USING COMBINATIONAL VERIFICATION FOR SEQUENTIAL CIRCUITS

by

Rajeev K. Ranjan, Vigyan Singhal, Fabio Somenzi,
and Robert K. Brayton

Memorandum No. UCB/ERL M97/77

10 October 1997

USING COMBINATIONAL VERIFICATION FOR SEQUENTIAL CIRCUITS

by

Rajeev K. Ranjan, Vigyan Singhal, Fabio Somenzi,
and Robert K. Brayton

Memorandum No. UCB/ERL M97/77

10 October 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Using Combinational Verification for Sequential Circuits

Rajeev K. Ranjan* Vigyan Singhal† Fabio Somenzi‡‡ Robert K. Brayton*

Abstract

Retiming combined with combinational optimization is a powerful sequential synthesis method. However, this methodology has not found wide application because formal verification of sequential synthesis is not practical and current simulation technology requires the correspondence of latches for ease in the detection of errors. We present a practical verification technique which permits such sequential synthesis for a class of circuits. In particular, we require certain constraints to be met on the feedback paths of the latches involved in the retiming process. For a general circuit, we can satisfy these constraints by fixing the location of some latches, e.g., by making them observable. We show that implementation verification after performing repeated retiming and synthesis on this class of circuit reduces to a combinational verification problem. We also demonstrate that our methodology covers a large class of circuits by applying it to a set of benchmarks and industrial designs.

1 Introduction

In combinational synthesis [1, 2], the positions of the latches are fixed and the logic is optimized. In retiming [3, 4], the latches are moved across fixed combinational gates. The effects of retiming are – changes in the number of latches (thereby leading to increase/decrease in area) and increase/decrease in the cycle time (leading to slower/faster clock rate). A side effect of retiming is that it enables interaction between different combinational logic blocks. Hence retiming followed by combinational synthesis enables logic optimization which would not have been possible by combinational optimization alone. Combinational synthesis generates new possibilities for the latch locations perhaps leading to further optimization. A sequence of retiming and combinational resynthesis steps can provide powerful optimization of a sequential circuit. After the initial algorithm proposed in [4] for a simple circuit containing single clock edge-triggered latches, many advancements were made in terms of efficient implementation and applicability of retiming with more complex memory elements. In particular, techniques given in [5, 6] can be applied to large sequential circuits. Retiming level-sensitive latches was addressed in [7, 8]. Recently, Legl *et al.* proposed retiming techniques for edge-triggered circuits with multiple clocks and load enables [9]. They introduced the notion of a latch class $cl = (CLK, LE)$, which is all latches connected to clock signal CLK and load signal LE. The retiming problem for multiple-class sequential circuits was reduced to an equivalent retiming for single class sequential circuits, thereby exploiting performance enhancements made in that domain. Since most of the industrial designs contain latches with different load signals and multiple clocks, their technique further improves the applicability of retiming to such designs.

Retiming and resynthesis, though less powerful in theory than full sequential optimization (based on unreachable states, input/output don't care sequences), covers a wide part of the optimization space. However, this technique has not had much success in obtaining a place in traditional synthesis methodology. One of the main bottlenecks has been the lack of efficient verification tools to verify the functionality of

*Department of EECS, University of California, Berkeley, CA 94720
†Cadence Berkeley Labs, Berkeley, 94701
‡Dept. of ECE, University of Colorado, Boulder

the optimized design. The verification complexity of a retimed and resynthesized design is not formally known. It is conjectured to be harder than the NP-hard class of problems. On the other hand, the verification problem for combinational logic optimization is a relatively easier problem in practice. Much work has gone into combining structural and functional techniques to obtain verification algorithms that can deal with reasonably large industrial circuits [10, 11, 12].

We propose a methodology which reduces a sequential verification problem into an equivalent combinational verification problem for a class of circuits. This allows exploitation of the advancements made in the field of combinational verification and use its powerful techniques to perform verification. Our method requires that for each latch with a feedback path, its next state function should be positive unate in the latch variable. Later we will show that the scope of this methodology allows i) the presence of self-loops on latches, ii) pipelined circuits where the latches cannot be retimed to periphery, iii) presence of latches trapped inside a combinational block iv) circuits with load-enabled latches and v) circuits where latches conditionally update their contents. Typically the industrial designs consists of two kinds of latches. The first kind constitute small finite state machines. Each such state machines are strongly connected. These machines interact with each other via the acyclic network of latches of second kind. In general, designers want to preserve the locations of the latches that hold the states of FSM (the first kind), since they want to monitor simulation results. Fixing some latch locations breaks the feedback paths and as a result, the circuit might satisfy our constraint. In case the given circuit still fails our constraint, we expose a minimum number of latches making their locations fixed. Then we perform retiming and resynthesis optimizations on the modified circuit. In practice, our approach does not incur significant optimization penalty due to this modification.

The outline of the paper is as follows. In Section 2 we present previous research in the area of sequential verification. We establish the notation, terminology, and our notion of equivalence in Section 3. In Section 4 we describe the basic idea behind our work and give appropriate definitions. In Section 5 we discuss our technique for a circuit with no feedback latches. In Section 6 we present the extension to include circuits containing feedback latches. The details of the experimental setup and results are given in Sections 7 and 8 respectively.

2 Previous Work

Many researchers have investigated the problem of sequential equivalence checking and in particular verification of retimed circuits. A popular approach is to compose the machines together and traverse the state space of the product machine. Explicit state enumeration techniques perform an explicit traversal of the state space. Due to the explicit nature of this technique, it is limited to only a small number of state elements. Symbolic techniques [13, 14] perform the state space traversal of the product machine. In these approaches the circuit is modeled as a finite state machine and the outputs are evaluated as functions of present state and primary inputs. Equivalence between two circuits implies identical values of corresponding outputs in all reachable states. In this technique, the size of the underlying data structure (some form of decision diagram) does not depend on the number of states or the state elements in the circuit. These techniques have been able to deal with up to a few hundred latches. However, the capability of the state-of-the-art symbolic methods falls below the smallest size designs being optimized in industry.

In [15], a technique is described where sequential optimization is performed on a modified circuit (where each pair of states can be distinguished by applying just one primary input). The modified circuit is obtained by making some latches observable which in turn restricts the amount of optimization that can be performed. The theoretical complexity of their verification problem remains PSPACE-complete (the complexity of an arbitrary sequential equivalence check). However, on a practical note, their technique requires state space traversal of individual machines as opposed to the product machine. They produced results on relatively small MCNC and ISCAS benchmarks because it was not possible to perform single machine state space traversal for large ones.

In [16], a combination of BDD-based and ATPG-based approaches is presented. This technique relies on finding equivalent points in the two circuits and the symbolic justification requires the computation of the transition relation for the product machine. They gave results on circuits optimized by "script.rugged" inside SIS. This optimization is mostly combinational and only redundancy identification and removal leads to minor sequential changes.

In [17], a structural technique for sequential verification is presented. The equivalence is performed by expanding the circuit into an iterative array and by proving equivalence of each time frame by well-known combinational verification techniques. Their technique relies on finding the logic transformations at each time frame. They show results on ISCAS benchmarks, where an optimized circuit is obtained by just one step of combinational optimization (using fx in SIS) followed by retiming. Application of their approach to optimized circuits obtained by a sequence of retiming and resynthesis operations seems difficult.

These solutions can be broadly divided into two categories. The solutions in the first category attempt to solve the general sequential equivalence problem [13, 14, 16, 17]. However, due to complexity of the problem, the proposed solutions are either limited to relatively small size circuits or to circuits which have undergone relatively fewer optimization transformations.

The second approach is to trade the optimization capability with the verification complexity. In this approach, the sequential optimization is constrained in order to reduce the verification complexity. In particular, by making all the latches observable, the sequential synthesis reduces to combinational optimization leading to combinational verification problem. Solution proposed in [15] falls in this category. Our methodology can also be viewed as offering another point in the tradeoff curve between constraints-on-synthesis versus complexity of verification [15].

We propose a technique which reduces the sequential equivalence problem to an instance of combinational equivalence; hence it can be applied in practical verification environments. For each latch, we impose certain constraints on the feedback path (if one exists). If the constraints are not met in the original circuit, we expose a minimum number of latches in order to satisfy the constraints. We allow arbitrary sequences of retiming and synthesis operations for logic optimization. Also, unlike structural based approaches [16, 17], our technique does not rely on the structural similarity between the circuits, we can deal with circuits which have gone through a sequence of retiming and synthesis optimizations. The techniques proposed apply to circuits containing edge-triggered latches (both regular and load-enabled).

