Latch Redundancy Removal Without Global Reset

Shaz Qadeer* **Robert K. Brayton**
Dept. of EECS
University of California at Berkeley
Berkeley, CA 94720
{shaz,brayton}@eecs.berkeley.edu

Vigyan Singhal
Cadence Berkeley Labs
1919 Addison St.
Berkeley, CA 94704
vigyan@cadence.com

Carl Pixley
Motorola Inc., MD F30B
5918 W. Courtyard Dr.
Austin, TX 78735
carl_pixley@email.mot.com

Abstract

For circuits where there may be latches with no reset line, we show how to replace some of them with combinational logic. All previous work in sequential optimization by latch removal assumes a designated initial state. Without this assumption, the design can power up in any state and earlier techniques are not applicable. We present an algorithm for identifying and replacing redundant latches by combinational logic such that no environment of the design can detect the change. The new design preserves the steady state behavior as well as all initializing sequences of the old design. We report experimental results on benchmark circuits and demonstrate savings in area without adverse impact on delay.

1 Introduction

Design replacement seeks to replace a given design with an optimized version so that the environment of the design cannot detect the replacement. For design replacement, where we pick arbitrary pieces of random logic from a large system and make a replacement, we cannot make any assumption about the environment (not even about the initializing sequence which the environment may use to initialize the design). It then becomes important to determine when a sequential design can be replaced for every possible environment. Most real designs have many latches. In this paper, we use "latch" to mean an edge-triggered latch or a flip-flop. Also, we are concerned only with replacing pieces of logic which are synchronous and where all latches are triggered by the same clock. which do not have a hardware reset line. We seek a theory of sequential replacement to handle such designs.

Many sequential synthesis and equivalence checking methods assume that 1) all latches have a global common reset line, and 2) this line is pulled on the first cycle of operation. We call this the DIS (designated initial state) assumption. Under this assumption, a design can safely replace another if the two global reset states are equivalent. Many sequential optimization techniques use don't cares resulting from unreachable states. Minimization of sequential logic [1], redundant latch removal [2], sequential redundancy removal [3] and equivalence net detection [4] rely on these don't cares. However, many practical designs avoid having a reset line to each latch. This implies that each such latch can non-deterministically power up either as a 0 or as a 1, leading to 2^t initial states, where t is the number of such latches. Hence none of these states can be considered unreachable and methods based on the DIS assumption will not find any flexibility.

These concerns led to a notion of design replacement, called safe replaceability [5], where the DIS assumption is not made. Loosely speaking, a safe replacement of a design preserves the steady–state behavior as well as all initializing sequences of the old design. Safe replacement requires that the new design be indistinguishable from the original from the first cycle after powering up. However, in most applications, the operation of a design does not begin instantaneously on power-up. Typically, many cycles (usually hundreds to thousands for digital ICs) occur before the operation begins. This allows additional flexibility for design replacement since the states that cannot be reached after the first n clock cycles, (for a given n) are of no interest. This notion of n-delay replacement has been used in [6, 7]. We use it also for identifying latch redundancy.

The latch removal problem has been tackled in [2, 8, 4] using the DIS assumption. Reachable states are computed and latches unnecessary for distinguishing between the reachable states are removed and replaced by combinational logic. In addition, the combinational logic used for generating the next state functions of those latches can be removed. We show that a sequential circuit can be optimized for area by removing redundant latches (even when all states are reachable) to produce a circuit which is a safe delayed replacement for the old one. Essentially, this means that the new design preserves the steady–state behavior of the old design and preserves each initializing sequence of the old design provided the sequence is applied after the new design has been allowed to clock for n cycles, where n is an input parameter for our optimization algorithm.

Another application of replacing latches by combinational logic is improving the effectiveness of FSM traversal, which is a crucial sub-step in many algorithms, especially those in formal verification [9]. For implicit FSM traversal using BDDs, the number of variables is directly affected by the number of latches. Selecting relatively smaller subcircuits and reducing the number of latches in it could be a good strategy for speeding up the verification of a large design. Under the DIS assumption, Ranjan et. al. [10] reduce the number of latches to make verification more tractable; our goal is similar but without making this assumption.

We first present an outline of the work on latch redundancy removal using the DIS assumption. We then present our algorithm based on safe replacement. Experimental results show that this approach is effective for area optimization and does not affect the circuit delay adversely. We also discuss implications to more efficient FSM traversal, and hence formal verification.

2 Preliminaries

Definition 1 *A* **deterministic Finite State Machine (DFSM)** M *is a quintuple,* $(Q, I, O, \lambda, \delta)$, *where* Q *is the set of states,* I *is the set of input values,* O *is the set of output values. The output function* $\lambda : (Q \times I) \to O$ *and the next state function* $\delta : (Q \times I) \to Q$ *are completely-specified functions.*

A hardware *design* D consists of a set of interconnected latches and gates. A design with n input wires, m output wires and t latches is naturally associated with a DFSM D. We will use D to also denote the set of states in the state space $\{0, 1\}^t$ (it will be

* Research supported by SRC under Contract No. 324-005

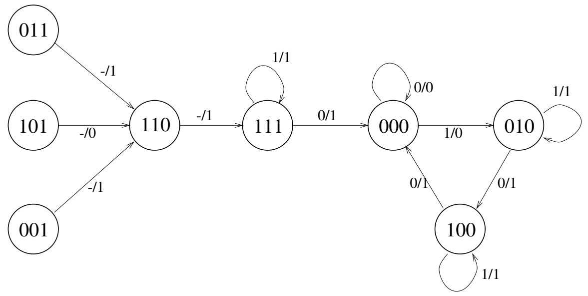

Figure 1: Design D.

