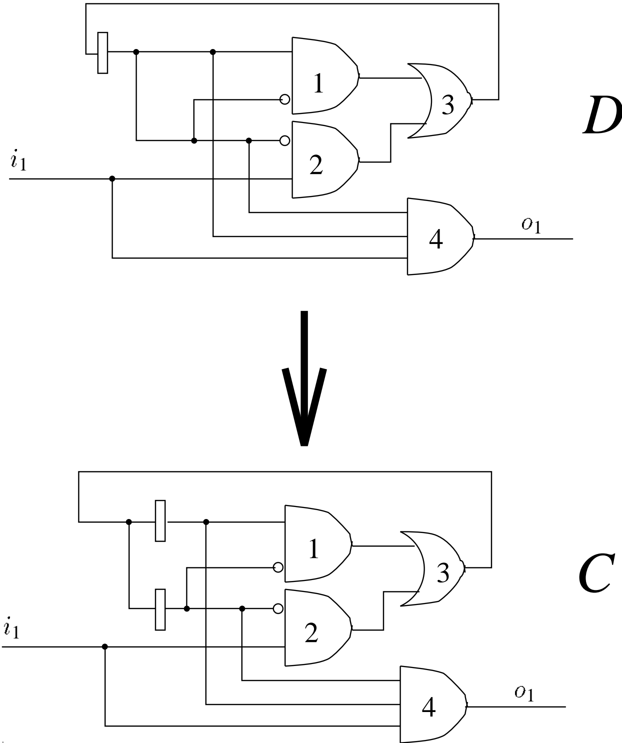

Figure 1: Retimed circuit is not initialized with input sequence 0.

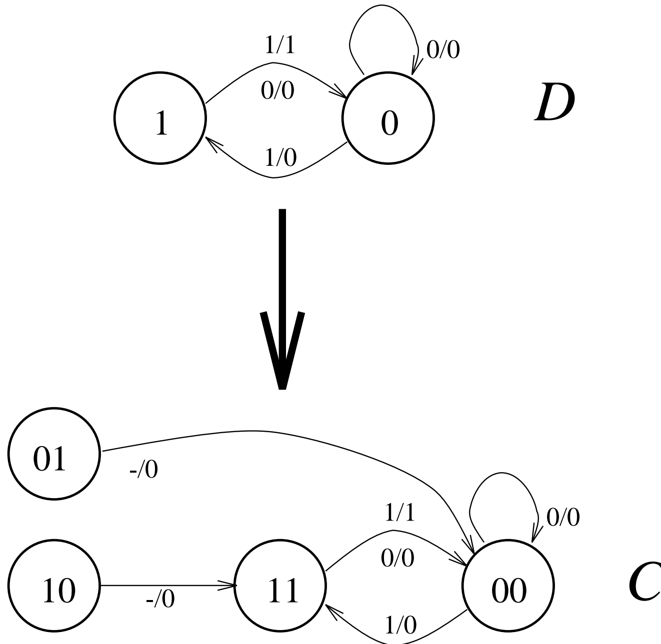

Figure 2: Design where retiming breaks down an initializing sequence of length 1.

power-up state of D	output sequence	power-up state of C	output sequence
0	$0 \cdot 0 \cdot 1 \cdot 0$	00	$0 \cdot 0 \cdot 1 \cdot 0$
1	$0 \cdot 0 \cdot 1 \cdot 0$	11	$0 \cdot 0 \cdot 1 \cdot 0$
		01	$0 \cdot 0 \cdot 1 \cdot 0$
		10	$0 \cdot 1 \cdot 0 \cdot 1$

Table 1: Simulation results for D and C on input sequence $0 \cdot 1 \cdot 1 \cdot 1$.

from all the power-up states of D and C are shown in Table 1. This input sequence produces the same output sequence from every power-up state of D. However, if design C powers up in state 10, it will output $0 \cdot 1 \cdot 0 \cdot 1$, resulting in an input/output behavior which was not present in the original design.

Suppose we had a sufficiently powerful simulator which given any input sequence, outputs

- a 1 at an output at some time step iff all power-up states output 1 at that time step
- a 0 at an output at some time step iff all power-up states output 0 at that time step
- an X otherwise, i.e. if there exist two power-up states, one of which outputs a 0 and the outputs a 1.

For the input sequence $0 \cdot 1 \cdot 1 \cdot 1$ this simulator would output $0 \cdot 0 \cdot 1 \cdot 0$ for design D and output $0 \cdot X \cdot X \cdot X$ for the design C.