In industrial design environment, combinational verification is applied to sequential circuits. However, this requires that little or no retiming is performed. These constraints limit the scope of retiming and synthesis transformations drastically. By contrast, our approach allows an arbitrary number of retiming and combinational synthesis transformations since it does not rely on structural similarity or matching state-bits.

3 Preliminaries

3.1 Circuits and Finite State Machines

A sequential circuit is an interconnection of combinational gates (no combinational cycles) and memory elements along with input and output ports. Typically various notions of sequential circuits differ in the definition of memory elements. We focus on sequential circuits where all the memory elements are edge-triggered latches driven by the same clock (single phase). However, these latches can have load-enable signals. sequential circuit is given as $C = (I, O, G, L)$, where I, O, G, and L are sets of inputs, outputs, gates, and latches respectively. Each latch $l \in L$ is a pair $l = (x, e)$, where x is the output signal of the latch and e is the load-enable signal. For a latch without any load-enable signal (also referred to as regular latch in this paper), we assume $e = 1$. Similar to the notion in [9], we define a latch class $cl = (e)$, which is all latches that have the the same load-enable signal e. This classification is important during retiming transformations, since latches can merge as the result of a move only if they belong to the same class.

3.2 Notion of Equivalence

Several notions of sequential equivalence are proposed in the literature. For circuits with a unique initial state, the "reset" equivalence is checked for all the states reachable from the respective initial states. For multiple initial states, the following notion of equivalence is used: two circuits C_1 and C_2 with initial states set S_{I_1} and S_{I_2} respectively are equivalent *if and only if* for each state $s \in S_{I_1}$, there exists a state $t \in S_{I_2}$ such that C_1 and C_2 are reset equivalent for these initial states and vice versa. For circuits with unknown initial state, several notions of equivalence have been proposed: post-synchronization equivalence [18], safe-replaceability [19], circuit-covering [20], and 3-valued equivalence to name a few.

Similar to 3-valued equivalence, we do not assume a power-up initial state for the latches. Instead, we assume that at power-up each latch has a non-deterministic Boolean value. Note that, this does not prevent the design from having a reset state for some latches which is activated when the reset line is pulled or a reset sequence is applied.

Since the power-up state is non-deterministic, the circuit behavior may not be deterministic for some input sequence. Given a circuit C with L latches, and an input sequence π, the output function $O_C(\pi) = o$ if the circuit produces output o on input sequence π from every power-up state (in $2^{|L|}$); if the circuit produces two different outputs o_1 and o_2 on input sequence π from two different power-up states, we say that $O_C(\pi) = \perp$, where $\perp$ denotes an undefined value.

Definition 1 (Exact 3-Valued Equivalence) *Two circuits C_1 and C_2 are exact 3-valued equivalent if and only if for any input sequence π: $O_{C_1}(\pi) = O_{C_2}(\pi)$.*

Notice that $\perp$ is somewhat similar to the value 'X' used in conservative 3-values logic simulation. However, $\perp$ gets rid of the conservative effects of 3-valued simulation: a 3-valued simulator may incorrectly say that a signal is 'X' because it does not have the ability to correlate the various instances of X values as illustrated in Figure 1. Circuits 1(a) and 1(b) are not 3-valued equivalent, but are equivalent by our definition.

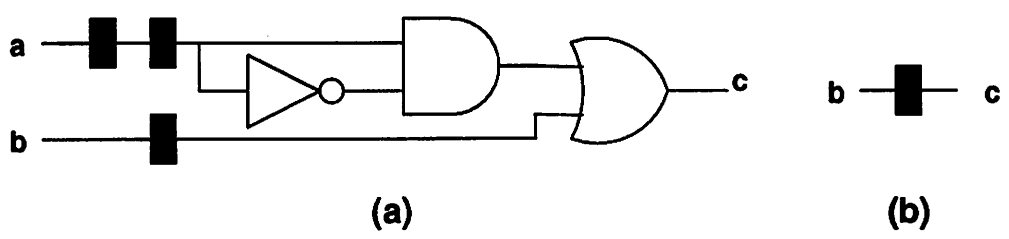


Figure 1: Example of circuits which are not 3-valued equivalent but are equivalent in our notion.

In the next section we present our technique to derive a combinational representation of the sequential circuits. In Sections 5 and 6 we apply it to sequential circuits without and with feedback respectively.

4 From Sequential to Combinational

We reduce the problem of sequential verification to an extension of combinational verification. The goal of our technique is to obtain a canonical acyclic combinational circuit from a given sequential circuit. Towards that we give the following extensions to regular Boolean functions.

$$\begin{aligned} t &: t \in \mathbb{Z} \quad \text{Represents current time} \\ \mathbb{T} &: \{\tau \in \mathbb{Z} : \tau \leq t\} \end{aligned}$$

4.1 Clocked Boolean Function

Definition 2 (Clocked Boolean Function) *A clocked Boolean function (CBF) is defined for circuits containing combinational gates and regular latches. Given a circuit C, the CBF for the circuit represents the functionality of the outputs. This functionality is given in terms of input values in multiple (but finite) clock cycles. Formally, a CBF for the output of a circuit, with n inputs and latch depth d is a Boolean function $F : \mathbb{B}^{n*d} \mapsto \mathbb{B}$. For a signal s in the circuit, the CBF of the signal $s(t)$ at time t is defined inductively as follows:*

- *If s is the output of a gate G, the corresponding CBF is the functional composition of the CBFs of its fanins at the same time instant, i.e., $s(t) = f_g(y_1(t), y_2(t), \cdots, y_n(t))$, where $y_1, y_2, \cdots, y_n$ are the fanin signals of G, and f_g represents its functionality.*
- *If s is the output of a latch, then the CBF is the value of its fanin after one clock cycle, i.e., $s(t) = y(t-1)$, where y is the input of the latch.*
- *If s is the primary input of the circuit, its CBF is an independent input variable $s(t)$. Note that $s(t)$ and $s(t')$ for $t \neq t'$ are different independent variables.*

We illustrate this concept using the following examples:

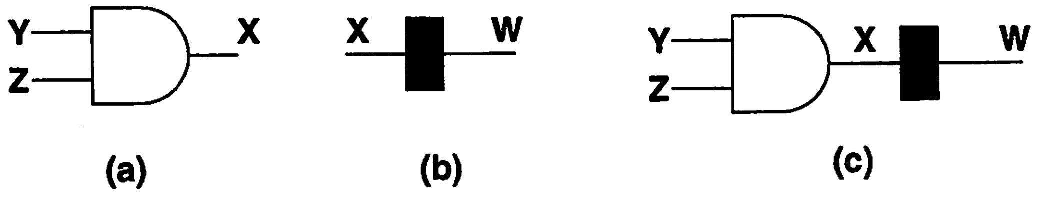


Figure 2: Functionality of AND gate and a latch.

The function f_x for the output of the AND gate is nothing but the logical AND of the functions at the input, i.e., $x(t) = y(t)z(t)$. The function for the latch is interpreted as the function of the latch input signal at the previous clock cycle, i.e., $w(t) = x(t-1)$. If we put the latch and the AND gate together as shown in Figure 2(c), the functionality of the latch output in terms of the primary inputs is given by,

$$w(t) = x(t-1) = y(t-1)z(t-1)$$

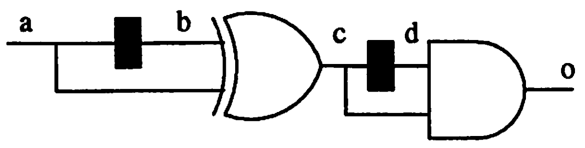


Figure 3: Example of a sequential circuit: latch trapped within a combinational block.

Consider the circuit given in Figure 3. The output function is given as:

$$\begin{aligned} o(t) &= c(t)d(t) \\ d(t) &= c(t-1) \\ c(t) &= b(t)a(t) \\ b(t) &= a(t-1) \\ o(t) &= [a(t-1) \oplus a(t)] \wedge [a(t-2) \oplus a(t-1)] \end{aligned}$$

Essentially, the output function depends on the value of input a in three different clock cycles and we have obtained the CBF for the output.

Unlike the regular Boolean functions which give the value of a signal based on the assignment of input values for one time instant only, the CBF gives the value of a signal for input values delayed by finite number of clock cycles. This notion is very similar to the notion of *Timed Boolean Function* given in [21]. In [21], similar expressions are obtained for the signals which integrate both timing and logical functionality and generalize the conventional Boolean functions to the temporal domain. These expressions were used in timing analysis, analysis and optimization of wave-pipelined circuits, and performance validation of circuits and systems.