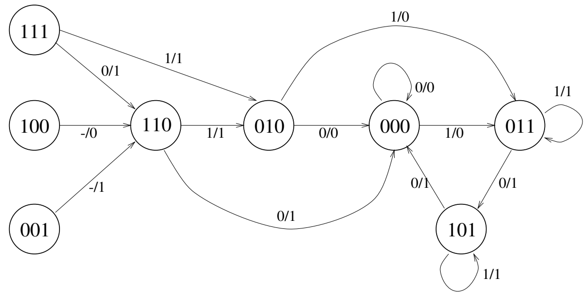

Figure 2: Example design A.

clear from the context if D refers to the design or the set of states). The input space $I_D = \{0,1\}^p$, and output space $O_D = \{0,1\}^m$, the next state function is $\delta_D : D \times I_D \to D$ and output function $\lambda_D : D \times I_D \to O_D$ are defined by the corresponding logic. δ_D andλ_D define the implicit state transition graph (STG) of the design. We are not assuming explicit set or reset pin to any latch; thus when the design powers up, it can non-deterministically power up in any of the 2^t states. If some of the latches have set or reset pins (either synchronous or asynchronous set or reset), they can easily be modeled by using latches without such pins and adding some additional logic.

We will also use λ_D and δ_D to denote the output and next state functions on sequences of inputs. So, if $\pi = a_1 \cdot a_2 \cdot a_3 \cdots a_q \in I_D^q$ is a sequence of q inputs, these functions are recursively defined as $\lambda_D(s, \pi) = \lambda_D(s, a_1) \cdot \lambda_D(\delta_D(s, a_1), \pi')$ and $\delta_D(s, \pi) = \delta_D(\delta_D(s, a_1), \pi')$, where $\pi' = a_2 \cdot a_3 \cdots a_p$.

Definition 2 *Given a design D, the n-***cycle delayed design** *(denoted by D^n) is the restriction of D to 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 if and only if there exist a power-up state s' in D and an input sequence of length n which drives s' to s. The* **outer-envelope** *of D, denoted by D^∞, is D^n where $D^n = D^m$ for any $m > n$.*

D^n is the 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 $m > n$.

For example, consider the design shown in Figure 1. For this design D, the various n-delayed designs are: $D^0 = D = \{011, 101, 001, 110, 111, 000, 010, 100\}$, $D^1 = \{110, 111, 000, 010, 100\}$, $D^2 = D^3 = \cdots = D^\infty = \{111, 000, 010, 100\}$.

For latch removal, we will use the following condition for replacement. This has earlier been used in [7, 6].

Definition 3 *Given a design D, a new design C is an n-***delay replacement** *for D (denoted by $C^n \preceq D$) if given any state $s \in C^n$ and any finite input sequence $\pi \in I^*$, there exists some state $t \in D$ such that the output behavior $\lambda_D(t, \pi) = \lambda_C(s, \pi)$, i.e. $\forall s \in C^n, \forall \pi \in I^*, \exists t \in D, \lambda_D(t, \pi) = \lambda_C(s, \pi)$.*

The intuition for this replacement is that, for a given n, C is a safe replacement for D as long as the operation of circuit C does not begin for at least n clock cycles after power-up. This is acceptable for most designs because we know that hundreds to thousands of clock cycles occur before operation begins. Such a number n can usually be determined from the system specification (for example, in a 20MHz design, a stabilizing time of 0.1ms amounts to 2000 clock cycles). Also, note that many commonly used synthesis operations, such as forward retiming [11, 12], cause delay replacements for a non-zero n.

Consider design A of Figure 2. It can be seen that design D of Figure 1 is a 1-delay replacement for design A (i.e. $D^1 \preceq A$) because it satisfies the condition in Definition 3 for all states in $D^1 = \{110, 111, 000, 010, 100\}$. However, design D is *not* a 0-delay replacement for design A because for the input sequence $0 \cdot 0 \cdot 0$, state $011 \in D^0$ outputs $1 \cdot 1 \cdot 1$ and no state in A shows this input-output behavior. In fact, $0 \cdot 0$ initializes A but does not initialize D^0; on the other hand, if D is an n-delay replacement for A, every initializing sequence for A is one for D^n [6].

Definition 4 *A* **terminal strongly connected component** *(tSCC) of a design is a minimal subset of states which is closed under all inputs.*

For example, the set of states $\{000, 010, 100\}$ constitute a tSCC of Design D in Figure 1. Notice that every tSCC of a design must be contained inside its outer-envelope.

3 Removing redundant latches under the *DIS* assumption

We briefly review the techniques, earlier presented in [2, 8, 4] for reducing latches in circuits assuming DIS.

Given a set of initial states S, extract the set of states reachable from S (see [1] for example). Let $C(\vec{x})$ be the characteristic function of the set of reachable states, where $\vec{x} = \{x_1, x_2, \ldots, x_t\}$ represent the t latches in the design. Since the states not in $C(\vec{x})$ are not reachable we do not care about their behavior. This allows some latches to be replaced with combinational logic. For example, suppose among all the states in $C(\vec{x})$, the value of the state bit x_j is a function of $(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_t)$. Then we can replace latch x_j by that combinational function. This replacement is guaranteed not to affect the behavior of the states in $C(\vec{x})$. Replacing a latch by combinational logic removes a latch as well as some combinational logic which was used only to drive this latch (see Figure 3).

The following two conditions must be satisfied to allow latches $\{x_{t'+1}, \ldots, x_t\}$ to be replaced by a function-set $F = (f_{t'+1}, \ldots, f_t)$ such that for any $i \in \{t'+1, \ldots, t\}$, $f_i : \{0,1\}^{t'} \to \{0,1\}$.

1. Latch redundancy condition. This ensures that for any $i \in \{t'+1, \ldots, t\}$, for every state in $C(\vec{x})$, the value of latch x_i is uniquely determined by the value of latches $\{x_1, \ldots, x_{t'}\}$:

$$\forall i \in \{t'+1, \ldots, t\} : \forall x_i \exists x_{i+1} \cdots \exists x_t C(\vec{x}) = 0 \quad (1)$$

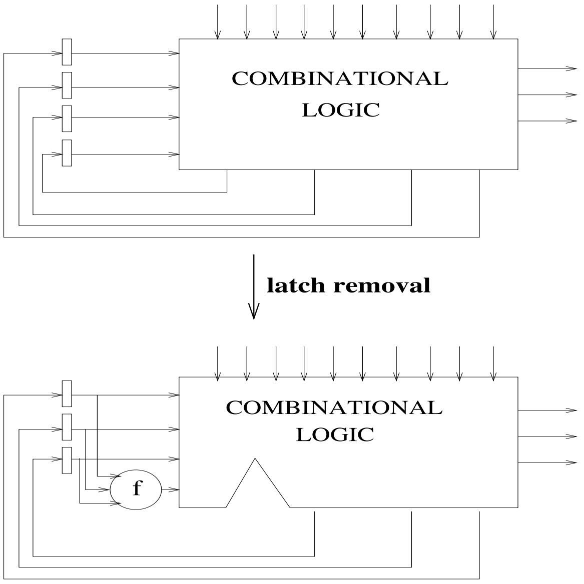

Figure 3: Replacing a latch with function f.