Note, however, that if we clock the circuit for one redundant cycle (with arbitrary input) before applying our input sequence, even our imaginary powerful simulator will produce the same output for both circuits. This is the sense of a *delayed circuit* which we define in Section 3 and is the notion of equivalence used by Leiserson and Saxe when proving that retiming was a valid transformation on a circuit.

2.2 Testing Example

The example in the previous subsection shows that retiming can change the behavior of a design as measured by a simulator. Next we show that test sets can also be affected similarly, contradicting the result of Marchok *et al.* [5]:

(Theorem 1 in [5]) *The retiming transformation preserves testability with respect to a single stuck-at-fault test set.*

This theorem implies that if a test sequence uncovers a given stuck-at-fault in a circuit, then the same sequence can detects the same faults in a retimed version of the circuit. For a counterexample, consider circuits D and C in Figure 3. For the given stuck-at-1 fault shown, the test sequence $0 \cdot 1$ detects the fault in the original circuit D. For the input sequence $0 \cdot 1$, the fault-free version of D produces the output $0 \cdot 0$ from all power-up states whereas the faulty version of D produces the output sequence $0 \cdot 1$. Thus, $0 \cdot 1$ is a valid test sequence for the stuck-at-fault in D since it distinguishes the faulty design from the fault-free design. However, for the fault-free version of circuit C, the input sequence $0 \cdot 1$ may produce output $0 \cdot 0$ or

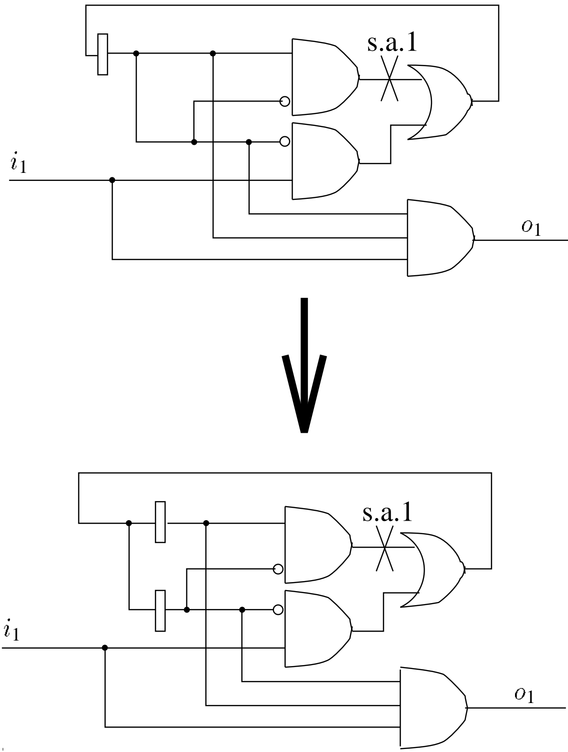

Figure 3: Retiming does not preserve test sequence $0 \cdot 1$.

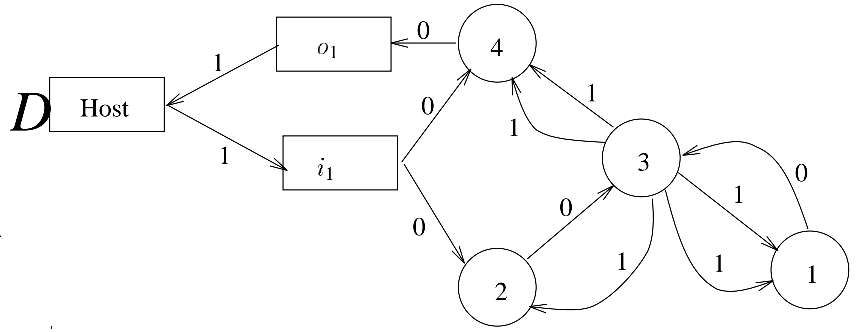

Figure 4: The edge-weighted digraph representing the circuits D and C from Figure 1.

$0 \cdot 1$ depending on which state C powers up in (see the STG for C in Figure 2); the faulty version of C still produces $0 \cdot 1$ from any power-up state. Thus, $0 \cdot 1$ is no longer a test sequence for the retimed design D.

3 Background

3.1 Leiserson-Saxe Retiming Model

Leiserson and Saxe introduced retiming [4] through a graph-theoretic model. A design is modeled as a finite edge-weighted directed graph $G = (V, E)$. Each vertex in V represents either a gate in the design, a primary input or output, or a special dummy node called the *host*. There is an edge in E from one gate to another if an output of this gate fans out to the second gate; there is an edge from the host to each primary input node; and, an edge from each primary output node to the host. The non-negative weight of an edge represents the number of latches on the corresponding path in the design. The host and primary input and primary output nodes are required to have a lag of 0. A retiming of a design is an assignment of each vertex v to an integer $lag(v)$ such that for every edge (u, v) with weight w, the value $w + lag(v) - lag(u)$ is non-negative. Informally, the lag of a vertex denotes the number of backward retiming moves across this register, for example a lag of -2 on a logic element means that 2 forward retiming moves are performed across this element.