4.2 Event Driven Boolean Function

First we define some notation.

$$\begin{array}{rcll} p_i(\tau) & : & \mathbb{T} \mapsto \mathbb{B} & \text{Boolean predicates over time} \\ P & = & \{p_i(\tau)\} & \text{Set of Boolean predicates} \\ \mathbb{E} & = & \bigcup_{k \geq 0}\{E : E \in P^k\} & \text{Set of events} \end{array}$$

where elements of P^k are denoted by $[p_1, p_2, \ldots, p_k]$ and an event $E \in \mathbb{E}$ is an ordered set of timed Boolean predicates.

Next we establish the time instant defined by an event. We define the function $\eta : \mathbb{E} \mapsto \mathbb{T}$ as follows:

$$\begin{aligned} \eta([\,]) &= t \quad \text{empty event denotes the current time} \\ \eta([p_1, p_2, \ldots, p_n]) &= \begin{cases} -\infty \text{ if } A([p_1, p_2, \ldots, p_n]) = \phi \\ \max_\tau\{\tau \in A([p_1, p_2, \ldots, p_n])\} \text{ otherwise} \end{cases} \\ &\text{where} \\ A([p_1, p_2, \ldots, p_n]) &= \{\tau < \eta([p_2, p_3, \ldots, p_n]) : p_1(\tau)\} \end{aligned}$$

Intuitively, for an event $E \in \mathbb{E}$, consisting of Boolean predicates over time, $\eta(E)$ gives the most recent time instant after which all the Boolean predicates in E have been active in the order in which they are listed. If the Boolean predicates in an event cannot be active in the order they are listed, $\eta(E) = -\infty$, indicating an undefined value.

Using the η notation, we now define the next extension to regular Boolean function.

Definition 3 (Event Driven Boolean Function) *An event driven Boolean function (EDBF) is defined for circuits containing combinational gates and enabled latches. The EDBF for the output of a circuit C, with n inputs and k distinct events, is a Boolean function $f : \mathbb{B}^{n*k} \mapsto \mathbb{B}$. For a signal s in C, and an event E, the functionality of s at time $\eta(E)$ is defined inductively as follows:*

- *If s is the output of a gate G, the corresponding EDBF is the functional composition of the EDBFs of its fanins values associated with the same event, i.e., $s(\eta(E)) = f_g(y_1(\eta(E)), y_2(\eta(E)), \dots, y_n(\eta(E)))$, where $y_1, y_2, \dots, y_n$ are the fanin signals of G, and f_g represents its functionality.*
- *If s is the output of a latch with the fanin signal y and the enable signal e, then it takes the most recent value of y at which e was active. This is given as $s(\eta(E)) = y(\eta([e, E]))$.*
- *If s is the primary input of the circuit, it represents an independent input variable.*

Intuitively, for a signal s and the associated event $E \in \mathbb{E}$, the EDBF $s(\eta(E))$ gives the value of s at the most recent time instant after which all the Boolean predicates in E were active in the time order consistent with the listed order.

The following examples illustrate the concept. In Figure 4, the value of signal y, can be represented as

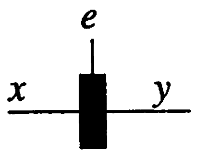


Figure 4: Combinational functionality in the presence of enabled latches (illustration I).

$y(\eta([e]))$, since the value of y is equal to the value of x at the time at which e was last active. In Figure 5, the functionality of signal z associated with an event E can be obtained as follows:

$$\begin{aligned}
z(\eta(E)) &= y(\eta(E)) \cdot x(\eta(E)) \\
y(\eta(E)) &= w(\eta([e_2, E])) \\
w(\eta([e_2, E])) &= u(\eta([e_1, e_2, E])) \\
x(\eta(E)) &= v(\eta([e_3, E])) \\
z(\eta(E)) &= u(\eta([e_1, e_2, E])) \cdot v(\eta([e_3, E])) \qquad (1) \\
& \qquad (2)
\end{aligned}$$

Eqn. 1 indicates that value of z is equal to AND of value of u which has been propagated through both latches L_1 and L_2 and of v which has got propagated through L_3.

In the next section we show how we make use of CBF and EDBF to obtain combinational function of sequential circuits.

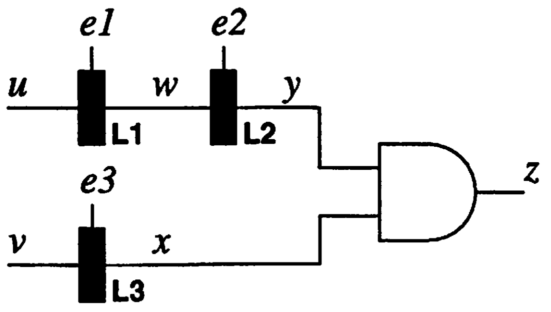


Figure 5: Combinational functionality in the presence of enabled latches (illustration II).

5 Sequential Circuits without Feedback

We consider sequential circuits without feedback paths (also known as "acyclic sequential circuits"). The typical circuits in this category include: pipelined circuits (Figure 6); and acyclic circuits with latches

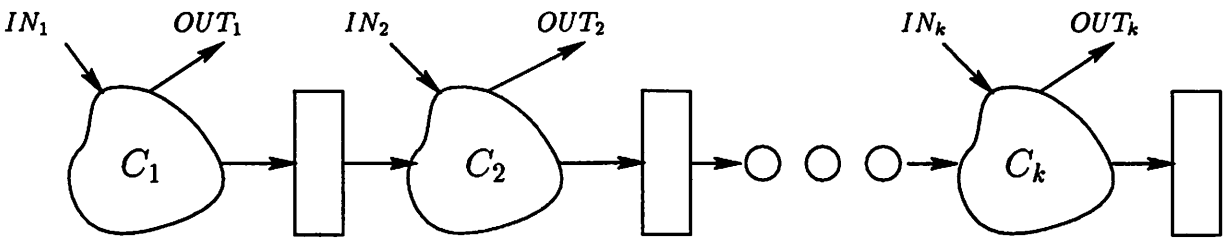


Figure 6: An example of acyclic sequential circuit: pipelined circuit.

trapped within a combinational block (Figure 3). We first explain our technique for circuits with regular latches (no load-enable signal) and then describe the case with load-enabled latches.

5.1 Circuits with Regular Latches

In this class of circuits the latches update their contents at each clock cycle. The functionality of the circuit depends on the input values possibly at multiple time instants.

We give the method to obtain the CBF for a general circuit. Given an acyclic sequential circuit C, in general, the value of a signal can be required for multiple time instants corresponding to different delays (depending on the number of latches along different paths between the signal and the primary outputs). Starting from primary outputs, we recursively obtain the CBF for each signal as shown in the Figure 7. The result of the CBF computation routine is a Boolean formula for each of the outputs in terms of values of inputs in multiple cycles. By treating the input values at different time instants as independent variables, we obtain a combinational function representation for the outputs of the circuit.

```
Compute_CBF($C$){
        foreach primary output $x$ {
            Compute_CBF_Recursively ($x, 0$);
        }
}
Compute_CBF_Recursively($x$,$d$){
        if $x$ is a primary input, return $x(t-d)$;
        if $f(x,d)$ is already computed, return $f(x,d)$;
        if $x$ is output of a latch {
            $y$ = corresponding latch input;
            $f(x,d)$ = Compute_CBF_Recursively($y, d+1$);
        }
        else{
            $G_x$ = Gate corresponding to signal $x$;
            foreach fanin $y$ of $G_x$ {
                Compute_CBF_Recursively($y, d$);
            }
            $f(x,d)$ = Compose the fan-in functions appropriately;
        }
        Cache the result of $f(x,d)$;
        return $f(x,d)$;
}
```

Figure 7: Computing CBF for outputs of a feedback free circuit.