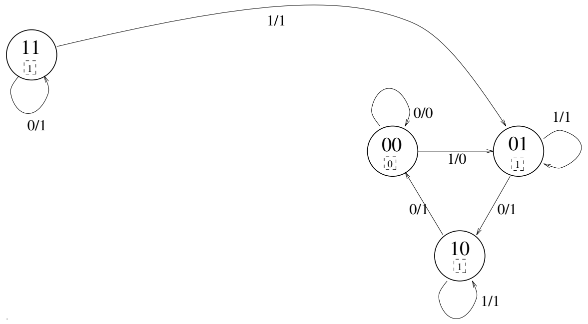

Figure 4: Design B. The value of $f_3 = x_1 + x_2$ is shown in the dotted box.

2. Function-set selection condition. This ensures that the function-set F is chosen so that for each state in $C(\vec{x})$, the values of the replaced latches are represented correctly:

$$(a_1, \ldots, a_t) \in C(\vec{x}) \Rightarrow \\ \forall i \in \{t'+1, \ldots, t\} : f_i(a_1, \ldots, a_{t'}) = a_i \qquad (2)$$

For an example, consider a design A_0 whose STG is identical to that of design A shown in Figure 2. A_0 is implemented by three latches $\{x_1, x_2, x_3\}$ whose encodings are shown in the figure; A_0 also has a designated state 000 (we emphasize that A_0 is a different design than A because it has an extra input line, the reset line, which, when asserted, sends the design to state 000). The set of reachable states from 000 is $C(\vec{x}) = \{000, 011, 101\}$. Using conditions 1 and 2 above, we can replace latch x_3 with the function $f_3 = x_1 + x_2$, and we get design B_0 whose STG[1] is identical to that of design B shown in Figure 4; the new designated initial state of design B_0 is 00. States 00, 01 and 10 are respectively equivalent to states 000, 011 and 101 in A_0. State 11 is not equivalent to any state in A_0 but its behavior is immaterial because the new design always starts in state 00, and can never reach state 11.

[1] In the next section we explain in more detail how the selection of x_3 and f_3 leads to this transition structure.

4 Removing redundant latches without the *DIS* assumption

It is known that a necessary condition for n-delay replacement is that every tSCC in the replacement design must be isomorphic to one in the original design [6]. To satisfy this necessary condition for safe replacement, we could try to choose a tSCC of the original design as $C(\vec{x})$ (to replace the notion of reachable states). In practice, most real designs have exactly one tSCC; all example circuits we have analyzed had one tSCC each. We now look at the naive extension of the algorithm in the previous section and see why it does not work if the design can power up in any state. The naive strategy for removing redundant latches would be to make a tSCC of the design play the role of $C(\vec{x})$ and then use exactly the same procedure, with conditions 1 and 2, for redundant latch identification and removal. Thus, we obtain design B (shown in Figure 4) as a replacement for design A in Figure 2. However, B is not an n-delay replacement of A for any n. This is because $B^n = B$ and $B \not\preceq A$. Observe that A can be reset from any state to state 000 under input sequence $0 \cdot 0$, but the same input sequence does not reset B even if we wait for arbitrary number of clock cycles after power-up (because B can remain in state 11 for an arbitrary number of cycles).

Henceforth, we use the term **core** to represent the set of states $C(\vec{x})$. A tSCC is one possible choice for the core; the choice for the core will be discussed in Section 4.3. For now, we only require that the core be closed under all inputs. Notice that design B failed to be a safe replacement of design A because state 11, the only state outside the core in B, had a self–loop on input 0. In general, the behavior of the states outside the core must be controlled to make sure that the new design is an n-delay replacement. This is a consequence of the absence of the DIS assumption. In our method, we preserve the behaviour of the states inside the core while ensuring that all states outside the core of the new design fall into the core after n clock cycles, no matter what the input sequence for those n clock cycles.

Suppose the original design is called A. There are two factors which determine the behavior of the new design B. First, the selection of latches to be replaced, $\hat{x} = (x_{t'+1}, \ldots, x_t)$, partitions the 2^t states in the original design into $2^{t'}$ equivalence classes, each with $2^{t-t'}$ states. Two states $\vec{a} = (a_1, \ldots, a_t)$ and $\vec{b} = (b_1, \ldots, b_t)$ lie in the same equivalence class if and only if for all $i \in \{1, \ldots, t'\}$: $a_i = b_i$. We denote the equivalence class of $\vec{a}$ by $\Omega(\vec{a})$. Each state of B represents an equivalence class of A, i.e. $\tilde{b} = (b_1, \ldots, b_{t'}) = (a_1, \ldots, a_{t'})$ represents $\Omega(\vec{a})$. Second, the selection of the function-set $F = (f_{t'+1}, \ldots, f_t)$ determines the representative state (of A) in each equivalence class, and the representative state determines the behavior of this class in the new design. Thus, if $f_{t'+1}(\tilde{a}) = c_1, f_{t'+2}(\tilde{a}) = c_2, f_t(\tilde{a}) = c_{t-t'}$, then $(a_1, \ldots, a_{t'}, c_1, \ldots c_{t-t'})$ is the representative state of $\Omega(\vec{a})$. Let $\phi(\tilde{a})$ denote the representative state of $\Omega(\vec{a})$. Now, for input $\vec{i}$, the next state of $\Omega(\vec{a})$ in B is $\delta_B(\tilde{a}, \vec{i}) = \Omega(\delta_A(\phi(\tilde{a}), \vec{i}))$ and the output is $\lambda_B(\tilde{a}, \vec{i}) = \lambda_A(\phi(\tilde{a}), \vec{i})$.