This model is not well suited for retiming on gate-level sequential circuits. The problem is that if a single output of an element fans out to more than one element, this model does not distinguish where the latches are placed with respect to the fanout junction. This is best seen with respect to the retiming example discussed in Section 2.1. Both the circuits in Figure 1 are represented by the same retiming graph, which is shown in Figure 4.

3.2 Circuit Model

Our model of a sequential circuit is the traditional net-list level model which consists of elementary cells from a library interconnected with wire connections (for example, the circuits in Figure 1). The library cells consist of combinational gates and latches. As discussed in the introduction, we allow latches with synchronous control signals (set, reset, load enable) but do not require that all latches be of this type.

The reason retiming caused a problem in the example of Section 2.1 is that we retimed a latch forward across a fanout junction and created a power-up state which could not occur in the original circuit. If there are multiple-output gates in the cell library such that there exist output vectors of the cell which cannot be produced by any input vector to the cell, then retiming latches forward across such elements leads to the same problem as retiming latches forward across fanout junctions.

This motivates the definition of justifiable and non-justifiable multiple-output gates. Consider a multi-output gate F with n inputs and m outputs. The m output functions are denoted by $f_1, \ldots, f_m$. F is *justifiable* if and only if for every output $y \in 2^m$, there exists an input $x \in 2^n$ such that $y = F(x)$; if there exists $y \in 2^m$ such that for all $x \in 2^n$, $y \neq F(x)$, then F is *non-justifiable*.

A k-way fanout junction ($k > 1$) is a special case of a multi-output gate (which we call JUNC) with 1 input line x and k ($k > 1$) output lines $y_1, y_2, \ldots, y_k$, where $y_1 = \cdots = y_k = x$ (Figure 5). The element JUNC is clearly non-justifiable since only two of the 2^k output vectors ($000\ldots0$ and $111\ldots1$) are possible. For the remainder of this paper we assume that all fanout junctions are replaced by JUNC elements. This implies that each output of each gate (latch) fans out to exactly one other gate (latch).

For a gate-level network which replaces junctions with

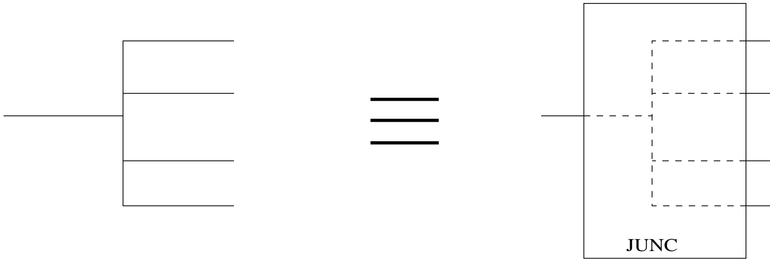

Figure 5: A junction can be treated as a multi-output gate.

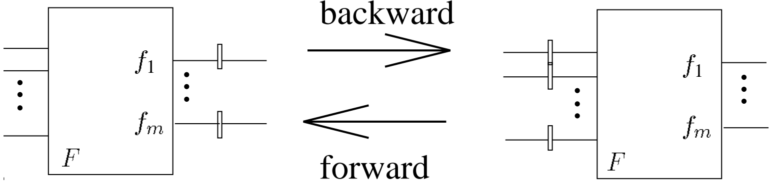

Figure 6: Forward and backward retiming moves across a multi-output element.

multi-output gates as described above, there are two kinds of atomic retiming moves: *forward* and *backward*. A forward move removes one latch from each of the n inputs and places one latch at each of the m outputs; a reverse move removes one latch from each of the m outputs and places one latch at each of the n inputs (Figure 6). We view retiming starting from an initial circuit and applying a sequence of these atomic moves to result in the retimed circuit.

3.3 Notions of Replaceability

Here we discuss the various notions of design replacement that are relevant to our work.

The notion of *safe replaceability* was presented in [7]. A design C is a *safe replacement* for a design D (denoted by $C \preceq D$) if for any state s_1 in design C and any input sequence, there exists a state s_0 in design D such that the output behavior from s_1 is the same as that from s_0 on that input sequence. It was shown that the above condition is necessary and sufficient so that the replacement cannot be detected by any environment that can only control and observe only the primary inputs and outputs, respectively, of the design.

