DELAYING SAFENESS FOR MORE FLEXIBILITY

by

Vigyan Singhal, Carl Pixley, Adnan Aziz,
and Robert K. Brayton

Memorandum No. UCB/ERL M95/5

17 January 1995

DELAYING SAFENESS FOR MORE FLEXIBILITY

by

Vigyan Singhal, Carl Pixley, Adnan Aziz,
and Robert K. Brayton

Memorandum No. UCB/ERL M95/5

17 January 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DELAYING SAFENESS FOR MORE FLEXIBILITY

by

Vigyan Singhal, Carl Pixley, Adnan Aziz,
and Robert K. Brayton

Memorandum No. UCB/ERL M95/5

17 January 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Delaying Safeness for More Flexibility*

Vigyan Singhal† Carl Pixley‡ Adnan Aziz† Robert K. Brayton†

January 17, 1995

Abstract

Recent work has identified the notion of safe replacement for sequential synchronous designs which may not have reset states. It has also been illustrated that many popular sequential resynthesis techniques, such as retiming, may cause unsafe replacements. However, this happens only due to changes in the transient behavior during the first few cycles after powering up the design. Some knowledge about the environment could then be used to show that these changes may be acceptable for a sequential replacement. We present a spectrum of replaceability notions which address these issues. Finally, we present experimental results to demonstrate that significant optimizations can be gained by using these new notions of replaceability, and that there is a trade-off between delaying safeness in replaceability and the degree of optimization that can be obtained.

1 Introduction

The problem of design replacement for gate-level synchronous, sequential circuits is to replace a given design with another (hopefully, optimized in some respect) without making any assumptions of the interactions of the designs with its environment. The sequential nature of the design comes from the use of memory elements (like latches and flip-flops). A large number of real industrial circuits contain many latches with no reset lines. Instead the whole design is somehow reset probably with some kind of implicit or explicit synchronizing sequence. Also, as in real designs, we cannot assume that a particular component of design is always initialized with a given initializing sequence. Such a

*Research supported by NSF/DARPA Grant MIP-8719546 and SRC Grant 94-DC-324

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

‡Motorola Inc., MD OE321, 6501 William Cannon Drive West, Austin, TX 78735

sequence is often undetermined and may change at a later stage in the design process, so we would like to optimize our design without placing any restrictions on the ways the environment of the component can interact with it.

Previous efforts have been made to tackle this problem[1]. A notion of design equivalence was defined in [Pix92]. Another notion of equivalence was suggested in [Che93] which was used for sequential resynthesis by redundancy removal in [EC93]. Other sequential resynthesis techniques relevant to our model are retiming and resynthesis [MSBS91] and synchronous recurrence equations [DD92]. In [PSAB94], it was shown that all the above notions of equivalence make some assumptions about the environment. There, a stronger notion of design replaceability was presented which does not make *any* assumptions of the environment. However, for many designs, this notion may be too strong as seen by the low amount of optimization obtained by using this notion. In this paper, we provide a parametrized notion of replacement, where the "safeness" of the replacement can be traded off for more flexibility (and hence more optimizations) by the designer. The parameter is based on the number of clock cycles that elapse between the time when a design is powered up and when it is used. Any realistic design will be used only after some cycles have elapsed after power-up; these cycles are necessary for the voltages and current to settle in immediately after power-up.

We have implemented these new notions of design replacement, and through experiments, show that there is significant optimization to be gained from using this notion. We also show the trade-off between the degree of safeness of the replacement and the amount of optimization.

2 Terminology

A gate-level synchronous design consists of a set of interconnected memory elements and hardware gates. The memory elements assume the next state value at the start of the clock cycle. For simplicity, we consider edge-triggered latches (simply called a "latch") clocked with a single phase clock. We assume that memory elements do not have reset lines, since those with reset lines can easily be transformed into ones without [PSAB94]. Consider a design D with n input wires, m output wires and t memory-elements. Let I_D and O_D represent the input values and output values of the design. Thus, I_D consists of 2^n values and O^D consists of 2^m values. The design D also has 2^t states

[1] In this paper, we will not discuss other sequential resynthesis techniques which use the information of a designated start state of the circuit to extract flexibility and use this for optimization. These methods cannot be applied to the case where memory elements do not have reset lines.

(since it has t latches); we use D to denote the state space of the design D also— it will be clear from the context if we are talking about the design or the state space. Input sequences will denote a finite sequence of input values (for example, input sequence $\pi = a_1 \cdot a_2 \cdots a_p \in I^p$). The next state function specified by design D is denoted by the completely-specified function $\delta_D : D \times I_D \rightarrow D$. Similarly, the output function of design D is denoted by the completely-specified function $\lambda_D : D \times I_D \rightarrow O_D$. We also use δ_D and λ_D to denote functions on input *sequences*, for example, $\delta_D(s, \pi) = \delta_D(\delta_D(s, a_1), \pi')$ and $\lambda_D(s, \pi) = \lambda_D(s, a_1) \cdot \lambda(\delta(s, a_1), \pi')$ where $\pi' = a_2 \cdot a_3 \cdots a_p$.

A design can also be represented by state transition graphs (STG's) denoting the deterministic finite state machine (DFSM) encoded by the design. A t-bit binary-valued label on a state denotes that, in the design, the state is implemented by that assignment of the t latches. Note that because a combinational function can be implemented in many different ways, the design-to-STG transformation is a many-to-one mapping.