Definition 4 (Sequential Depth) *For an acyclic sequential circuit C, the sequential depth is equal to the largest delay for which an input affects the output. Note that d can be lower than the topological latch depth (maximum number of latches along a path between an input-output pair) due to false dependencies.*

Lemma 5.1 *Given an acyclic circuit C with sequential depth d, suppose $\hat{C}$ is sequentially equivalent to C. Then the sequential depth of $\hat{C}$ is d.*

Proof: Suppose the depth of $\hat{C}$ is $\hat{d} > d$. Then there are sequences I_1 and I_2 of length $\hat{d}$ and identical in the last $\hat{d} - 1$ vectors such that some output of $\hat{C}$ differs on I_1 and I_2 after applying the last vector. However, the output of C will be the same. Hence C is not equivalent to $\hat{C}$, which leads to contradiction. The case when $d > \hat{d}$ is similar. ∎

5.1.1 Canonicity of the Formula

Theorem 5.1 *Suppose C_1 and C_2 are two circuits and F_1 and F_2 their CBFs. Then $F_1 \equiv F_2 \Leftrightarrow C_1 \equiv C_2$, where equivalence between the circuits is exact 3-valued as defined in Section 3.2, and equivalence between the CBFs is combinational.*

Proof: (sketch)
$\Leftarrow$
Assume that $F_1 \not\equiv F_2$. Then there exists a CBF minterm m on the input values up to d clock cycles such that $F_1(m) \neq F_2(m)$. Since the circuit has finite depth, using this minterm m we can generate an input sequence of length d such that when applied to the two circuits, will produce different simulation results. This implies $C_1 \neq C_2$.
$\Rightarrow$
Assume that $C_1 \not\equiv C_2$. Then there exists an input sequence π such that $C_1(\pi) \neq C_2(\pi)$. Since the circuits are acyclic and have finite memory, π need not be longer than d. Using this sequence we can generate a CBF minterm such that when applied to the two formulae, will produce different results implying $F_1 \neq F_2$. Hence contradiction. ∎

Note that the above result are stated for any two sequential equivalent circuits not just those obtained by retiming and combinational optimization.

5.2 Circuits with Load-enabled Latches

In the case where the latch output is controlled by an enable signal as well, the functionality is as follows: if the enable signal is high, the latch propagates the data value to the output, else the latch retains its old value. In [9], a retiming technique was proposed to handle latches with different enable signals and different clocks. In this work, we propose a verification methodology where all the latches are driven by the same clock but can have different enable signals. Extension to circuits with multiple clocks is straightforward.

We obtain a Boolean function along the lines of the previous case (regular latches). However, in this case we make use of event driven Boolean functions (EDBF) as defined in Section 4.2. By instantiating separate Boolean variables for each unique combination of primary input and event, we create a combinational representation of the circuit.

Starting from primary outputs, we recursively obtain the EDBF for each signal as shown in the Figure 8.

```
Compute_EDBF($C$){
    foreach primary output $x$
        Compute_EDBF_Recursively ($x$, [])
}
Compute_EDBF_Recursively($x, E$){
    if $x$ is a primary input, return $(x, E)$.
    if $F(x, E)$ is already computed, return $F(x, E)$.
    if $x$ is a latch output {
        $y$ = latch input
        $e$ = enable signal
        $F(x, E)$ = Compute_EDBF_Recursively($y$, $[e, E]$).
    }
    else{
        $G_x$ = Gate corresponding to signal $x$.
        foreach fanin $y$ of $G_x$ {
                Compute_EDBF_Recursively($y, E$)
        }
        $F(x, E)$ = Compose the fan-in functions appropriately.
    }
    return $F(x, E)$.
}
```

Figure 8: Computing EDBF for the outputs of a circuit

5.2.1 Canonicity of the Formula

Lemma 5.2 *Given an acyclic sequential circuit with load-enabled latches, an input/output pair a path between the pair, the number of latches and the event associated with the sequence of enabling signlas of the latches along the path is invariant during retiming (ala [9]) and synthesis optimization steps.*

Proof: Let us first consider the retiming transformation.

Suppose $\{G_1, G_2, ..., G_k\}$ is a path of gates between an input I and output O. Assume that the latches cannot be retimed across input and output ports. Suppose, during a retiming move, x latches move across gate G_i. If the latches are moved in the forward direction, then x latches are moved from each fanin of G_i (including G_{i-1}) to each fanout of G_i (including G_{i+1}). Hence the number of latches between G_{i-1} and G_{i+1} along the path remains the same. Suppose $e_1, e_2, \ldots, e_k$ is the sequence of enable signals of the latches along a path between input I and output O. For the forward or backward movement of load-enabled latches, the latches being moved must belong to the same enable class. Also, since the circuit is acyclic, a latch cannot cross over to another latch during retiming thereby changing the order of enable signals. It implies that the sequence of the enable signals is preserved.

Since combinational synthesis keeps the latch positions fixed, the latch count and the sequence of enable signals along any path in the circuit does not change. To establish that a path pertaining to the true dependency is preserved during the transformation, we make use of illustration in the Figure 9. Since

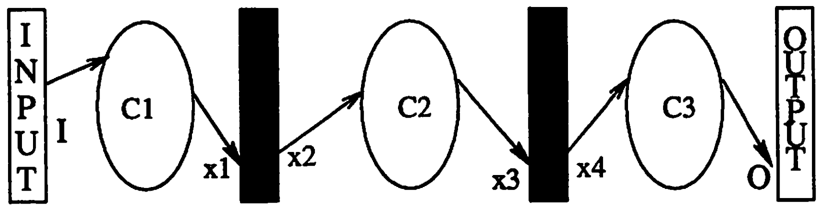


Figure 9: Topological arrangement of latches (black boxes) and combinational blocks (ovals).

the circuit is acyclic, we can arrange the combinational logic and the latches as shown (for simplicity, only two layers of latches are shown in the figure). Now the path $I \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow O$, from input I to output O as shown in the figure. For combinational optimization x_1, x_3 and x_2, x_4 are treated as primary outputs and primary inputs respectively. Hence to preserve the functionality of combinational blocks, paths from I to x_1, from x_2 to x_3, and from x_4 to O, must be preserved. This implies that the number of latches and the sequence of enable signals along the path is also preserved. ■

Theorem 5.2 *Given two acyclic sequential circuits C_1 and C_2 with load-enabled latches, such that C_1 has been obtained from C_2 by retiming and combinational synthesis transformations. Suppose F_1 and F_2 are their EDBFs as computed by the algorithm of Figure 8. Then $F_1 \equiv F_2 \Leftrightarrow C_1 \equiv C_2$.*

Proof: (sketch)

$\Rightarrow$

Assume $C_1 \not\equiv C_2$. Then there exists an input sequence π, such that $C_1(\pi) \neq C_2(\pi)$. Without loss of generality, we assume that for some output k, $C_{1_k}(\pi) \neq \bot$ and $C_{1_k}(\pi) \neq C_{2_k}(\pi)$. Now, since $C_{1_k}(\pi) \neq \bot$, it implies that for input sequence π, all the enable signals for the k^{th} output must be active in the sequence they appear in the circuit. Since the number of latches and the enable sequence must be same in C_1 and

C_2 (from Lemma 5.2), $C_{2_k}(\pi) \neq \perp$. Hence using π, we can create an EDBF minterm m, such that $F_1(m) \neq F_2(m)$.

$\Leftarrow$

Assume $F_1 \neq F_2$. Since the number of latches and sequence of enable signals is same for C_1 and C_2 (from Lemma 5.2), the support variable set is identical for F_1 and F_2. Consider an EDBF minterm m such that $F_1(m) \neq F_2(m)$. The minterm m can be used to generate a sequence of events and input values such that when applied to the circuits, C_1 and C_2 will result in different outputs. ■

Unlike the regular latch case, the result does not hold for any two sequentially equivalent circuits. This is illustrated by following two examples.

In Figure 10, two sequentially equivalent circuits are presented. However, their EDBFs would be

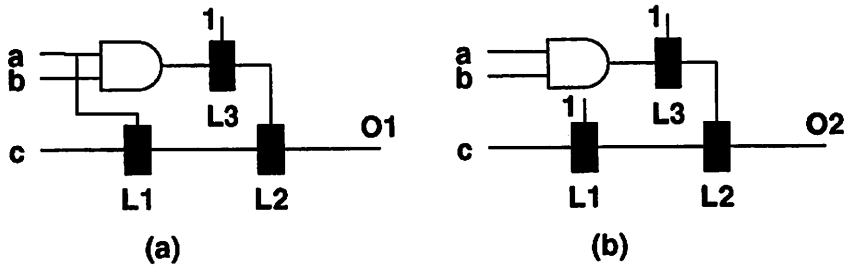


Figure 10: EDBF can lead to false negatives: illustration I.

different since the enable signal of latch L_1 is different in the two circuits. The EDBF for the outputs O_1 and O_2 can be given as following:

$$\begin{aligned} O_1 &= c(\eta[a(\tau), a(\tau-1)b(\tau-1)]) && (3)\\ O_2 &= c(\eta[1, a(\tau-1)b(\tau-1)]) \\ &= c(\eta[a(\tau-1)b(\tau-1)]) && (4) \end{aligned}$$