A stronger notion of design replacement is the classical notion of state machine implication usually defined in the context of state machine equivalence. A design C *implies* design D (we denote that by $C \subseteq D$) if for any state s_1 in design C there exists a state s_0 in design D such that s_1 is equivalent to s_0 (i.e., on any input sequence, the output behavior from s_1 is the same as that from s_0).

The difference between $\preceq$ and $\subseteq$ lies in the fact that for the former the state s_0 in D depends not only on s_1 but also on the input sequence, whereas for $\subseteq$ s_0 only depends on s_1 and is the same for any input sequence. It has been shown in [7], that if $C \preceq D$, there may be a state in C which is not equivalent to any state in D. Thus, $\subseteq$ is a strictly stronger notion than $\preceq$.

In the following sections we will prove results which characterize the conditions under which the relation $\subseteq$ holds between two designs. The following proposition shows that these results automatically imply safe replaceability ($\preceq$) as well.

Proposition 1 *If $C \subseteq D$, then $C \preceq D$.*

Proof Consider any state $s_1 \in C$. Since $C \subseteq D$, there exists a state $s_0 \in D$ which is equivalent to s_1. Thus, given s_1, for any input sequence π, there is a state in D (namely s_0) such that s_1 and s_0 output the same sequence on π. ∎

3.4 Delayed Designs

We need the notion of delayed designs (similar to *sufficiently old configuration* of [4]) to show the results in our paper. Given a design D, an n-cycle-delayed design (denoted by D^n) is the set of states of D if we allow arbitrary inputs to run through design D for n clock cycles after power-up. The design D^n will be design D minus some transient behavior of design D which can be seen only during the first n cycles after power-up. For example, consider the design C in Figures 1 and 2. If we run arbitrary inputs through C for one cycle after power-up, we can never reach states 01 or 10. The delayed design C^1 consists of states 11 and 00 only and thus C^1 is equivalent to the design D.

4 Safety of Retiming Moves

In this section we classify retiming moves into those which are strictly safe for safe-replacement, and those which are safe only under the assumption of a delayed design.

There are four kinds of atomic retiming moves:

(i) Backward across a justifiable element.

(ii) Forward across a justifiable element.

(iii) Backward across a non-justifiable element.

(iv) Forward across a non-justifiable element.

Proposition 2 *If design C is obtained from design D by a single retiming move which is either a backward move or across a justifiable element, then $C \subseteq D$.*

Proof Let the retiming move be across a multi-output combinational element $F = (f_1, \ldots f_m)$.

Cases (i) and (iii) **(backward retiming moves)**. Assume that design D has latches $l_1, l_2, \ldots, l_k$. Let the latches $l_1, l_2, \ldots, l_m$ be retimed to latches $l'_1, l'_2, \ldots, l'_n$. Consider any state in design C, say $s_1 \equiv [(l'_1, l'_2, \ldots, l'_n, l_{m+1}, \ldots, l_k) = (Y'_1, Y'_2, \ldots, Y'_n, Y_{m+1}, \ldots, Y_k)]$. Let $\vec{Y'}$ denote $(Y'_1, Y'_2, \ldots, Y'_n)$. We claim that state $s_0 \equiv [(l_1, l_2, \ldots, l_k) = (f_1(\vec{Y'}), f_2(\vec{Y'}), \ldots, f_m(\vec{Y'}), Y_{n+1}, \ldots, Y_k)]$ in design D has the same input-output behavior as s_1. The proof of this

claim is by induction on the length of an input sequence. Outside of the retimed area, the two circuits D and C are identical. On the first clock cycle, the local output of F is $(f_1(\vec{Y'}), f_2(\vec{Y'}), \ldots, f_m(\vec{Y'}))$ in both designs D and C. Suppose the local outputs are identical upto the k-th primary input vector. Then the k-th local input vector $\vec{W}$ reaching the retimed area will be identical for both designs. Now, for $(k+1)$-th primary input vector, the local outputs of the retimed area in both designs is $(f_1(\vec{W}), f_2(\vec{W}), \ldots, f_m(\vec{W}))$. Thus, by induction, for every input sequence, the local outputs are equal in both designs. Since the two designs are identical outside the retimed area the two primary outputs of the two designs are also equal. Thus, s_0 and s_1 are equivalent.