Conversely, given a DFSM (a set of states closed under any input), we use the term "design" to denote any gate-level implementation which encodes this DFSM. Notice that even though the number of states in the DFSM may not equal to 2^t for some natural number t, we can always use more than one encoding of the t latches to denote the same state in the DFSM, thus realizing a gate-level implementation for the DFSM.

3 Notions of Sequential Replaceability

A safe replacement condition is design such that if the original design is replaced with the new design, there is no way the environment can detect the replacement by looking at the input-output behavior of the design.

The precise condition for safe replacement was presented in [SP94]:

Definition 1 *Design D_1 is a* safe replacement *for design D_0 (denoted by $D_1 \preceq D_0$) if given any state $s_1 \in D_1$ and any finite input sequence $\pi \in I^*$, there exists <u>some</u> state $s_0 \in D_0$ such that the output behavior $\lambda_{D_1}(s_1, \pi) = \lambda_{D_0}(s_0, \pi)$.*

As, an example of safe replacement, consider design P and Q in Figures 1 and 2, respectively. It can be seen that $Q \preceq P$.

It was shown there that the above condition is the weakest possible (allows maximum flexibility for replacement) which guarantees that the environment cannot detect the replacement.

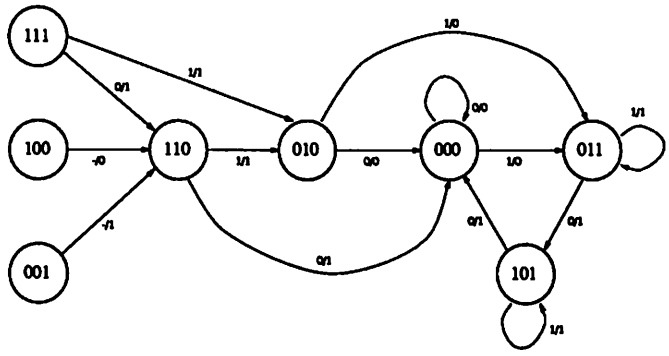

Figure 1: Example design P

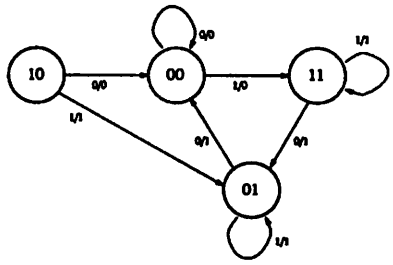

Figure 2: Replacement design Q

We frequently use the following two observations from [SP94]:

- If each state of design C is equivalent to a state in design D, then $C \preceq D$.
- The relation $\preceq$ is transitive.

Other notions of replacement have been presented and used for sequential synthesis. It is shown in [PSAB94] that all of these notions, like retiming/resynthesis [MSBS91], design equivalence [Pix92], synchronous recurrence equations [DD92], sequential redundancy removal [Che93] do make use of some information from the environment and the replacements may not be safe if the environments are allowed to be arbitrary.

Unfortunately, the notion of safe replacement was shown to be quite stringent to achieve sufficient flexibility for any significant optimizations. This motivated our current work, where based on the number of clock cycles between the power-up and when a design is actually is used, sufficient flexibility can be extracted from a design.

4 Delayed replacements

Definition 2 *Given a design D, an* n-cycle delayed design *(denoted by D^n) is the set of states $\{s|\exists \pi \in I^n, s' \in D : \delta_D(s', \pi) = s\}$, i.e. a state s belongs to D^n iff there exist a power-up state s' in D and an input sequence of length n which drives s' to s. Let D^∞ denote the set of states $\{s|\forall n : \exists \pi \in I^n, s' \in D : \delta_D(s', \pi) = s\}$, i.e. a state s belongs to D^∞ iff for each natural number n there exist a power-up state s' in D and an input sequence of length n which drives s' to s.*

D^n is thes set of states into which any state must fall when clocked n times with any sequence of inputs. It is easy to see that the set of states in D^n is closed under all inputs; also that the set of states in D^m is a subset of the states in D^n if $m > n$. The design D^∞ can be obtained by a fixed-point operation starting from D, because $D^\infty = D^n$, where n is the smallest number such that $D^{n-1} = D^n$. Using the terminology in [Jeo92, PSAB94], this number n is the number of onion rings of the design D.

We call the set of states in D^∞ as the *stable* states of D, and the states in $D \setminus D^\infty$ as the *transient* states of D. If, after powering up a design, we pass an arbitrary set of inputs through the design for a long enough number of clock cycles we will be left with the stable set.

For example, consider the design shown in Figure 1. For this design P, the various n-delayed designs are: $P = \{111, 100, 001, 110, 010, 000, 101, 011\}$, $P^1 = \{110, 010, 000, 101, 011\}$, $P^2 = \{010, 000, 101, 011\}$, $P^3 = P^4 = \cdots = P^\infty = \{000, 101, 011\}$.

Now, we give two parameterized definitions for safe replacement:

Definition 3 *Given a design D, a new design C is an* n-delay-preserving *replacement for D, if $C^n \preceq D^n$.*

The intuition behind this definition is that the environment of the replaced design is willing to ignore an arbitrary set of outputs for n clock cycles after power-up. This may be the case when it is known that the synchronizing sequence for the entire system will not reach the target design for the first n clock cycles. Or, alternately, the designer is prepared to accept that for at least n clock cycles after power-up the design is allowed to stabilize and its outputs are not used; any synchronizing sequence is applied to the system only n cycles after the power-up.