Our technique will result in a false negative since the events defining the time instant for the value of c are syntactically different even though the definition is the same. We can work around this problem by rewriting our events. For example, it can be proven that,

$$p(\tau) \geq q(\tau) \quad \Rightarrow \quad \eta[p(\tau), q(\tau-1)] = \eta[q(\tau-1)] \qquad (5)$$

Applying (5), on (3) (with $p(\tau) = a(\tau)$ and $q(\tau) = a(\tau)b(\tau)$), we get,

$$\begin{aligned} O_1 &= c(\eta[a(\tau-1)b(\tau-1)]) \quad \text{From (5)} \\ &= O_2 \end{aligned}$$

This rewriting rule extends the applicability of our technique. However, this rule is not complete, as shown by the next example. In Figure 11, (a) and (b) are two sequentially equivalent circuits. In this case, the enable signals to both the latches are the same. However, the data inputs to the latches are different. The EDBF representation for these two circuits are following:

$$\begin{aligned} O_1 &= b(\eta(\bar{a}+b)) \\ O_2 &= a(\eta(\bar{a}+b)) + b(\eta(\bar{a}+b)) \end{aligned}$$

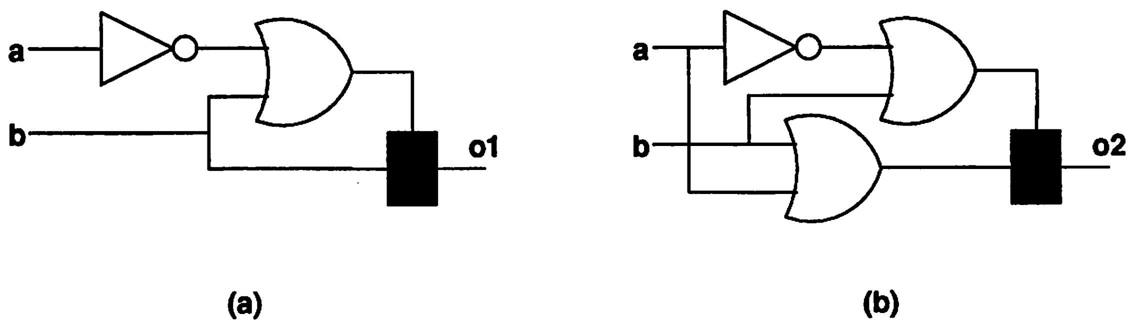


Figure 11: EDBF can lead to false negatives: illustration II.

This results in a false negative. Essentially, in this example there is some interaction between the enable and the data signals of the latch, resulting in equivalent sequential functionality even though the EDBFs are different. To handle these cases, we need to establish equivalence not only between different forms of events, but also between different forms of event/data interaction. Until then, our methodology for circuits with load enabled latches provides conservative check.

6 Sequential Circuits with Feedback

In these circuits there exists a feedback path for some latches. Our strategy is to model a latch with feedback in the form of an enabled latch with appropriate enable and data signals as shown in Figure 12. Next we derive the conditions under which this modeling is feasible.

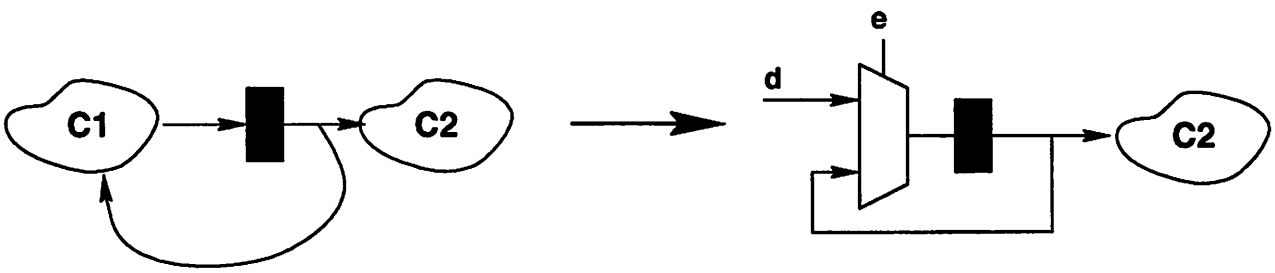


Figure 12: Modeling feedback path for a latch with enable and data signals

Lemma 6.1 (Decomposition Condition) *Suppose the next-state function of a latch x given as $F(x)$. Then $F(x) = e \cdot d + \bar{e}x \Leftrightarrow F_{\bar{x}} \subseteq F_x$, i.e., $F(x)$ can be decomposed in the form of Figure 12 if and only if $F(x)$ is positive unate in x.*

Proof:
$\Rightarrow$

$F_x = ed + \bar{e} \supseteq ed = F_{\bar{x}}$.
$\Leftarrow$
Let $e = \bar{F}_x + F_{\bar{x}}$ and $d = F_{\bar{x}}$. Then,

$$\begin{aligned} ed + \bar{e}x &= (\bar{F}_x + F_{\bar{x}})F_{\bar{x}} + xF_x\bar{F}_{\bar{x}} \\ &= xF_x + \bar{x}F_{\bar{x}} + xF_{\bar{x}} \\ &= F \quad \text{(for a positive unate function)} \end{aligned}$$

■

As a matter of fact, any d, which satisfies,

$$\begin{aligned} F_{\bar{x}} &\leq d \leq F_x \\ \text{i.e., } B &\leq d \leq A + B \end{aligned} \qquad (6)$$

can be used as the data signal. The value of e, on the other hand, is unique (the derivation is straightforward).

Thus for latches whose next state function is positive unate in the latch variable, the feedback can be modeled via a multiplexer. The advantage of the model shown in Figure 12, is that a latch fed by a multiplexer can be thought of as an enabled latch as shown in the Figure 13. This gets rid of the

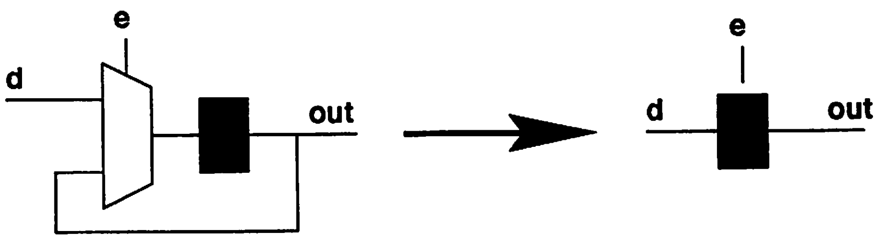


Figure 13: Modeling an enabled latch with extra logic

feedback path and for our purposes the circuit becomes acyclic. Now we can apply the analysis techniques developed in Section 5.2 for acyclic circuit with enabled latches. However, we need to be aware of following issues:

1. The data-input and the enable signal both need to be independent of the latch signal, else it will create a cycle.

2. The data value d obtained from the function $F = ed + \bar{e}x$ is not unique as shown in (6) since d has a don't care as e. Hence for two circuits C_1 and C_2 we can come up with different decompositions leading to false negatives. This is the basis behind the counterexample in Figure 11, where the decomposition of the next state function $ax + b$ is different for the two circuits. This can be handled in following ways:

   (a) By fixing the latch modeling in the circuit, i.e., once we model the feedback path of a latch by an enabled latch, we restrict the logic optimization of the feedback logic by not using e as don't care and also, we move the latch in tandem with the logic for the enable signal. This will guarantee the event correspondence in two circuits. However, by preserving the multiplexor logic we incur some optimization penalty.

(b) By using the lower limit of the possible data signal, i.e., $d = F_{\bar{x}}$. This guarantees the matching of the enable signals, but an optimization penalty may be incurred.

(c) Perform canonical decomposition of the enable and data signals. Below we give a sufficient condition for such decomposition.