Case (ii) **(forward retiming move across a non-justifiable element)** Assume that design D has latches $l_1, l_2, \ldots, l_k$. Let the latches $l_1, l_2, \ldots, l_n$ be retimed to latches $l'_1, l'_2, \ldots, l'_m$. Consider any state in design C, say $s_1 \equiv [(l'_1, l'_2, \ldots, l'_m, l_{n+1}, \ldots, l_k) = (Y'_1, Y'_2, \ldots, Y'_m, Y_{n+1}, \ldots, Y_k)]$. Since the logic element is justifiable, there must exist an input vector $\vec{Z} = (Z_1, \ldots, Z_n)$ such that for each $i \in \{1, \ldots, m\}$: $f_i(\vec{Z}) = Y'_i$. Now, consider the state $s_0 \equiv [(l_1, l_2, \ldots, l_k) = (Z_1, \ldots, Z_n, Y_{n+1}, \ldots, Y_k)]$ in design D. Using an induction argument, similar to the last case, on the length of the input sequence, we can easily show that the two states s_0 and s_1 are equivalent.

Thus, for all three cases, for every state in C there is a state in D which is equivalent to it. ■

Proposition 3 *If design C is obtained from design D by a single forward retiming move across a non-justifiable element, then $C^1 \subseteq D$.*

Proof Assume that design D has latches $l_1, l_2, \ldots, l_k$. Let the latches $l_1, l_2, \ldots, l_n$ be retimed to latches $l'_1, l'_2, \ldots, l'_m$. Consider any state in design C^1, say $s_1 \equiv [(l'_1, l'_2, \ldots, l'_m, l_{n+1}, \ldots, l_k) = (Y'_1, Y'_2, \ldots, Y'_m, Y_{n+1}, \ldots, Y_k)]$. Since the design C^1 represents the design C after it has been clocked through for 1 cycle, there must exist a vector $\vec{Z} = (Z_1, \ldots, Z_n)$ such that for each $i \in \{1, \ldots, m\}$: $f_i(\vec{Z}) = Y'_i$. The rest of the proof is identical to that of case (ii) in Proposition 2. ■

Propositions 2 and 3 lead to the following corollary, which is the primary correctness result proven by Leiserson and Saxe [4] (notice that our proof is much simpler than the 4-page proof given in [4]).

Corollary 4 (Retiming Lemma in [4]) *If C is obtained from D using an arbitrary sequence of retiming moves, then $C^n \subseteq D$, for some positive finite integer n.*

The retiming lemma requires us to delay using the retimed design for n clock cycles after power-up. In that sense this may change the observed behavior from a design, as shown by the example in Section 2.1. However, if we disallow the forward retiming moves across non-justifiable elements we have the following result which shows that the retimed design is a safe replacement of the original design.

Corollary 5 *If C is obtained from D using an arbitrary sequence of retiming moves, none of which is a forward retiming move across a non-justifiable element, then $C \subseteq D$.*

In the proof of Lemma 1 in [4], the integer n was shown to be the equal to $\max_{v \in V}(-lag(v))$, i.e. the maximum number of forward retiming moves across any combinational element in the circuit.[4] We can tighten this result to make it independent of the number of forward retiming moves across any non-justifiable element.

Theorem 6 *If C is obtained from D using a sequence of retiming moves such that there are no more than k forward retiming moves across any non-justifiable element, then $C^k \subseteq D$.*

The proof of the above theorem is quite long, and is omitted here because it is not the focus of this paper. This result would imply, for example, that if a design has only single-output gates, then any number of retiming moves (forward or backward) across those gates is fine, but we must restrict the number of forward retiming moves across any fanout junction to be at most k for $C^k \subseteq D$ to hold.

Note that the maximum number of forward retiming moves across any gate can be bounded by the maximum number of registers in any simple cycle in the circuit.[5]

Test Set Preservation

The example given in Section 2.2 showed that retiming may invalidate a single-stuck-at-fault which was valid before retiming. We can use the results discussed above to show the following result:

Theorem 7 *If C is obtained from D using a sequence of retiming moves such that there are no more than k forward retiming moves, then the test set for D is also a test for C^k.*

Proof (Sketch) Consider the fault-free circuit G and the faulty circuit F. Create a circuit $T \equiv (G \parallel F)$ which denotes the two circuits next to each other and each pair of outputs of G and F fed to an XNOR gate. Any single stuck-at-fault test will produce a 0 at one of the outputs of T. Consider any single forward retiming transformation which modifies the two parts of T in the same way; the resulting circuit is T'. Now, for any single stuck-at-fault outside the retimed area we can use the techniques in the proof of Theorem 6 to prove the desired result. The forward retiming step creates m new nets between the logic element and the m retimed latches. For any single stuck-at-fault on any such input net to a latch, we use the observation that we can use the same test as for the output net of the latch but we may have to delay the circuit by 1 clock cycle to let the stuck-at-value settle inside the latch. ■

[4] Since the host is required to have $lag(v) = 0$, n is well-defined for a given retiming.

[5] Recall that the primary outputs of the circuit are connected to the *host* and the primary inputs are fed by the *host*; hence, cycles may include paths from the primary outputs through the host to the primary inputs.