Notice that the set of replaced latches $(x_{t'+1}, \ldots, x_t)$ must satisfy the latch redundancy condition 1, where $C(\vec{x})$ is the core. Also the selection of the function-set F must satisfy the function-set selection condition 2. Based on the derivation of the behavior of the new design discussed above, it can be seen that if we replace latch x_3 in design A of Figure 2 with the function-set $F = (f_1)$ such that $f_1 = x_1 + x_2$ we get the new design shown in Figure 4 (for example, state 11 goes to state 11 on input 0 because $\delta_B(11, 0) =$

procedure compatible-set (**input:** $\mathcal{C}(\vec{x})$, $\{x_{t'+1}, \ldots, x_t\}$, $\mathcal{T}(\vec{x}, \vec{i}, \vec{y}), n)$ {
    $j \leftarrow 0$
    $\mathcal{S}(\vec{x}) \leftarrow \mathcal{C}(\vec{x})$
    $\mathcal{P}(\vec{x}) \leftarrow \exists x_{t'+1} \exists x_{t'+2} \cdots \exists x_t : [\mathcal{S}(\vec{x})]$
    while (($j < n$) **and** ($\mathcal{P}(\vec{x}) \neq 1$)) {
        $j \leftarrow j + 1$
        $\mathcal{P}(\vec{y}) \leftarrow (\mathcal{P}(\vec{x}))_{(\vec{y} \leftarrow \vec{x})}$
        $\mathcal{R}(\vec{x}) \leftarrow \forall \vec{i} \exists \vec{y} : [\mathcal{T}(\vec{x}, \vec{i}, \vec{y}) \wedge \mathcal{P}(\vec{y})]$
        if ($\mathcal{R}(\vec{x}) \cdot \overline{\mathcal{P}(\vec{x})} = 0$) **goto** *end*;
        $\mathcal{S}(\vec{x}) \leftarrow \mathcal{S}(\vec{x}) + \mathcal{R}(\vec{x}) \cdot \overline{\mathcal{P}(\vec{x})}$
        $\mathcal{P}(\vec{x}) \leftarrow \exists x_{t'+1} \exists x_{t'+2} \cdots \exists x_t : [\mathcal{S}(\vec{x})]$
    }
end: **if** ($\mathcal{P}(\vec{x}) = 1)\mathbf{return}\mathcal{S}(\vec{x})$
    else return FAIL
}

Figure 5: Procedure to obtain $\mathcal{S}(\vec{x})$ (input $\mathcal{C}(\vec{x})$ is the core set of states, $\mathcal{T}(\vec{x}, \vec{i}, \vec{y})$ is the transition relation of the design and n is the delay parameter for n-delay replacement)

$\Omega(\delta_A(\phi(11), 0)) = \Omega(\delta_A(111, 0)) = \Omega(110) = 11$). For this example, the representative elements can be read off by using the number in the dotted box in Figure 4; for example, $\phi(\Omega(100)) = \phi(\Omega(101)) = \phi(10) = 101$.

However, as previously discussed, the function replacement $F = (x_1 + x_2)$ is not an n-delay replacement for A. Thus we also need to regulate the behavior of equivalence classes which do not contain any state in the core of the original design. Notice that each state in the core is the representative element of its class, and the behavior of this class in the new design is equivalent to that of the core state in the original design.

We will satisfy the requirement for n-delay replacement by ensuring that if we wait for n clock cycles with arbitrary input vectors, we reach a state inside the core.

4.1 Choices for the function-set F

Definition 5 *Given a set of states S in $\{0,1\}^t$ (i.e. a set of states of the original design), a function-set $F = (f_{t'+1}, \ldots, f_t)$, where $f_i : \{0,1\}^{t'} \rightarrow \{0,1\}$ for all $i \in \{t'+1, \ldots, t\}$, is* **compatible** *with S if for any vector $\tilde{a} = (a_1, \ldots, a_{t'})$, $(a_1, \ldots, a_{t'}, f_{t'+1}(\tilde{a}), \ldots, f_t(\tilde{a})) \in S$.*

First assume that we have chosen the core set of states and a set of latches $(x_{t'+1}, \ldots, x_n)$ to be replaced (so that condition 1 is satisfied). Thus we already have a partition of equivalence classes for states in $\{0,1\}^t$. Now we will select a set of states S in $\{0,1\}^t$, and derive a function-set F compatible with S. We use the procedure in Figure 5 to obtain S (which is denoted by its characteristic function $\mathcal{S}(\vec{x})$). States in set S are candidates for representative elements of their respective equivalence classes. Each equivalence class has at least one state in S; equivalence classes whose states are in the core have exactly one state in S.

Theorem 4.1 *If the procedure "compatible-set" returns a set $\mathcal{S}(\vec{x})$, then there exists at least one function-set $F = (f_{t'+1}, \ldots, f_t) : \{0,1\}^{t'} \rightarrow \{0,1\}^{t-t'}$ compatible with $\mathcal{S}$. If we replace latches $(x_{t'+1}, \ldots, x_t)$ by F, then the new design is an n-delay replacement of the original design.*

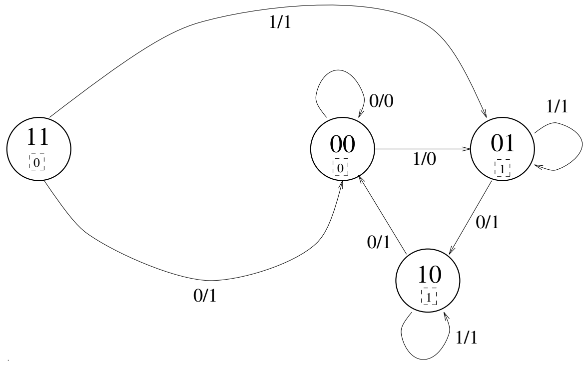

Figure 6: Design C. The value of $f_3 = x_1 \oplus x_2$ is shown in the dotted box.

Proof: First notice that the set $\mathcal{P}(\vec{x})$ is the set of all equivalence classes which have at least one member in $\mathcal{S}(\vec{x})$. Since the procedure returns a set $\mathcal{S}(\vec{x})$ only when $\mathcal{P}(\vec{x}) = 1$ we know that there exists at least one function-set which is compatible with $\mathcal{S}$.