We can weaken the above replacement requirement in Definition 3 even further (by compromising "safeness") to get the following definition:

Definition 4 *Given a design D, a new design C is an* n-delay-accumulating *replacement for D, if $C^n \preceq D$.*

The intuition behind this is that if the original design worked in an environment, the replaced design will work in the same environment if it is allowed to settle for an extra n cycles after the power-up. This might be the case when the designer knows that the environment around the design is flexible enough to wait for n extra clock cycles and then use whatever input/output behavior from the design it was expecting. This definition requires delaying the interaction of the design with the environment for a fixed number of clock cycles with arbitrary inputs. On the other hand, the delay preserving definition (Definition 3) uses some knowledge about the interaction of the design with the environment and does not necessarily mean an extra wait after the power-up. The more important difference is explained in Propositions 4.4 and 4.5 which show that delay-accumulating replacements accumulate delays whereas delay-preserving do not. Also, we can show that n-delay preserving implies n-delay accumulating (Proposition 4.1). Later on in this section we show examples of the two kinds of delay replacements.

Proposition 4.1 *If $C^n \preceq D^n$, then $C^n \preceq D$.*

Proof: Since the set of states of D^n is a subset of the set of states in D, it follows that $D^n \preceq D$. [SP94] shows that $\preceq$ is transitive, and this proves that $C^n \preceq D$. ∎

As the delay parameter n (we will call it *slack*) increases, the corresponding replacements impose strictly weaker restrictions, as the following two results show:

Proposition 4.2 *If $C^n \preceq D^n$, and $m > n$, then $C^m \preceq D^m$.*

Proof: By contradiction. Suppose $C^n \preceq D^n$, and $m > n$, and $C^m \not\preceq D^m$. Without loss of generality, $m = n + 1$. There exists a state $s \in C^m$ and an input sequence π such that for every state $t \in D^m$: $\lambda(s, \pi) \neq \lambda(t, \pi)$. Since $s \in C^{n+1}$, there must be a state $s' \in C^n$ and an input a such that $\delta(s', a) = s$. Now, consider the state $s \in C^{n+1}$, and the input sequence $a \cdot \pi$. Since, for every state $t' \in D^n$: $\delta(t', a) \in D^{n+1}$, it is true that for every state $t' \in D^n$: $\lambda(s', a \cdot \pi) \neq \lambda(t', a \cdot \pi)$. This contradicts the fact that $C^n \preceq D^n$. ∎

Proposition 4.3 *If $C^n \preceq D$, and $m > n$, then $C^m \preceq D$.*

Proof: The proof follows from the fact that the set of states in C^m is a subset of the set of states in C^n. ∎

The above results about the relative strengths of the replaceability notions are illustrated in Figure 3.

Proposition 4.4 (each delay-preserving replacement preserves power-up delay) *If* $C^n \preceq D^n$ *and* $B^m \preceq C^m$, *and* $p = \max(m, n)$, *then* $B^p \preceq D^p$.

Proof: Suppose $m > n$. Then $C^m \preceq D^m$, from Proposition 4.2. Since $\preceq$ is transitive, $B^p \preceq D^p$. The proof is similar if $m \leq n$. ∎

Proposition 4.5 (each delay-accumulating replacement adds to power-up delay) *If* $C^n \preceq D$ *and* $B^m \preceq C$, *then* $B^{m+n} \preceq D$.

Proof: From Proposition 4.2, $B^{m+n} \preceq C^n$. Since $\preceq$ is transitive, we have $B^{m+n} \preceq D$. ∎

Thus, the nice property about delay-preserving replacements is that if many design changes or optimizations are made in succession, the final design is still as safe as one of the replacements in the chain; for example, if the designer makes a series of 1-delay-preserving replacements, knowing that the first input vector to the design can be arbitrary, the final design is still a 1-delay-preserving replacement.

On the other hand, the delay-accumulating replacements add to the delays after power-up; for example, three 1-delay-accumulating replacements would lead to a 3-delay-accumulating replacement. However, if the designer is initially given a total slack allowed for the replacements, many delay-accumulating replacements can be made in succession, as long as the total delay does not exceed the slack.

It is interesting to note that as the slack is increased, in the limit both delay-preserving and delay-accumulating replacements become identical concepts (see Figure 3 for the complete picture of the safeness/flexibility tradeoffs):

Proposition 4.6 $C^\infty \preceq D^\infty$ *if and only if* $C^\infty \preceq D$.

Proof: **(Only if part)** Assume that $C^\infty \preceq D^\infty$. Since, D^∞ is a closed subset of states in D, it follows that $D^\infty \preceq D$. From the transitivity of $\preceq$ [SP94], we have $C^\infty \preceq D$.
(If part) Assume that $C^\infty \preceq D$. but $C^\infty \not\preceq D^\infty$. Then there exists a state $s \in C^\infty$ and an input sequence π such that for any state $t \in D^\infty$: $\lambda(s, \pi) \neq \lambda(t, \pi)$ (Claim 1). Suppose n is the number