Lemma 6.2 *Given a function $F = Ax + B$, suppose (e, d) and $(e, \hat{d})$ are two decompositions such that e and d have disjoint Boolean supports. Then $d = \hat{d}$, i.e., there is a unique decomposition of F such that d and e have different supports (if such decomposition exists).*

Proof: We have,

$$\begin{aligned} ed + e'x &= e\hat{d} + e'x \\ \Rightarrow ed\bar{x} &= e\hat{d}\bar{x} \\ \Rightarrow ed &= e\hat{d} \end{aligned} \tag{7}$$

Eqn. 7 follows from the fact that ed is independent of x. Since e and d have different supports (and so have e and $\hat{d}$), from (7) d and $\hat{d}$ must have the same support. Suppose X and Y are the support sets for e and $d, \hat{d}$ respectively.
Assume $d \neq \hat{d}$. Then there exists a minterm y on the Y variables such that $d(y) \neq \hat{d}(y)$. Choose an arbitrary minterm x on variables of X such that $e(x) = 1$. Suppose $(x \cup y)$ is the minterm on X and Y variables.

$$\begin{aligned} d(y) &= d(x \cup y) \\ \hat{d}(y) &= \hat{d}(x \cup y) \\ e(x) &= e(x \cup y) = 1 \end{aligned} \tag{8}$$

Since $d(y) \neq \hat{d}(y)$, thus $e(x \cup y)d(x \cup y) \neq e(x \cup y)\hat{d}(x \cup y)$. This contradicts (7). ■

The feedback modeling as derived in Figures 12 and 13 is best suited for the class of circuits where latches update their values when a set of conditions is met, else they keep their previous values. This is illustrated in Figure 14. The latches with feedback paths, for which we cannot derive the enabled latch

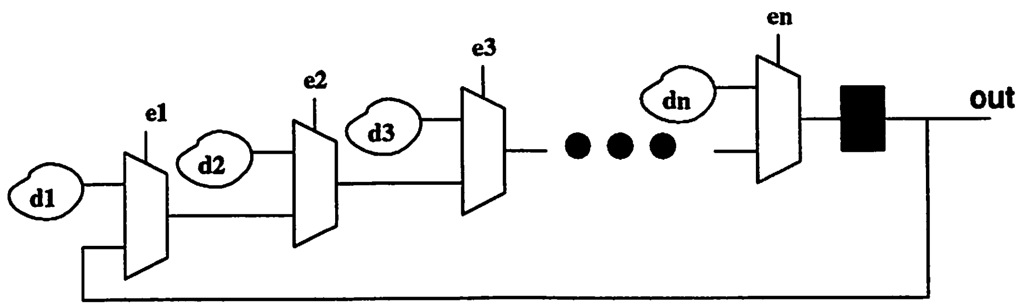


Figure 14: Conditional updating of the latch content.

model, are handled in the following way. We find minimum number of latches that need to be exposed,

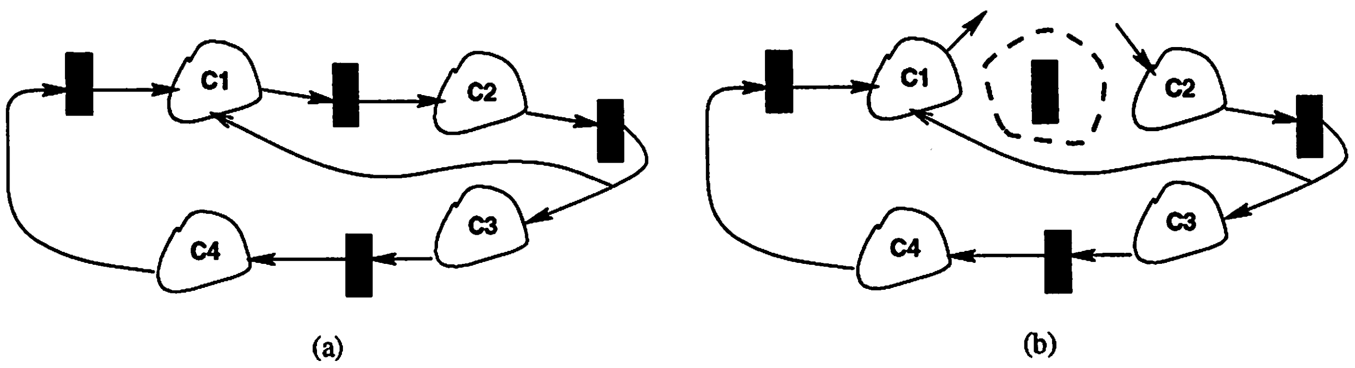


Figure 15: Making some latches observable to meet the feedback criterion.

i.e., need to be made observable, in order to remove the feedback path for these latches. This is illustrated in Figure 15. By exposing latches, we treat their outputs as primary inputs and hence the feedback paths are broken.

After finding the minimal set of latches to be exposed, we impose constraints on the synthesis step such that these latches cannot be moved during retiming.

7 Experimental Setup

7.1 Circuit Modification

Given a sequential circuit C, we create a directed graph $G = (V, \vec{E})$ in the following manner. For each combinational gate, latch, primary input and primary outputs we create a node. An edge from node v_i to v_j is created if there is a fanout from gate/latch/primary-input i to gate/latch/primary-output j. The graph in general has cycles due to feedback paths to latches. Once we identify the latches with feedback paths that do not satisfy the criterion mentioned in the previous section, we obtain a minimal set of latches to expose. This is the "minimum feedback vertex set problem" which is NP-complete. We used a modified version of the heuristics given in [22].

7.2 Retiming

Retiming was done using *Minaret* [6]. This tool only supports the constant delay model (we could not find any efficient public domain retiming tools, which supported better delay models). Retiming was performed in two modes. First, the minimum feasible period was obtained and the area of the circuit was optimized for this period. In the second mode, the delay obtained through combinational optimization was used as the timing constraint and then constrained minimum area retiming was performed.

We could not find a public domain retiming tool which could handle latches with enable signals as proposed in [9] and shown in Figure 16.

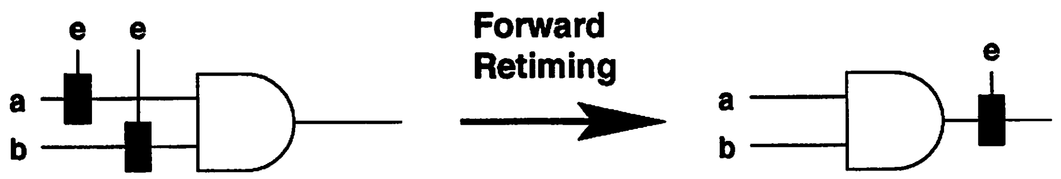


Figure 16: Retiming enabled latch across gates

7.3 Combinational Optimization

We perform combinational optimization to obtain a minimum delay circuit. SIS [2] was used for synthesis purposes. A modified version of "script.delay" was used as shown in Figure 17. The modifications were

```
sweep
decomp -q
tech_decomp -o 2
resub -a -d
sweep
reduce_depth -b -r
eliminate -l 100 -1
simplify -l
sweep
decomp -q
fx -l
tech_decomp -o 2
rlib mylib2.genlib
rlib -a lib2_latch.genlib
map -s -n 1 -AFG -p -B -b 1000
print_delay -p1 -a -m unit
```

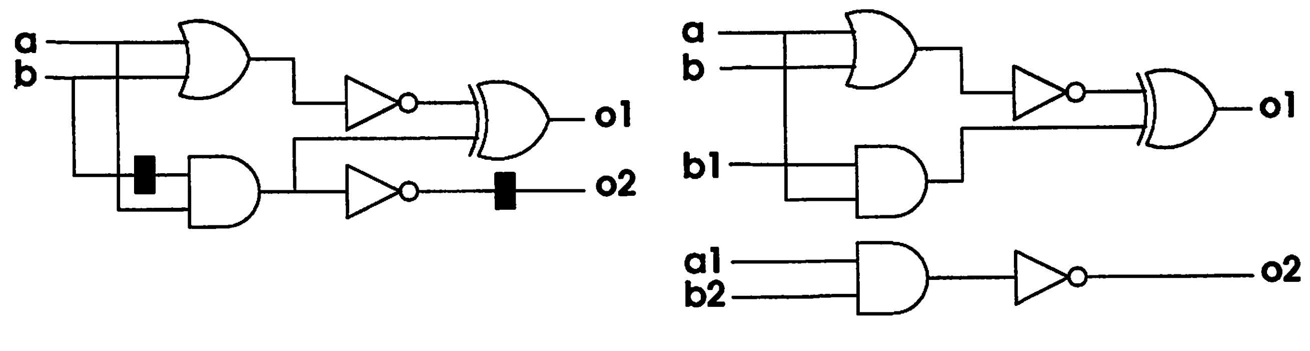
Figure 17: Script for synthesizing minimum delay circuit

made because the original script was not able to handle large designs (or took very long to complete).

As mentioned earlier, the unit delay model was used during retiming. Hence for consistency we used the unit delay model during synthesis steps as well. To keep the size of gates small, we created a library consisting of inverter, 2-input nand and 2-input nor gates only. Also, for reasonable optimization results we limited the number of fanouts for each gate to four. The delay models and the fanout limitation changes were achieved by appropriately modifying the library.