Now, observe that if a state from an equivalence class is added to $\mathcal{S}(\vec{x})$ in iteration $j = p$, then for any iteration $i > p$, no state from this equivalence class is added to $\mathcal{S}(\vec{x})$. If one or more states from some equivalence classes are added in iteration $j = p$, we say that this equivalence class is covered in the p-th iteration. Suppose we obtain a new design by choosing a function-set which is compatible with $\mathcal{S}(\vec{x})$. Now consider any state in the new design. This state $\tilde{a}$ represents an equivalence class in the old design and derives its behavior from the representative state $\vec{a}$ of that equivalence class. Let this equivalence class be covered in the q-th iteration; thus $\vec{a}$ was added to $\mathcal{S}(\vec{x})$ in iteration $j = q$. Thus $\vec{a} \in \mathcal{R}(\vec{x})$ in iteration $j = q$, and for any input, the next state of $\vec{a}$ lies in an equivalence class which was covered in r-th iteration, for some $r < q$. Thus in the new design, for any input vector, state $\tilde{a}$ goes to some state which represents an equivalence class which was covered in an earlier iteration. Since we have at most $j = n$ iterations, it is clear that for any state $\tilde{a}$ in the new design, if we apply any arbitrary input vectors for n steps, we will reach a state $\tilde{b}$ representing an equivalence class which was covered in iteration $j = 0$. Since iteration $j = 0$ only covers the the equivalence classes for the core states, $\tilde{b}$ is equivalent to a state in the core of the old design. Thus, the new design is an n-delay replacement of the old one. ■

As an example, start with design A in Figure 2 and choose the set $\{000, 011, 101\}$ as the core and x_3 as the latch to be replaced. The procedure *compatible-set* returns the set $\mathcal{S}(\vec{x}) = \{000, 011, 101, 110\}$. $F = (f_3)$, where $f_3 = x_1 \oplus x_2$, is the only function set compatible with this. The resulting design C (shown in Figure 6) is a 1-delay replacement of A.

4.2 Selecting an Optimal function-set F

If the procedure *compatible-set* returns a set S, any function-set F compatible with S is allowed. However, our overall goal is to optimize the area of the original circuit, so we would like to choose an F which has the smallest area. In fact, if we cannot find F within a certain bound, the saving obtained by removing the redundant latches may be lost(refer once again to Figure 3 for the tradeoff in area saving).

Selecting the function-set F is equivalent to determining a multi-level network with t' inputs and $(t - t')$ outputs so that the network

procedure greedy-function-set (**input:** $\mathcal{S}(\tilde{x}, \hat{x})$, $\{x_1, \ldots, x_{t'}\}, \{x_{t'+1}, \ldots, x_t\}$) {
    for $j = t'$ **to** t **do** {
        $\mathcal{O}(\tilde{x}) \leftarrow \exists x_{t'+1} \cdots \exists x_t : [\mathcal{S}(\tilde{x}, \hat{x}) \cdot (x_j = 1)]$
        /* $\mathcal{O}(\tilde{x})$ is the on set for f_j */
        $\mathcal{D}(\tilde{x}) \leftarrow \forall x_j \exists x_{t'+1} \cdots \exists x_{j-1} \exists x_{j+1} \cdots$
            $\cdots x_t : [\mathcal{S}(\tilde{x}, \hat{x})]$
        /* $\mathcal{D}(\tilde{x})$ is the don't-care set for f_j */
        $f_j(\tilde{x}) \leftarrow$ bdd-minimize$(\mathcal{O}(\tilde{x}), \mathcal{D}(\tilde{x}))$
        $\mathcal{S}(\tilde{x}, \hat{x}) \leftarrow \mathcal{S}(\tilde{x}, \hat{x}) \cdot (x_j = f_j(\tilde{x}))$
    }
    $F \leftarrow (f_{t'+1}, \ldots, f_t)$
    return F
}

Figure 7: Procedure to obtain a function-set

is compatible with $S(\tilde{x}, \hat{x})$ where $\tilde{x} = \{x_1, \ldots, x_{t'}\}$ and $\hat{x} = \{x_{t'+1}, \ldots, x_t\}$. Notice, that this is the problem of obtaining a multi-level network compatible with a Boolean relation in terms of its inputs and outputs. This can be solved in two steps. First, we obtain any arbitrary multi-level network representing some F which is compatible with S. Then, one approach would be to use the techniques in [13] to optimize this network maintaining compatibility with S.

However, our experiments (presented later) show that the size of F is relatively small compared to the given circuit sizes and that the greedy approach described by procedure *greedy-function-set* in Figure 7 worked well. We order the latches in an arbitrary order. For each latch to be replaced, we find the on-set and don't care set for the replacement function. Then we use a heuristic to find a function which is compatible with this on-set and don't care set. Since all the Boolean quantities are represented as BDDs, we use a BDD minimization procedure, which heuristically finds an implementation which has the smallest support (we use a simpler version of the algorithm in [14]); alternately, we could have used *bdd-generalized-cofactor* (due to [15]) or any of many other BDD minimization algorithms with respect to a don't care set (see [16]). The minimized BDD is converted to a similar looking network whose nodes are multiplexors controlled by the variable of the corresponding BDD node and then the SIS commands `collapse` and `full_simplify` are run on the network. Since we are dealing with very small functions, the choice of the heuristic for BDD minimization does not matter. Once we find a function replacement for a latch we restrict the compatible set S by this function.

4.3 Choosing the core

Before selecting the function-set F, we need to choose a core to identify a set of latches that satisfy condition 1. Recall that a core is a set of states closed under all inputs. Obviously, if a set of latches satisfies condition 1 for a choice of core, it also satisfies the condition for any subset of the core. As argued earlier, most real designs have a single tSCC, and for such designs each choice of the core must be a superset of the tSCC. Thus, it seems that choosing the tSCC as the core should be the obvious choice since it allows us most flexibility in choosing the set of redundant latches. However, this choice may not be the right one for the following reason.