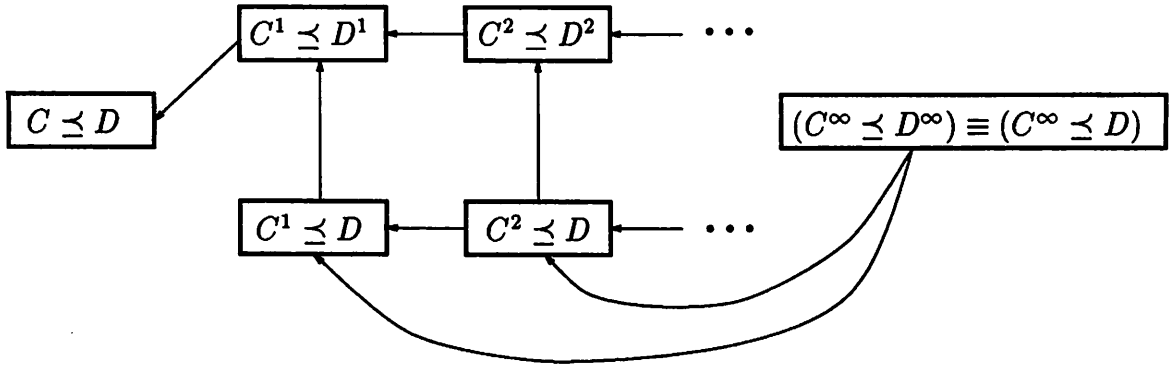

Figure 3: The safeness/flexibility tradeoff between the various replaceability notions. The arrows point towards a strictly safer notion which will afford strictly smaller flexibility for resynthesis. The arrows are transitive. The $C^\infty \preceq D^\infty$ notion is the least safe among those shown.

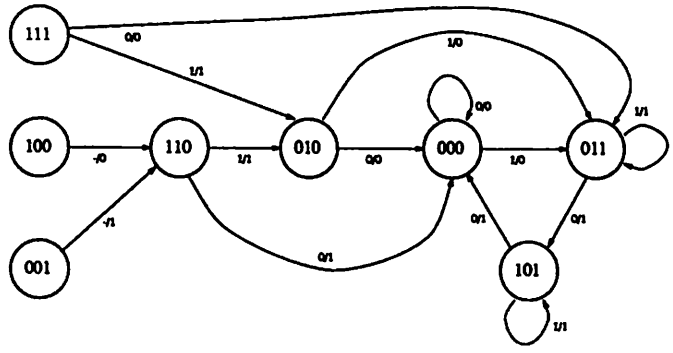

Figure 4: Example design R

of onion rings of D, (i.e. $D^\infty = D^n$). Then, since $s \in C^\infty$, there exists a state $s' \in C^\infty$ and an input sequence ρ such that $|\rho| > n$ and $\delta(s', \rho) = s$ (Claim 2). Now consider the input string $\rho \cdot \pi$ applied to state $s' \in C^\infty$. For any state $t \in D$, $\delta(t, \rho) \in D^\infty$ since $|\rho| > n$. From Claims 1 and 2, it follows that for any state $t \in D$: $\lambda(s', \rho \cdot \pi) \neq \lambda(t, \rho \cdot \pi)$. Since $s' \in C^\infty$ and t is any state in D this contradicts that $C^\infty \preceq D$. Thus by contradiction, $C^\infty \preceq D^\infty$. ∎

We call such a replacement, a *self-stabilizing replacement*, i.e. if $C^\infty \preceq D^\infty$ then C is a self-stabilizing replacement for D. Intuitively, C can replace D the designer has sufficient control over the environment so that it can wait for a sufficient number (finite and bounded by a function of the design) of clock cycles after the power-up. So, we can replace design D with design C if we can afford to run enough cycles through the design after the power-up so that the transient behavior of the design disappears.

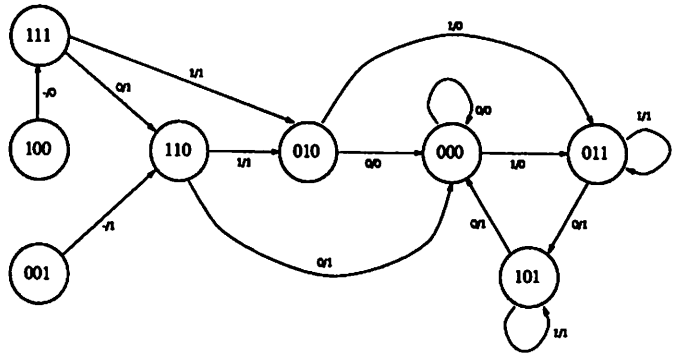

Figure 5: Example design S

As examples of delay-replacements consider designs R and S in Figures 4 and 5. It can be seen that $R^1 \preceq P^1$; however, $R \not\preceq P$ (the state $111 \in R$ and input sequence $0 \cdot 0 \cdot 0$ cannot be simulated by any state in P). From Proposition 4.1, $R^1 \preceq P$. It can also be seen that $S^1 \preceq P$; however $S^1 \not\preceq P^1$ (the state $111 \in S^1$ and input sequence $0 \cdot 1 \cdot 0$ cannot be simulated by any state in P^1).

Compositionality of Delayed Replacements

If a delayed replacement (delay-preserving or delay-accumulating) of a component of a large design is used, it is desirable to characterize the compositional effects of the above notions for design replacement.

Denote a composition of two designs A and B by $A \otimes B$. Some of inputs of A will be outputs of B; the remaining inputs of A will be primary inputs of the composed design $A \otimes B$; Similarly, some of the outputs of A will be inputs of B and the remaining outputs of A are primary outputs of $A \otimes B$. Similarly for design B.