7.4 Generating Equivalent Combinational Equivalence Problems

In order to leverage from the existing combinational equivalence tools, we mapped the equivalence problem of CBF/EDBFs into combinational equivalence problems. This was done by creating a combinational circuit with appropriate variables which represents the CBF or EDBF. An illustration is shown in Fig-



(a) (b)

Figure 18: Generating equivalent combinational equivalence problems

ure 18. The combinational circuit Figure 18(b) represents the CBF for the sequential circuit Figure 18(a). Essentially, if the circuit outputs depend on the value of a signal at k different time instants (for a circuit with regular latches) or with k different enable signal paths (for a circuit with enabled latches), the cone of logic for the signal is replicated k times. The size of these circuits could become large due to replications. Note, however, that this step is performed only for convenience (to treat the combinational equivalence checker as a black box). In practice, a modified combinational equivalence checker could be used which would not require generation of such circuits and hence no blow-up would occur.

The combinational verification was performed by an in-house tool similar to the ones presented in [10, 12].

8 Experimental Results

Our experiment consisted of following steps (see also Figure 19).

1. Given the sequential circuit (A), modify it appropriately to satisfy constraint on all feedback paths to obtain a new circuit (B). This is done by creating a circuit graph and finding a minimal feedback vertex set. Due to lack of a retiming tool which could handle load-enabled latches, we did not model any latches with feedback path as load-enabled latches (as shown in Figure 12). In general, this leads to fewer latches that need to be exposed.

2. Perform synthesis for delay optimization and min-period retiming on the modified circuit (B) to obtain a new circuit (C).

3. To illustrate the advantage of combining retiming with combinational synthesis, we also performed pure combinational optimization (using the same script) on original circuit (A) to obtain circuit (D).

4. We also compared the saving in area by performing constrained minimum area retiming. This was done on circuit (B) with the delay value of circuit (D) to obtain a new circuit (E).

5. To measure the loss in optimization due to modification in step 1, retiming and synthesis optimization on original circuit (A) was performed to obtain an optimized circuit (F).

6. Step 5 was repeated to measure the loss of optimization in circuit (E). This was done by performing constrained minimum area retiming on (A) with the delay value of circuit (D) to obtain a new circuit (G).

7. Combinational circuits (H and J) were created (as described in Section 7.4) to obtain circuits (B) and (C) respectively.

8. Perform combinational verification between (H) and (J). Verifying equivalence of circuits (B) and (E) would be similar and is not done in the experiment.

The active area and delay numbers are obtained by the "map" command. The verification was performed on an UltraSparc-1 with 256MB of memory.

In Table 1, we have given results comparing the optimization potential of our strategy and also the corresponding verification times.

All the industrial circuits we investigated contained load-enabled latches. Since we did not have access to a retiming tool for circuits with load-enabled latches, we could not perform retiming on these circuits and hence could not get optimization and verification results. However, we did extensive analysis on them to understand the nature of feedback paths to latches. After analyzing a set of circuits we observed that most of the feedback path exists due to interaction with memory and communication layer as shown in Figure 20. Typically, designers want to keep the boundary between the design and communication layer/memory preserved and they do not synthesize them together. We can take advantage of this fact and can assume for our purposes that these feedback paths do not exist. In Table 2, we have given the number of latches exposed in order to satisfy the feedback path constraint. Currently we do only structural analysis which can detect the kind of circuits as shown in Figure 14. A more detailed functional analysis (based on the next state function of the latches as explained in Section 6) would lead to reduced number of exposed latches.

8.1 Analysis

By analyzing the data given in Tables 1 and 2 we make following observations:

1. Comparing δ values in columns C and D, for most of the circuits the delay values obtained through our approach is better than that by purely combinational optimization. In some cases delay values reduces by as much as 50%. The area penalty incurred in the process is negligible.

2. Comparing area numbers in columns D and E, for the same delay, retiming allows us to reduce the area.

3. The verification times were quite reasonable. Most of the examples took less than a minute to verify. The maximum time taken is fifteen minutes. Note that, for only few of these sequential circuits the state-space can be traversed, and for fewer yet the state-space of the product machine can be traversed. This makes the proposed technique quite attractive.

4. Comparing the area numbers in columns D and E, we observe that the penalty paid in terms of reduced optimization capability was not significant in most of the cases.

5. Looking at the data for industrial circuits from Table 2, we observe that even though these circuits are highly control intensive implying a relatively tight interaction among latches, we did not need to expose more than 50% latches and sometimes as few as 2% latches were exposed. As mentioned, these numbers will decrease when positive unateness is used.

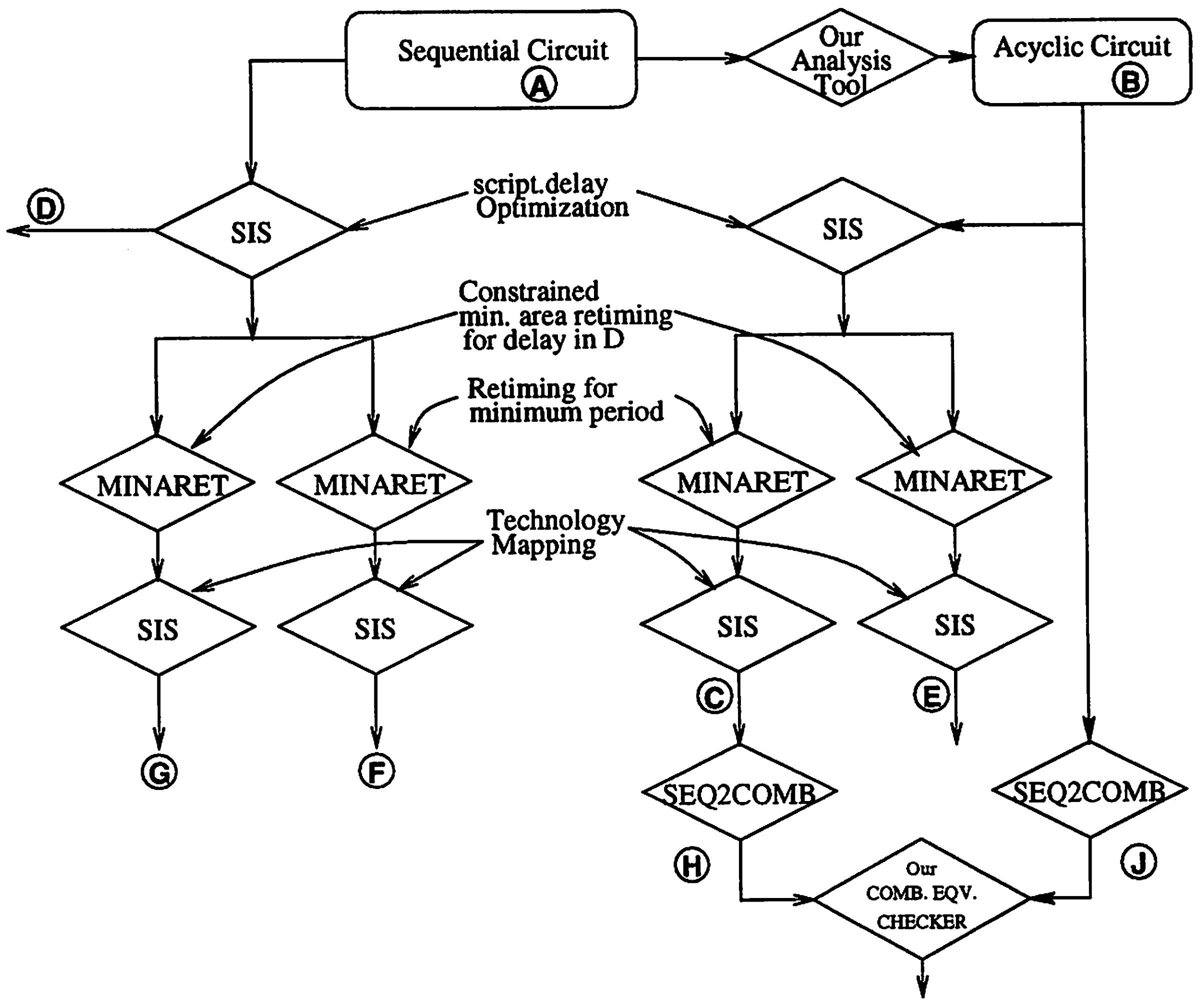


Figure 19: Flow chart indicating experimental set up.