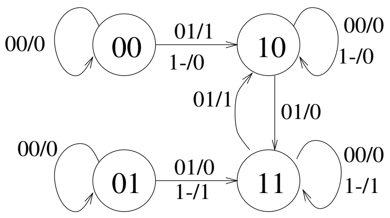

Figure 8: Design E

Consider design D in Figure 1. Suppose we choose the tSCC $\{000, 010, 100\}$ as the core. We find that latch x_3 is redundant. The procedure *compatible-set* identifies candidates for representing their equivalence classes (000, 010 and 100 are already representatives for their respective equivalence classes 00, 01 and 10; we need to find candidates to represent the equivalence class 11). We observe that for all inputs, neither of the states 110 or 111 goes to equivalence class 00, 10 or 01 (the value of $\mathcal{P}(\vec{x})$ at iteration $j = 0$). This means that $(\mathcal{R}(\vec{x}) \cdot \overline{\mathcal{P}(\vec{x})} = 0)$ and the procedure returns FAIL. Thus, we are unable to replace latch x_3. On the other hand, if we chose the outer envelope of the design (D^∞) $\{000, 010, 100, 111\}$ latch x_3 could be replaced by the function $f_3 = x_1 \cdot x_2$ since the procedure *compatible-set* returns $\mathcal{S}(\vec{x}) = \{000, 010, 100, 111\}$. In fact, we can prove that if we choose the outer-envelope of a design D as the core the procedure *compatible-set* never returns FAIL for any $n \geq m$, where m is such that $D^\infty = D^m$.

Theorem 4.2 *If we set $\mathcal{C}(\vec{x})$ to be the set of states in D^k, the k-cycle delayed design, and set n to be greater than or equal to k, then the procedure "compatible-set" does not return FAIL.*

Proof: We will prove the above by showing that after the **while** loop terminates, $\mathcal{P}(\vec{x}) = 1$. We show that by proving the claim stated below. In the following, we say that state $\vec{a}$ lies in the l-th onion ring if and only if $\vec{a} \in Q_{D^{k-l}} \setminus Q_{D^{k-l+1}}$ for $l \in \{1, 2, \ldots, k\}$; if $\vec{a} \in Q_{D^k}$, we say that $\vec{a}$ lies in the 0-th onion ring. It should be obvious that for any l such that $0 < l \leq k$, for any arbitrary input vector, any state in the l-th onion ring goes to a state which lies in the p-th onion ring for some $p < l$.
Claim: If state $\vec{a}(a_1, \ldots, a_t)$ belongs to the l-th onion ring, then after iteration $j = l$, $\mathcal{P}(\vec{x})$ contains $\tilde{a} = (a_1, \ldots, a_{t'})$.
We prove this claim by induction.
Base case. ($l = 0$): Since $\vec{a}$ belongs to D^k, it belongs to $\mathcal{C}(\vec{x})$ and hence $\tilde{a}$ belongs to $\mathcal{P}(\vec{x})$ when it is initialized at $j = 0$.
Induction step. Assume that the claim is true for any state which lies in the p-th onion-ring for some $p < l$. We will show prove the induction step for a state $\vec{a}$ which belongs to the l-th onion ring. Consider iteration $j = l$. Either, $\mathcal{P}(\vec{x})$ already contains $\tilde{a}$ before this iteration, in which case we are done. Or else, $\mathcal{P}(\vec{x})$ does not contain $\tilde{a}$ before this iteration. However, we know that for any arbitrary input vector, $\vec{a}$ transitions to a state $\vec{b}$ which lies in the p-th onion ring for some $p < l$. By our induction hypothesis, $\tilde{b}$ belongs to $\mathcal{P}(\vec{x})$ from the previous iteration. Thus, $\vec{a}$ must belong to the set $\mathcal{R}(\vec{x})$ computed at $j = l$. Hence, $\tilde{a}$ belongs to $\mathcal{P}(\vec{x})$ after iteration $j = l$. ∎

This theorem shows that if the outer envelope is chosen as the core and there is a non-empty set of redundant latches, then those latches can be removed. On the other hand, in the following example, choosing the tSCC as the core lets us replace a latch, but no latches can be removed if the outer-envelope is chosen. Consider the design E in Figure 8.

procedure latch-replacement (**input:**
network in terms of $(\vec{x}, \vec{i}, \vec{y}, \vec{o})$) {
  /* let y_j denote the BDD for the j-th next state variable */
  /* let o_k denote the BDD for the k-th output variable */
  for (each node p in the network)
    Compute a BDD $\mathcal{S}_p(\vec{x}, \vec{i})$ representing node p
    in terms of $(\vec{x}, \vec{i})$
  $\mathcal{T}(\vec{x}, \vec{i}, \vec{y}) \leftarrow (\prod_{j=1}^{t}(y_j \equiv \mathcal{S}_{y_j}(\vec{i}, \vec{x})))$
  $\mathcal{C}(\vec{x}) \leftarrow$ tSCC($\mathcal{T}(\vec{i}, \vec{x}, \vec{y})$)
  $\hat{x} \leftarrow \emptyset$; $\mathcal{B}(\vec{x}) \leftarrow \mathcal{C}(\vec{x})$
  foreach (latch x) **do**
    if ($\forall x : [\mathcal{B}(\vec{x})] = 0$) {
      $\hat{x} \leftarrow \hat{x} \cup \{x\}$
      $\mathcal{B}(\vec{x}) \leftarrow \exists x : [\mathcal{B}(\vec{x})]$
    }
  $\tilde{x} \leftarrow \vec{x} \setminus \hat{x}$
  $\mathcal{A}(\vec{x}) \leftarrow$ compatible-set($\mathcal{C}(\vec{x}), \hat{x}, \mathcal{T}(\vec{i}, \vec{x}, \vec{y}), \infty$)
  Let $n \leftarrow j$, if the loop in compatible-set()
  terminates in the j-th iteration
  $F \leftarrow$ greedy-function-set($\mathcal{A}(\vec{x}), \tilde{x}, \hat{x}$)
  Replace latches $\hat{x}$ by an implementation of F
  Recursively sweep away dead logic which doesn't
  fan out anywhere
  return (optimized network, n)
}

Figure 9: Procedure to replace latches with logic

If we choose the tSCC $\{10, 11\}$ as the core, it can be seen that we can replace latch x_1 by the function $f_1 = 1$. However, if we choose the outer-envelope $\{00, 01, 10, 11\}$, no latch is redundant.