The following result follows from the definition of safe replacement ($\preceq$):

Proposition 4.7 (compositionality of safe replaceability) *If $C \preceq D$, then $(R \otimes C) \preceq (R \otimes D)$, for any composed design $R \otimes D$.*

Proof: Consider any input sequence π to the design $R \otimes D$, and any state $(r, d) \in (R \otimes D)$. If we simulate the design $(R \otimes D)$ starting from the state (r, d) and inputing the sequence π, we will observe a sequence ρ which in the input to the sub-design D, and an output sequence σ which is the output of D. Since $C \preceq D$, there must be a state $c \in C$ such that the design C outputs σ on the input sequence ρ. Now, consider the state $(r, c) \in (R \otimes C)$. Clearly, on the input sequence π,

the composed design $(R \otimes C)$ will produce the same output sequence from state (r, c) as the design $(R \otimes D)$ would from state (r, d). ∎

The above result means that a series of safe replacements of any component results in a safe replacement of the composed design.

Proposition 4.8 (compositionality of delay-accumulating replacement) *If $C^n \preceq D$, then for any composed design $R \otimes D$, $(R \otimes C)^n \preceq (R \otimes D)$.*

Proof: Consider the composed design $(R \otimes C^n)$, which represents design R composed with the n-delayed design C^n. Clearly, every state in the design $(R \otimes C)^n$ is also present in design $(R \otimes C^n)$ (the converse may not be true). Thus $(R \otimes C)^n \preceq (R \otimes C^n)$. Now, since $C^n \preceq D$, by Proposition 4.7, $(R \otimes C^n) \preceq (R \otimes D)$. From the transitivity of $\preceq$, $(R \otimes C)^n \preceq (R \otimes D)$. ∎

This result implies that if a component is replaced with an n-delay-accumulating replacement, the same behavior of the composed design is obtained after a delay of n clock cycles after powering up the design.

Unfortunately, the compositionality result does not work for delay-preserving replacements, that is, if $C^n \preceq D^n$, then there might exist a composed design $R \otimes D$ such that $(R \otimes C)^n \not\preceq (R \otimes D)^n$. We can see this by the simple example in Figure 6. $(R \otimes C)^1$ consists of states $\{(r_0, c_1), (r_1, c_1)\}$ and $(R \otimes D)^1$ is $\{(r_1, d_0)\}$. Clearly $(R \otimes C)^1 \not\preceq (R \otimes D)^1$.

However, by Proposition 4.1, an n-delay-preserving replacement implies an n-delay-accumulating replacement. Thus if we replace a sub-design C by an n-delay-preserving replacement D, the entire design displays the same input/output behavior as the original after an extra n clock cycles delay after powering up.

What happens when two delay-replacements are made in different parts of the design? For safe replacement, it is easy to see that such design replacements do not cause problems:

Proposition 4.9 *If $Q \preceq R$ and $C \preceq D$, then $(Q \otimes C) \preceq (R \otimes D)$, for any composed design $R \otimes D$.*

Proof: From Proposition 4.7, we have $(Q \otimes C) \preceq (R \otimes C)$ as well as $(R \otimes C) \preceq (R \otimes D)$. The result follows from the transitivity of $\preceq$. ∎

For delay-accumulating replacements, it is fortuitous that for the composed design the total slack in just the largest of the individual replacement slacks (i.e. the individual slacks on the different components do not accumulate):

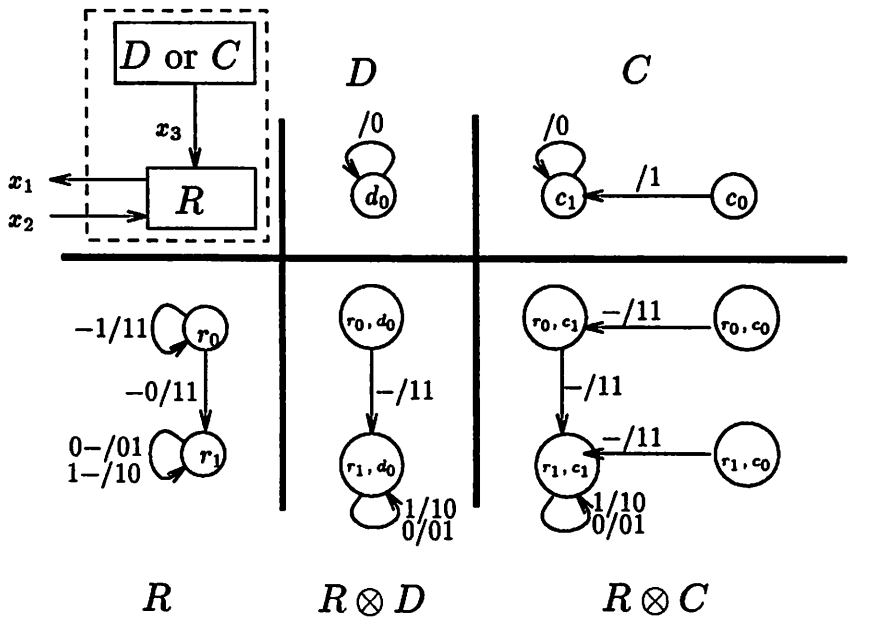

Figure 6: Delay-accumulating replacement does not preserve compositionality. The input and output wires for the composition are shown at the top left. The labels on R denote x_2x_3/x_1, those on C or D denote $/x_3$, and those on $(R \otimes D)$ or $(R \otimes C)$ denote x_2/x_1. Note that design D and C do not have any inputs.