For the example in Section 2.2 this theorem concludes that the test sequence $0 \cdot 1$ is a test sequence for the same fault in design C (Figure 3). Thus, either of the two sequences $0 \cdot 0 \cdot 1$ or $1 \cdot 0 \cdot 1$ would serve as a test sequence for C. It can be seen by simulating the design that either of these test sequences produces $X \cdot 0 \cdot 0$ in the fault-free version of C and the sequence $X \cdot 0 \cdot 1$ in the faulty version, thus distinguishing the two versions on the 3rd clock cycle.

5 Retiming Preserves Conservative Three-valued Simulation

Consider again the example of Figure 1 of Section 2. In circuit D the output of AND gate-1 is 0 whether the latch has value 0 or 1. For this reason, it is easy to deduce that a single cycle with primary input 0 will reset the latch to value 0. However, notice that if the latch is assigned the indeterminate value X, then a conservative three-valued simulator (CLS) will propagate X values to both of the inputs of AND gate-1. Furthermore, the CLS will propagate an X as output of the AND gate because the CLS has lost the information that the X's are complements of each other. Hence a single cycle input of 0 that will *actually* reset design D will not *appear* to reset design D in three-valued simulation. This should not surprise us because the amount of information lost by three-valued simulation is precisely the same information lost by moving a latch forward across an unjustifiable element. We show that this is a general phenomena relating three valued simulation and retiming.

Simulation is an important component of the IC design verification process. The most popular and fastest way of simulating gate-level designs is three-valued simulation [3]. It is assumed that all latches power up as X, meaning that the value is undetermined. Three-valued logic is well-known for gate-level elements [2]. Three-valued simulation results give the output sequences for a given input sequence. However, it is well-known that three-valued simulation is more conservative than reality: if three-valued simulation shows a 0 (or a 1) for an output, that output will be 0 (or 1) for all power-up states; however, the converse is not true because three-valued simulation might show an X for an output even though that output may be determined to be either 0 or 1 from all power-up states. Although three-valued simulation is conservative, in the absence of other fast methods of verification it is popular in the design process. In fact, if a three-valued simulation shows an X where a designer expects a 0 or a 1, the designer often changes the design so that the output of a CLS agrees with the desired output, even though the original design may also have been correct.

This motivates the work presented in this section. Here, we show that if our yardstick for correctness of a design is the output of a CLS, starting from the state in which all latches are initialized to X, then retiming transformations do not change the observed behavior of a design (as seen from the output of the CLS).

Given that a conservative three-valued simulation is used for the design, we can assume that three-valued logic is already defined for all the combinational logic elements in the design, for example, for a 2-input NOR gate, the output is 1 if both inputs are 0, 0 if either input is 1, and X otherwise. We have to show that if two designs D_0 and D_n start off with each latch initialized to X, and D_n is obtained from D_0 by a sequence of retiming moves, then any sequence of three-valued inputs will produce the same outputs from both D_n and D_0. As in the last section, we model junctions as a special single-input multiple-output combinational element JUNC. Also, assume that if all inputs of any combinational element are X's, then all outputs are X's. We need this condition to guarantee that when all of the latches are initialized to X's in both an initial and a retimed circuit, they will generate the same outputs. For example, if a gate had a constant value of, say 0, then then a forward retiming move across this gate would assign the value of X in the initial state. If the output of the gate were a primary output, they would obviously have different observable behavior.

We prove the main result in this section by considering one retiming move at a time, and then using induction on the number of retiming moves. Suppose that design D is obtained from design C by a *single* retiming move (Figure 7). We define a relation $\mathcal{R}$ between the states of C and the states of D. We will assume that the retiming move is backward across F. The case for a *forward* retiming move is symmetric — just the roles of C and D are interchanged in the definition of $\mathcal{R}$. Since D results from C by a single retiming move, designs C and D are identical except for the retiming area around the logic element F shown in the figure. Suppose that F implements the functions $f_1, f_2, \ldots, f_q$. For design C, let inputs to F be $(i_1, i_2, \ldots, i_k)$, the latches C be $(l_1, l_2, \ldots, l_q)$ and the outputs be $(o_1, o_2, \ldots, o_q)$. For design D, let the inputs be $(i'_1, i'_2, \ldots, i'_k)$, the latches be $(ll_1, ll_2, \ldots, ll_k)$ and the outputs of F be $(o'_1, o'_2, \ldots, o'_q)$.

Now consider any state s_0 of C and state s_1 of D. We are going to define what it means for s_0 and s_1 to be related by $\mathcal{R}$. First we require that (1) corresponding latches outside of the retiming area have the same values for states s_0 and s_1. Second we require that (2) corresponding outputs of the retiming area have equal values, that is for each i, $o_i = o'_i$. We observe that conditions (1) and (2) imply that, for the same primary input vector, the primary outputs of the two circuits have equal values.