In our experiments, we first choose a tSCC as the core. A tSCC is obtained by successively pruning the state space and performing forward and backward reachability computations from a state picked at random from the unpruned state space. If the state picked randomly belongs to some tSCC then the forward reachable set (which will be a tSCC) will be contained in the backward reachable set; otherwise the whole backward reachable set can be pruned away. In the worst case the forward or backward reachability computations can require iterations exponential in the number of state bits; however, we did not see this kind of blow-up in our experiments. If we are not able to replace any latches, we choose the outer envelope as the core. In fact, for all except four examples, we were able to successfully use the tSCC as the core. For `s344`, `s349`, `tuctl`, and `vactl` (see Table 1), *compatible-set* returned FAIL, in which case we used the outer-envelope as the core.

4.4 Selecting the set of redundant latches

Given a set $\mathcal{C}(\vec{x})$, finding the maximum set of redundant latches is an NP-complete problem. Most algorithms for solving this problem heuristically generate an ordering of the latches. The latches are tested for redundancy and removed in that order. The heuristic used by Lin and Newton [8] (to maximize the number of latches chosen as redundant), orders latches by decreasing unateness. The unateness of a variable x_i is the absolute value of the difference between the number of minterms in $\mathcal{C}(\vec{x})_{x_i=0}$ and $\mathcal{C}(\vec{x})_{x_i=1}$. The intuition is that the more unate a variable is, the less it contributes to distinguishing the states in $\mathcal{C}(\vec{x})$. We use this heuristic and then select the redundant latches by going through this order and adding a variable to the set if the current set still satisfies Condition 1.

5 Experiments

Our entire algorithm for replacing latches with combinational logic is shown in Figure 9.

We used ∞ as the value of n to pass to *compatible-set* because in all examples, the **while** loop terminated in less than 1000 iterations. Thus, we obtain n-delay replacements where $n < 1000$. It is usually safe to assume that more than 1000 cycles are allowed before operation starts (for a 20MHz design, 1000 clock cycles amount to just 0.05ms). On the other hand, the parameter n could be easily specified as an input to the procedure.

We implemented this algorithm in SIS [17] and experimented with ISCAS89 circuits. All the experiments were performed on a DEC workstation based on Alpha21064. To make our BDD-based algorithm tractable we chose circuits with fewer than 30 latches. However, notice that since our replacement criterion permits us to apply our method to any arbitrary sub-circuit of a design, we can optimize large designs by picking and optimizing random pieces of logic from them. We also manually extracted four random sub-circuits from an industrial digital design to study the efficacy of our procedure for the methodology of replacing arbitrary pieces of sequential logic chosen from a big design. The results appear in Table 1 (the last four circuits are the industrial designs).

We preprocessed all circuits with SIS commands: `sweep; eliminate -1`. For circuits where the number of states in the tSCC is more than half the number of total states (2^L), e.g. `s27`, no latch can be removed; so we did not process them any further. Notice that for most circuits the time for tSCC computation dominates the total CPU time. On the other hand if we had used the outer-envelope (OE) as the core, significantly lower CPU times would result and only the results for `s641` and `s713` may have been affected; however, the CPU times are reasonable anyway, so we worked with tSCC as the first choice for core. We can see from the table that for many circuits we were successful in removing some latches, and all final circuits were n-delay replacements of the respective original circuits, with n no more than 667. Notice also that for many circuits the total final size[2] (of the combinational part of the circuit) is smaller than the original size because the addition of F was more than offset by the removal of the fan-in logic of the latches (Figure 3).

Table 2 shows the effect of our algorithm on the final implementation of the circuits. Here we took the final circuits from Table 1 (only those where latch removal worked) and ran the SIS script `script.rugged`[3] (to do technology independent optimization) followed by a technology mapping step using the MCNC91 library. We see that the mapped areas are on average 10.8% less than the areas obtained without latch removal. Note that the CPU time with latch replacement includes the time taken to remove redundant latches. It is worth noting that the total CPU time with latch removal is actually lower than the total CPU time without this option! This is probably because logic optimization becomes simpler once we have fewer latches (and thus fewer state variables). In particular, reachability computations become much faster.

So far we have removed latches and replaced them by a combinational function of other latches. It is conceivable that such an operation could increase the delay of the paths between the latches.

[2] In this table, circuit sizes are expressed in number of literals in factored form (see [18]).

[3] In the last step of `script.rugged`, we used the sequential optimization procedure from [6] instead of `full_simplify` because we wanted to exploit the sequential flexibility in addition to the combinational flexibility.

Ckt.	PI	PO	L	Computation of core				Orig. size	Final circuit				
				OE		tSCC			Latches removed	Delay n	Size of F	Total size	Time
				size	time	size	time						
s27	4	1	3	6	0.02	6	0.03	–	–	–	–	–	–
s208	10	1	8	256	0.07	256	2.58	–	–	–	–	–	–
s298	3	6	14	218	0.29	218	0.91	150	2	7	4	130	1.26
s344	9	11	15	5342	0.59	1487	1.62	156	0	–	–	156	5.20
s349	9	11	15	5342	0.62	1487	1.70	160	0	–	–	160	5.33
s382	3	6	21	8864	16.50	8864	26.72	176	3	101	6	160	35.08
s386	7	7	6	13	0.14	13	0.20	204	0	–	–	204	0.21
s400	3	6	21	8864	15.07	8864	24.90	184	3	101	6	166	33.32
s420	18	1	16	65536	0.18	65536	1666.67	–	–	–	–	–	–
s444	3	6	21	8864	15.41	8864	25.78	184	3	101	6	166	33.09
s526	3	6	21	8868	14.28	8868	22.02	283	2	667	4	263	44.63
s641	35	23	19	6461	1.65	1544	4.05	199	5	2	9	172	3.81
s713	35	23	19	6461	1.67	1544	3.88	204	5	2	9	175	4.33
s820	18	19	5	25	0.33	25	0.44	–	–	–	–	–	–
s832	18	19	5	25	0.34	25	0.54	–	–	–	–	–	–
s953	16	23	29	504	1.24	504	10.01	489	9	1	96	522	12.77
s1196	14	14	18	2615	2.49	2615	11.54	618	0	–	–	618	11.80
s1238	14	14	18	2615	2.63	2615	8.93	690	0	–	–	690	9.14
s1488	8	19	6	48	0.65	48	0.87	–	–	–	–	–	–
s1494	8	19	6	48	0.69	48	0.97	–	–	–	–	–	–
tuctl	17	11	13	508	0.55	451	2.09	101	0	–	–	101	3.02
idmux	17	17	8	42	0.14	42	0.20	64	1	2	2	66	0.27
vactl	17	11	21	2270	0.50	2126	3.55	105	6	4	15	96	4.93
ofmux	17	17	15	162	0.19	162	0.35	64	4	2	7	59	0.52