Proposition 4.10 *If $Q^m \preceq R$ and $C^n \preceq D$, then for any composition $R \otimes D$, it is true that $(Q \otimes C)^p \preceq (R \otimes D)$, where $p = \max(m, n)$.*

Proof: Since each state in $(Q \otimes C)^p$ is also in $Q^p \otimes C^p$, then $(Q \otimes C)^p \preceq (Q^p \otimes C^p)$. Each state in Q^p is in Q^m and each state in C^p is in C^n; thus, $(Q^p \otimes C^p) \preceq (Q^m \otimes C^n)$. From Proposition 4.7, $(Q^m \otimes C^n) \preceq (R \otimes C^n)$, and similarly $(R \otimes C^n) \preceq (R \otimes D)$. Thus, the required result by the transitivity of $\preceq$. ■.

This means that delay-accumulating replacements can be made in different parts of the design, and the resulting overall design is as safe as weakest individual replacement (the replacement with the greatest slack). Since, delay-preserving implies delay-accumulating, the above result also holds for delay-preserving replacements.

5 Resynthesis for Delay-Preserving Replaceability

Here we describe how to extract the flexibility for delay-preserving replaceability in order to do multi-level resynthesis on sequential designs. The objective of this synthesis could be an area reduction or speed-up of the design. The experiments performed in Section 6 target area reduction. The methods described in the following are based on the multi-level resynthesis algorithm described in [PSAB94].

A sequential gate-level design can be viewed as a connection between a purely combinational part and memory elements. The inputs to the combinational part are the primary inputs to the design, denoted by $\vec{i}$, and the outputs of the memory elements are denoted by the present state vector $\vec{x}$. The outputs of the combinational part are the primary outputs of design are denoted by $\vec{o}$, and the inputs to the memory elements, denoted by the next state vector $\vec{y}$. For safe replaceability, [PSAB94] described a Boolean relation $\mathcal{R}(\vec{i}, \vec{x}, \vec{o}, \vec{y})$ such that any function $f : (\vec{i}, \vec{x}) \rightarrow (\vec{o}, \vec{y})$ which belongs to this relation is a safe replacement to the given design. Once this relation is obtained, multi-level resynthesis techniques can be used to exploit the flexibility afforded by this relation to optimize the given design.

To do resynthesis for delay-preserving replaceability, we describe how to obtain a relation $\mathcal{Q}(\vec{i}, \vec{x}, \vec{o}, \vec{y})$ which describes the flexibility for replacement. This can then be used in exactly the same way as in [PSAB94] to do multi-level synthesis for this relation.

First, we review the concept of *onion rings* [Jeo92, PSAB94]. Given a design D, the i-cycle delayed design D^i denotes the i-th onion ring of D. Suppose D has $(k+1)$ onion rings, i.e. for all $i \geq k$: $D^i = D^{i+1}$. We partition the state space of D into $(k+1)$ partitions, $B_0, B_1, \ldots, B_k$. For $0 \leq i \leq k-1$, B_i is defined as $D^i \setminus D^{i+1}$; also, B_k is defined as D^k (Figure 7). For example, for the design in Figure 1, $B_0 = \{111, 100, 001\}$, $B_1 = \{110\}$, $B_2 = \{010\}$, $B_3 = \{000, 101, 011\}$.

Clearly any state reachable from itself under some input sequence belongs to each D^i, and thus the innermost onion ring B_k. We will not alter the behavior of any such state (a "stable" state). All the flexibility for resynthesis comes from the states not reachable from themselves, the set of "transient" states, i.e. $D \setminus B_k$.

Recall that an n-delay preserving replacement allows the new design to have some flexibility over the first n clock cycles after the power-up, and after that it in indistinguishable from the original design. Based on how many cycles the design is allowed to stabilize, the designer supplies the slack n to be used for resynthesis— the higher n is, the greater the flexibility allowed for resynthesis, at the cost of a lower degree of safeness. A value of $n = 0$ implies safe replaceability. Resynthesis for

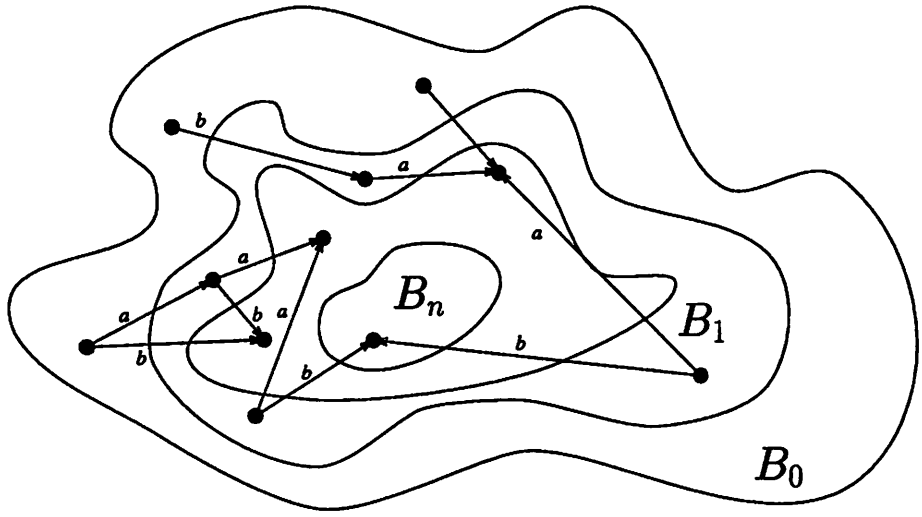

Figure 7: The onion ring structure for the design D. The bullets denote states in the state space of D, and the labels on the edges denote the input vector for the state transition.

this is described in [PSAB94].