Circuit	A	F			C				D		G		E		Time
	# L	# L	Area	δ	%	# L	Area	δ	Area	δ	# L	Area	# L	Area	H vs J
minmax10	30	30	0.87	50	66	30	0.74	50	1.00	56	30	0.87	30	0.74	2
minmax12	36	36	0.87	54	66	36	0.75	54	1.00	57	36	0.87	36	0.75	2
minmax20	60	60	0.83	55	66	60	0.80	64	1.00	94	60	0.83	60	0.80	3
minmax32	96	96	0.81	88	66	96	0.72	96	1.00	145	96	0.81	96	0.72	5
prolog	65	85	1.06	15	43	65	0.97	17	1.00	18	65	1.00	65	0.97	7
s1196	18	18	1.00	21	0	18	1.00	21	1.00	21	18	1.00	18	1.00	5
s1238	18	18	1.00	19	0	18	0.99	19	1.00	19	18	1.00	18	0.99	7
s1269	37	69	1.12	24	75	37	0.98	32	1.00	33	37	1.00	37	0.98	6
s1423	74	78	1.01	40	95	74	1.00	44	1.00	45	74	1.00	74	1.00	6
s3271	116	221	1.18	16	94	116	0.99	25	1.00	26	116	1.00	116	0.99	7
s3384	183	174	0.97	30	39	154	0.95	56	1.00	57	154	0.95	154	0.95	34
s400	21	32	1.17	11	71	21	0.98	13	1.00	14	18	0.95	21	0.98	1
s444	21	32	1.17	10	71	21	0.99	13	1.00	14	18	0.95	21	0.99	1
s4863	88	146	1.05	32	18	142	1.04	32	1.00	58	78	0.99	83	0.99	4:25
s641	19	19	1.00	24	78	19	1.00	24	1.00	24	19	1.00	19	1.00	1
s6669	231	272	1.02	32	17	234	1.00	55	1.00	85	193	0.97	214	0.98	1:54
s713	19	19	1.00	25	78	19	0.96	24	1.00	25	19	1.00	19	0.96	1
s9234	135	169	1.05	21	66	144	1.00	27	1.00	30	128	0.98	135	0.98	22
s953	29	22	0.95	16	20	29	1.00	14	1.00	16	22	0.95	29	1.00	3
s967	29	22	0.95	15	20	29	1.00	14	1.00	15	22	0.95	29	1.00	3
s3330	65	78	1.04	15	43	65	0.96	17	1.00	19	65	1.00	65	0.96	7
s15850	515	537	1.01	34	72	515	1.00	46	1.00	46	495	0.99	515	1.00	11:24
s38417	1464	1285	0.96	33	70	1463	1.03	36	1.00	37	1248	0.95	1463	1.03	15:32

Table 1: Results on sequential optimization and verification.

A: Original circuit
B: Modified circuit (not shown in the table)
C: Obtained from B after retiming (for minimum period) and synthesis
D: Obtained from A after Combinational optimization only
E: Obtained from B after retiming (for delay in D) and resynthesis
F: Obtained from A after retiming (for minimum period) and synthesis
G: Obtained from A after retiming (for delay in D) and resynthesis
H: Combinational circuit for the CBF for circuit B
J: Combinational circuit for the CBF for circuit C
L: Latches
%: Percentage of latches exposed in B
δ: Delay of the circuit
Area: Normalized against the area of D.
H vs J: CPU time (in minutes:seconds) for combinational verification between H and J

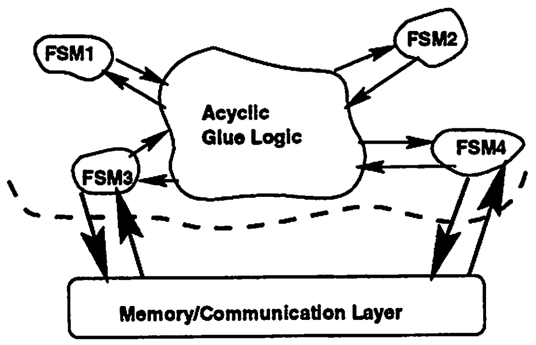


Figure 20: Feedback paths due to interaction with memory and communication layer

Example	# Latches	# Exposed
ex1	2157	934
ex2	100	16
ex3	146	56
ex4	1437	835
ex5	672	305
ex6	412	250
ex7	453	81
ex8	968	470
ex9	783	15
ex10	634	174
ex11	792	369
ex12	2206	691

Table 2: Number of latches exposed for some industrial circuits

9 Conclusions and Future Directions

We proposed a practical verification technique for circuits which have undergone retiming and combinational synthesis transformations. In particular, we show that the corresponding sequential verification can be reduced to an extension of combinational verification. The proposed technique can deal with circuits with and without feedback paths, and with regular and load-enabled latches. We impose a constraint on the feedback path (if one exists) of latches. If these constraints are not met by the original circuit, we fix the position of some of the latches to cut the feedback paths. Experimental results indicate that imposing constraints does not result in significant optimization penalty. Our strategy can be used to obtain faster circuits by allowing the retiming transformations while performing fast verification as indicated by our experimental results.

To make our approach exact for arbitrary sequential optimization, we need to develop a complete technique to distinguish events and combination of events and signals. Also, a better technique could be used to find the latches to be exposed. The strategy of finding the minimum latches may not always be optimum for area/delay optimizations. The need would be to identify latches, such that the least amount of area/delay penalty is paid by exposing them.

10 Acknowledgments

We would like to thank Naresh Maheshwari from Iowa State University for making the retiming tool Minaret available and for providing help with the tool from time to time.

References

[1] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] E. M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[3] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing Synchronous Circuitry by Retiming," in *Advanced Research in VLSI: Proc. of the Third Caltech Conf.*, pp. 86–116, Computer Science Press, 1983.

[4] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," in *Algorithmica*, pp. 5–35, 1991.

[5] N. Shenoy and R. Rudell, "Efficient Implementation of Retiming," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 226–233, Nov. 1994.

[6] N. Maheshwari and S. S. Sapatnekar, "An Improved Algorithm for Minimum-Area Retiming," in *Proc. of the IEEE/ACM Design Automation Conf.*, 1997.

[7] N. Shenoy and A. Brayton, R.K.and Sangiovanni-Vincentelli, "Minimum Padding to Satisfy Short Path Constraints," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 156–61, Nov. 1993.

[8] B. Lockyear and C. Ebeling, "Retiming of Multi-phase, Level-clocked Circuits," in *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pp. 265–280, Mar. 1992.

[9] C. Legl, P. Vanbekbergen, and A. Wang, "Retiming of Edge-Triggered Circuits with Mulitple Clocks and Load enables," in *Proc. IEEE/ACM Intl. Workshop on Logic Synthesis*, 1997.

[10] Y. Matsunaga, "An Efficient Equivalence Checker for Combinational Circuits," in *Proc. of the IEEE/ACM Design Automation Conf.*, pp. 629–34, June 1996.

[11] J. Jain, R. Mukherjee, and M. Fujita, "FLOVER: Filtering Oriented Combinational Verification Approach," in *Proc. IEEE/ACM Intl. Workshop on Logic Synthesis*, pp. 263–68, May 1997.

[12] A. Kuehlmann and F. Krohm, "Equivalence Checking Using Cuts and Heaps," in *Proc. of the IEEE/ACM Design Automation Conf.*, pp. 263–68, June 1997.

[13] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines using BDD's," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 130–133, Nov. 1990.

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

[15] P. Ashar, A. Gupta, and S. Malik, "Using Complete-1-Distinguishability for FSM Equivalence Checking," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 1996.

[16] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "AQUILA: An Equivalence Verifier for Large Sequential Circuits," in *Proc. of Asian and South Pacific Design Automation Conf.*, 1997.

[17] D. Stoffel and W. Kunz, "A structural fixpoint iteration for sequential logic equivalence checking based on retiming," in *Proc. Intl. Conf. on Computer-Aided Design*, 1997.

[18] C. Pixley, "A Theory and Implementation of Sequential Hardware Equivalence," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1469–1494, Dec. 1992.

[19] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton, "Multi-level Synthesis for Safe Replaceability," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 442–49, Nov. 1994.

[20] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "An ATPG-based ramework for Verifying Sequential Equivalence," in *Proc. Intl. Test Conf.*, 1996.

[21] W. Lam, *Algebraic Methods for timing analysis and testing in high performance designs*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, Apr. 1993. Memorandum No. UCB/ERL M94/19.

[22] D. H. Lee and S. M. Reddy, "On Determining Scan Flip-Flops in Partial-Scan Designs," in *Proc. Intl. Conf. on Computer-Aided Design*, pp. 322–25, 1990.