Table 1: Experimental results for latch replacement. PI, PO and L denote the number of inputs, outputs and latches respectively.

Ckt.	With latch replacement			No latch replacement			Ratios	
	Mapped area	Delay	CPU time	Mapped area	Delay	CPU time	Area	Delay
s298	278	17.6	8.49	312	19.1	8.83	0.891	0.921
s382	428	21.6	55.62	472	21.6	168.30	0.907	1.000
s400	419	27.0	53.76	460	26.9	162.71	0.911	1.004
s444	419	27.0	57.10	458	27.0	178.67	0.915	1.000
s526	456	22.4	108.22	460	18.0	73.24	0.991	1.245
s641	378	28.1	30.29	460	36.5	34.91	0.822	0.770
s713	377	27.9	31.10	460	36.5	42.04	0.820	0.764
s953	940	22.8	70.68	1012	20.8	75.85	0.929	1.096
idmux	174	15.4	4.84	182	15.4	5.22	0.956	1.000
vactl	284	14.3	14.53	323	14.3	14.93	0.879	1.000
ofmux	203	14.3	5.13	252	16.6	6.21	0.806	0.861

Table 2: Experimental results after technology mapping.

This intuition is based on the observation that a combinational path between a replaced latch and some other latch may become longer due to the intervening replacement logic. However, using latch removal, the longest static path delays of the mapped circuits in Table 2 are on average 4.0% improved over those without replacement. The only circuit where latch replacement seems to have caused longer delay is `s526`.

These results are very encouraging since they demonstrate reduction in the number of latches, in the size of mapped circuits and even in the delay for the circuits.

6 Conclusions

We presented a method for replacing latches with combinational logic for circuits where each latch need not have a global reset line. Our experiments indicate that our method is very effective in reducing the area of circuits and does not affect the performance (delay) adversely.

Reducing the number of latches should be useful also for algorithms which require state traversal, for example, formal verification via symbolic model checking [9].

The replacement condition allows optimizations on random pieces of sequential logic allowing applications to large circuits by making such optimizations on small subcircuits. We plan to pursue this aspect of our method in the future.

References

[1] 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*, (Santa Clara, CA), pp. 414–417, Nov. 1990.

[2] C. Berthet, O. Coudert, and J. C. Madre, "New Ideas on Symbolic Manipulation of Finite State Machines," in *Proc. Intl. Conf. on Computer Design*, (Cambridge, MA), pp. 224–227, Oct. 1990.

[3] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy Identification and Removal Based on Implicit State Enumeration," in *Proc. Intl. Conf. on Computer Design*, (Cambridge, MA), pp. 77–80, Oct. 1991.

[4] G. Berry and H. J. Touati, "Optimized Controller Synthesis Using Esterel," in *Workshop Notes of Intl. Workshop on Logic Synthesis*, (Tahoe City, CA), May 1993.

[5] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton, "Multi-level Synthesis for Safe Replaceability," in *Proc. Intl. Conf. on Computer-Aided Design*, (San Jose, CA), pp. 442–449, Nov. 1994.

[6] V. Singhal, C. Pixley, A. Aziz, and R. K. Brayton, "Exploiting Power-up Delay for Sequential Optimization," in *Proc. European Design Automation Conf.*, (Brighton, Great Britain), pp. 54–59, Sept. 1995.

[7] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying Sequential Redundancies Without Search," in *Proc. of the Design Automation Conf.*, (Las Vegas, NV), June 1996.

[8] B. Lin and A. R. Newton, "Exact Redundant State Registers Removal Based on Binary Decision Diagrams," in *Proc. of the MCNC Intl. Workshop on Logic Synthesis*, vol. 1, May 1991.

[9] K. L. McMillan, *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[10] R. K. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R. K. Brayton, "Efficient Formal Design Verification: Data Structure + Algorithms," in *Workshop Notes of the Intl. Workshop on Logic Synthesis*, (Tahoe City, CA), May 1995.

[11] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly, "On Test Set Preservation of Retimed Circuits," in *Proc. of the Design Automation Conf.*, (San Francisco, CA), pp. 176–182, June 1995.

[12] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton, "The Validity of Retiming Sequential Circuits," in *Proc. of the Design Automation Conf.*, (San Francisco, CA), pp. 316–321, June 1995.

[13] H. Savoj and R. K. Brayton, "Observability Relations and Observability Don't Cares," in *Proc. Intl. Conf. on Computer-Aided Design*, (Santa Clara, CA), pp. 518–521, Nov. 1991.

[14] B. Lin, "Efficient Symbolic Support Manipulation," in *Proc. Intl. Conf. on Computer Design*, (Cambridge, MA), pp. 513–516, Oct. 1993.

[15] O. Coudert and J. C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits," in *Proc. Intl. Conf. on Computer-Aided Design*, (Santa Clara, CA), pp. 126–129, Nov. 1990.

[16] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic Minimization of BDDs Using Don't Cares," in *Proc. Intl. Conf. on Computer-Aided Design*, (San Diego, CA), pp. 225–231, June 1994.

[17] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. Intl. Conf. on Computer Design*, (Cambridge, MA), pp. 328–333, Oct. 1992.

[18] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proceedings of the IEEE*, vol. 78, pp. 264–300, Feb. 1990.