For an n-delay-preserving replacement, we will express the flexibility to obtain a new design C such that $C^n = D^n$. Note, that this is a conservative algorithm, since only $C^n \preceq D^n$ is needed.

First, we informally describe the flexibility which will be specified later using the a Boolean relation $\mathcal{Q}$. We note here that techniques for using a Boolean relation to do multi-level synthesis [SB91] require the Boolean relation to be such that the starting design satisfies the relation. The behavior of the states in B_n (the "stable" states) is preserved. For $0 \leq i < n$, we allow a state in B_i to transition to any state in $\bigcup_{j=i+1}^{n} B_j$ on any input. Clearly, the original design satisfies this flexibility relation. Also, any design C that satisfies the relation satisfies $C^n = D^n$. Also note that for all states in $\bigcup_{i=0}^{n-1} B_i$, the outputs can be arbitrarily chosen without affecting C^n.

Formally the Boolean relation $\mathcal{Q}(\vec{i}, \vec{x}, \vec{o}, \vec{y})$ which characterizes this flexibility is:

$$\mathcal{Q}(\vec{i}, \vec{x}, \vec{o}, \vec{y}) = \Sigma_{i=0}^{n-1}[(\vec{x} \in B_i) \wedge (\vec{y} \in D^{i+1})] + [(\vec{x} \in B_n) \wedge (\vec{y} = \delta_D(\vec{x}, \vec{i})) \wedge (\vec{o} = \lambda_D(\vec{x}, \vec{i}))]$$

The intuition for the above relation $\mathcal{Q}$ is that given n, we choose to preserve the behavior of all states in the set B_n, i.e. states in this set have the same output and next-state functions as in the given design D. For states outside of B_n, if the state lies in B_i, we allow the next state of such a state to be any state in D^{i+1}. We do not care about the output from this state. All this ensures

that n cycles after power-up, the new design would be in a state in B_n and thus the new design would be an n-delay-preserving replacement for the original design.

Notice that for any integer $m \geq k$, where $(k+1)$ is the number of onion rings of the design D, the delayed design D^m is the same as D^∞, i.e. the set of stable states. Thus, the flexibility described by the relation $\mathcal{Q}$ for m-delayed replacement is the same as that for k-delayed replacement.

Once we have the Boolean relation $\mathcal{Q}(\vec{i}, \vec{x}, \vec{o}, \vec{y})$ we can use standard multi-level synthesis techniques to propagate this flexibility to individual nodes in the network and then minimize the networks [PSAB94, SB91].

6 Experiments

In this section we report experimental results using the algorithm described in Section 5 on ISCAS85 sequential circuits. We focused on the area reduction for n-delay-preserving replacements. We report results for various values of n.

The experimental results, obtained using a DECstation5900 are shown in Table 1. The starting circuits were obtained from ISCAS89 benchmark circuits with the SIS commands (`sweep; eliminate -1`) applied. These eliminate single-input and constant nodes and collapse nodes which do not fan out to more than one node. First we show the optimizations obtained by just doing the standard multi-level combinational optimization [SBT91] using observability don't cares (ODC) propagated to the individual nodes in the network. We also show the optimizations obtained by the safe replacement resynthesis method described in [PSAB94]. Then, we show the results of the method presented in this paper for n-delay preserving replacements for slacks of 1, 2, 5 and ∞. The table show that for many examples, significant additional optimizations are obtained by delaying "safeness" by using delay preserving replacements. Even for $n = 1$, we see good results for some example, e.g. `s386`, `s1488`, `s1494`. Also, in most cases the CPU times for n-delay preserving replacements are within an order of magnitude from the CPU time for combinational resynthesis. The much larger CPU times for pure safe replaceability can be attributed to the rigid conditions for safe replacements which require placing constraints on the outputs from the "transient" states as well as correlating the next states of "transient" states with the inputs [PSAB94]. Both these constraints are avoided by the resynthesis method presented in Section 5. This leads to a much smaller BDD to express the relation $\mathcal{Q}$ and thus, faster multi-level resynthesis.

We performed an experiment on one of the benchmark circuit with large number of onion rings