We now show that that if $s_0 \mathcal{R} s_1$ and if both designs are clocked one cycle with the same primary inputs to get states s'_0 and s'_1, respectively, then $s'_0 \mathcal{R} s'_1$. This implies that for any sequence of the same three-valued inputs to the two designs, the primary outputs will also be equal.

Theorem 8 *Suppose circuit D is obtained from C by one retiming move. Suppose also that s_0 and s_1 are three-valued states of C and D respectively, and suppose that $s_0 \mathcal{R} s_1$. Let s'_0 and s'_1 be states of C and D respectively after introducing the same input sequence. Then $s'_0 \mathcal{R} s'_1$.*

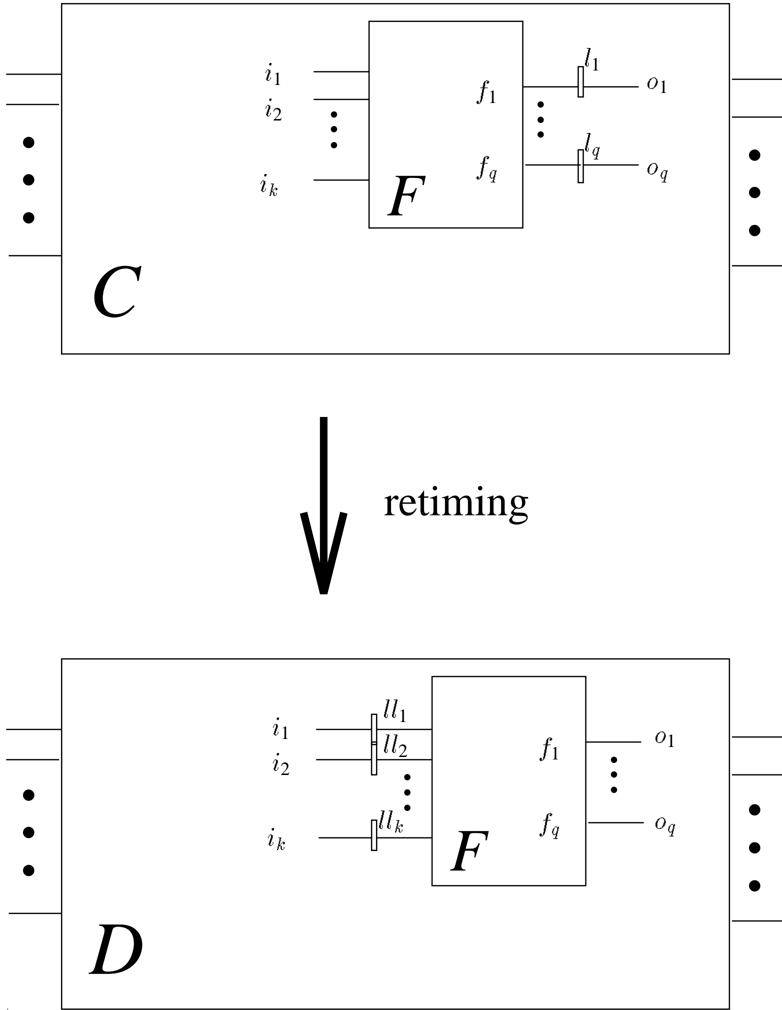

Figure 7: Backward retiming move across a multi-output logic element.

Proof Suppose circuit D is obtained from C by a *backward* retiming move. The forward case is the same with C and D reversed. Suppose that s_0 and s_1 are states of designs C and D respectively, such that $s_0 \mathcal{R} s_1$. Consider the case where the input sequence is of length 1. We aim to show that next states s_0' and s_1' are also related by $\mathcal{R}$. Conditions (1) and (2) of $\mathcal{R}$ and the fact that primary inputs are the same imply that outside the retiming area, the next state functions of corresponding latches have the same value. Therefore condition (1) of $\mathcal{R}$ is satisfied for next states s_0' and s_1'.

We observe that conditions (1) and (2) imply that (3) the corresponding inputs to the retiming area are equal, that is, for all j, $i_j = i_j'$. But (3) implies that that the inputs (i.e., next state) to latches $l_1, l_2, \ldots, l_q$ equal $F(i_1', i_2', \ldots, i_k')$. That is to say, the inputs to the l latches equal F of the inputs of the ll latches. This in turn implies that, in the next state, corresponding outputs of the retiming area are equal, i.e., condition (2) holds in the next state. We therefore conclude that $s_0' \mathcal{R} s_1'$.