Ckt.	I/O/L	k	Initial size	ODC only		Safe replacement		n-delay-preserving replacement							
								$n=1$		$n=2$		$n=5$		$n=\infty$	
				r	time	r	time	r	time	r	time	r	time	r	time
s27	4/1/3	1	12	0	0.05	0	0.10	0	0.08						
s208	10/1/8	0	95	7	0.66										
s298	3/6/14	7	150	20	1.13	20	4.12	20	1.70	20	1.76	37	1.98	43	3.02
s349	9/11/15	7	160	6	2.24	6	316.54	6	3.58	12	5.76	12	11.44	15	21.17
s382	3/6/21	101	176	12	1.61	12	17.59	14	10.58	14	33.77	14	54.81	20	161.51
s386	7/7/6	1	204	17	1.46	31	2.34	77	2.04						
s400	3/6/21	101	184	22	1.57	22	18.57	24	11.04	24	36.48	24	57.52	30	163.88
s444	3/6/21	101	184	21	1.64	21	20.33	23	11.33	23	41.85	23	70.13	28	188.03
s510	19/7/6	4	280	1	4.26	1	9.94	1	5.09	1	5.19	1	5.51		
s526	3/6/21	667	283	43	3.07	43	15.33	43	6.54	45	9.85	46	11.00	95	66.30
s641	35/23/19	1	199	5	4.13	timeout		5	19.06						
s713	35/23/19	1	204	9	4.25	timeout		9	20.37						
s820	18/19/5	1	504	142	9.87	143	32.92	156	11.07						
s832	18/19/5	1	521	152	11.15	152	32.49	163	13.53						
s953	16/23/29	1	489	5	12.94	5	70.27	5	45.24						
s1196	14/14/18	2	618	18	78.71	18	1556.71	18	187.62	18	207.11				
s1238	14/14/18	2	690	65	107.12	65	1976.36	65	224.32	65	240.97				
s1488	8/19/6	1	813	57	23.54	57	102.59	116	22.80						
s1494	8/19/6	1	819	65	23.82	65	90.50	122	22.81						

Table 1: Experimental results. I/O/L denotes the number of inputs, outputs and latches. The number of onion rings for each design $(k+1)$. The initial size of the circuits and the savings r are in the number of literals. ODC only denotes optimization obtained only by combinational resynthesis. Since we know that for any integers $m \geq k$, we would get the same results for both m-delay replacement and k-delay replacement, the redundant experiments (denoted by blanks in the above table) were not performed.

n-delay-preserving replacement for s526; initial size = 283 literals		
n	reduction	time
1	43	5.93
100	54	22.47
200	58	29.24
300	70	34.52
400	69	38.60
500	69	44.05
600	72	48.78
667	95	68.14

Table 2: Safeness/flexibility tradeoff for s526; reduction is in number of literals

(s526) to explore the tradeoffs between flexbility and sacrificing "safeness". The results are in Table 2. The table shows that by allowing more delay n we do get additional flexibility. Also, the CPU times increase with higher values of n, partly because the time taken to compute the Boolean relation Q goes up with higher n.

In the experiments above, the initial nodes of the circuits have been collapsed minimally. We see that safe replaceability gives very marginal improvements over pure combinational reductions, and at a much larger CPU time cost. For this reason, it was suggested in [PSAB94] that we might get better use of the safe replaceability notion by using larger node sizes in the circuits. They increased the node sizes in these benchmark circuits by using SIS commands `eliminate 10` or `collapse`. For the sake of completeness, we ran our algorithm on these starting points also. Results are reported in Table 3. Once again, we see that using n-delay replacements instead of safe replacements allows us greater flexibility for resynthesis and gives better optimizations. Also, the CPU times are once again much better than those for safe replacements.

7 Conclusions

We presented the notions of delay-preserving and delay-accumulating replacement, which allow additional degree of flexibility over safe replacement by letting the design settle in for a few more clock cycles after power-up. The number can be controlled by the designer and specified to our synthesis tool, and then we can use the maximum flexibility while guaranteeing the degree of safeness

Ckt.	Initial size	ODC only		Safe replacement		n-delay-preserving replacement							
						$n=1$		$n=2$		$n=5$		$n=\infty$	
		r	time	r	time	r	time	r	time	r	time	r	time
s349	173	16	1.43	16	177.79	16	3.93	22	6.29	22	11.48	24	20.93
s386	205	53	1.22	67	2.61	107	2.08						
s444	236	65	1.76	65	17.25	67	11.61	67	41.15	67	63.91	70	185.09
s510	307	52	2.94	52	4.43	54	3.34	53	3.75	56	3.07		
s526	323	79	7.93	90	14.26	90	10.67	90	14.56	91	12.81	115	52.57
s713	285	83	5.59	timeout		83	28.16						
s832	470	110	4.64	119	17.93	138	8.23						
s953	700	98	20.42	103	51.78	110	42.51						
s1238	882	246	133.14	246	1060.94	246	412.69	246	429.11				
s1494	896	352	38.69	354	78.04	372	45.11						

Table 3: Experimental results with the starting netlist identical to that in [PSAB94].

the designer desires. We have shown experimental results to illustrate that we obtain significant optimizations at affordable CPU costs ny using our notions of replacement.

References

[Che93] 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.

[DD92] M. Damiani and G. De Micheli. Recurrence Equations and the Optimization of Synchronous Logic Circuits. In *Proc. of the Design Automation Conf.*, pages 556–561, June 1992.

[EC93] L. Entrena and K.-T. Cheng. Sequential Logic Optimization by Redundancy Addition and Removal. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 310–315, November 1993.

[Jeo92] Seh-Woong Jeong. *Binary Decision Diagrams and their Applications to Implicit Enumeration Techniques in Logic Synthesis.* PhD thesis, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309, 1992.

[MSBS91] 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.

[Pix92] 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.

[PSAB94] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton. Multi-level Synthesis for Safe Replaceability. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1994.

[SB91] H. Savoj and R. K. Brayton. Observability Relations and Observability Don't Cares. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 518–521, November 1991.

[SBT91] H. Savoj, R. K. Brayton, and H. Touati. Extracting Local Don't Cares for Network Optimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 514–517, November 1991.

[SP94] V. Singhal and C. Pixley. The Verification Problem for Safe Replaceability. In D. L. Dill, editor, *Proc. of the Conf. on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 311–323. Springer-Verlag, June 1994.