By induction on the length of the input sequence, we conclude that the relation $\mathcal{R}$ is preserved by any sequence of inputs starting from states in which $\mathcal{R}$ is true and the theorem is established. ■

By induction on the number of retiming moves, we have the following corollary:

Corollary 9 *Suppose circuit D_n is obtained from D_0 by a sequence of n retiming moves. Suppose also that s_0 and s_1 are states of D_0 and D_n respectively and that $s_0 \mathcal{R} s_1$. Then for any sequence of three-valued input vectors, the output sequences of D_0 and D_n from the states s_0 and s_1, respectively, are the same.*

When a CLS is used to verify the correctness of designs, all latches are initialized to X's and input vector sequences are supplied to the CLS. If all latches are initialized to X's in both the original design D_0 and the retimed design D_n, then clearly the two initial three-valued states in the two designs are related by the relation $\mathcal{R}$ (it is here that we need our assumption that for any combinational element in the design if all inputs are X's, then all outputs are X's). The above results lead to the following result, which establishes the validity of retiming moves, if three-valued simulation is used as a criterion for correctness of the design:

Corollary 10 *Suppose circuit D_n is obtained from D_0 by any sequence of retiming moves. Suppose that s_0 and s_1 are the states of D_0 and D_n respectively obtained by initializing each of the latches to the value X. Then for any sequence π of three-valued inputs, the output sequences from of D_0 and D_n are the same. If π resets D_0 then it also resets D_n and vice-versa.*

6 Conclusions

Much of the prior logic synthesis literature has pursued refinements of retiming without first addressing the validity of retiming as part of a design methodology. Our goal in this paper has been to show how to fit retiming into a synthesis design

flow. We strongly advocate that useful sequential optimization techniques must deal with circuits which contain latches without reset signals. Hence, we have explored retiming under this model and demonstrated that retiming could cause a simulator to produce different results. We then showed that the conservative nature of traditional three-valued simulators allows retiming to maintain a simulation invariant. Because, in practice, all current design methodologies rely on this type of three-valued simulation, we conclude that retiming of designs without set and reset signals fits into a synthesis methodology.

We have not addressed a technical point: whether retiming materially affects the operation of the real circuit. We know that if we wait *long enough* (a number of clock cycles bounded by the number of latches in any cycle) then retiming does not affect the circuit operation. When a circuit powers-up there is always some delay before the circuit begins operation; it takes time, potentially many clock cycles, for the voltages to settle, etc. Therefore, the requirement that the circuit settle for a slightly longer number of cycles before it begins computation may not, in actuality, cause a problem. However, modern design methodologies rely heavily on logic simulation to the point that if simulation says the circuit doesn't work, then the designer must assume the circuit doesn't work.

One future area we wish to explore further is the notion of *three-valued safe replacement*. This is similar to the notion of equivalence used by Cheng [1] for synthesis via redundancy removal. We have shown in this paper that if we replace the strict notion of equivalent output sequences with the weaker notion of equivalent output from a conservative three-valued simulator, that retiming can be viewed as an operation preserving safe replaceability. We would like to develop algorithms to validate three-valued simulation equivalence and other optimization algorithms which seek only to preserve this invariant (and not the invariant of safe replaceability).

7 Acknowledgements

We would like to thank Sharad Malik of Princeton University for pointing out the problem with modelling reset logic explicitly. During this research, the first author was supported by NSF/DARPA Grant MIP-8719546.

References

[1] K.-T. Cheng. Redundancy Removal for Sequential Circuits Without Reset States. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 12(1):13–24, January 1993.

[2] E. B. Eichelberger. Hazard Detection in Combinational and Sequential Circuits. *IBM J. Res. and Devep.*, pages 90–99, March 1965.

[3] J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg. A Three-Value Computer Design Verification System. *IBM J. Res. and Devep.*, pages 178–188, 1969.

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

[5] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski. Test Set Preservation under Retiming Transformation. Technical Report CMUCAD-94-23, Carnegie Mellon University, 1994. Presented at Intl. Test Synthesis Workshop, Santa Barbara, CA, May 1994.

[6] C. Pixley. A Theory and Implementation of Sequential Hardware Equivalence. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 11(12):1469–1494, December 1992.

[7] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton. Multilevel Synthesis for Safe Replaceability. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 442–449, November 1994.

[8] I. Pomeranz and S. M. Reddy. Classification of Faults in Synchronous Sequential Circuits. *IEEE Transactions on Computers*, 42(9):1066–1077, September 1993.

[9] N. Shenoy and R. Rudell. Efficient Implementation of Retiming. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 226–233, November 1994.

[10] 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.