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 sequential verification is not practical and current simulation methodology requires the correspondence of latches disallowing any movement of latches. 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 equivalence checking 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

Many solutions to the sequential equivalence problem have been proposed in the literature which can be broadly divided into two categories. The solutions in the first category attempt to solve the general sequential equivalence problem [17, 3, 6, 16]. However, due to the complexity of the problem, the proposed solutions are either limited to relatively small-sized circuits or to circuits which have undergone relatively fewer optimization transformations.

The second approach is to trade off the optimization capability with the verification complexity. In this approach, sequential optimization is constrained in order to reduce the verification complexity. In the limit, by making all the latches observable, sequential synthesis reduces to combinational optimization leading to combinational verification problems. The solution proposed in [1] falls in this category.

We propose a methodology that reduces a sequential verification problem into an equivalent combinational verification problem for a class of circuits. This allows us to leverage powerful combinational verification techniques. Our method requires the next state function to be positive unate in the latch variable for each latch with a feedback path. If the original circuit does not meet these constraints, we make a minimum number of latches observable to satisfy the constraints. Thus, our methodology effectively offers another tradeoff point between constraints-on-synthesis versus complexity-of-verification. Later we show this methodology allows self-loops on latches, pipelined circuits where the latches cannot be retimed to the periphery, latches trapped inside combinational blocks, with load-enabled latches, and latches that conditionally update their contents.

Related to our technique, Bischoff *et al.* [2] verify the implementation of the bus interface unit for the Alpha 21264 microprocessor. They compare RTL with gates extracted from a custom transistor netlist. However, no formal framework for such verification is presented and no technique to handle circuits with feedback paths was given. Wherever applicable, we make relevant distinction with their work.

Our paper proceeds from definitions in Section 2 to the basic idea in Section 3. In Section 4 we discuss our technique for a circuit with no feedback latches and in Section 5 we extend it to circuits containing feedback latches. The details of the experimental setup and results are given in Sections 6 and 7, respectively.

2 Preliminaries

In this section we present our circuit model and notion of equivalence.

2.1 Circuit Model

A sequential circuit is an acyclic interconnection of combinational gates connected to memory elements along with input and output ports. Various notions of sequential circuits typically differ in the definition of memory elements. Our memory elements are edge-triggered latches driven by the same clock (single phase) with load-enable signals. A sequential circuit is $C = (I, O, G, L)$, 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 latch's output signal and e is its load-enable signal. A latch without a load-enable signal (a "regular latch" in this paper) has $e = 1$. Following [10], we define a latch class $cl = (e)$ to be all latches with 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.

2.2 Notion of Equivalence

We use the notion of **steady-state equivalence**. Similar to 3-valued equivalence, we do not assume a power-up initial state for the latches. Instead, we assume each latch powers up with value X. This does not prevent the design from having a reset state for some latches activated when the reset line is pulled or a reset sequence is applied. Since the latches power-up in the state X, the circuit outputs may not take a Boolean value for a given input sequence.

Definition 1 *For circuit C, an input sequence π is a* **steady-state input sequence** *if the 3-valued simulation of C on π (the final value of all its outputs), denoted $O_C(\pi)$, results in Boolean values for all its outputs. The set of all steady-state input sequences of a circuit C is denoted as Π_C.*

Definition 2 (Steady-state equivalence) *Two circuits C_1 and C_2 are* **steady-state equivalent** *if and only if $\forall \pi, \pi \in \Pi_{C_1} \cap \Pi_{C_2} \Rightarrow O_{C_1}(\pi) = O_{C_2}(\pi)$.*

Intuitively, if an input sequence produces non-X values for the outputs of both the circuits, then the output values must match for the circuits to be steady-state equivalent. Note that this notion of equivalence is along the lines of *sufficiently old configuration* [11] or *delayed designs* [15].

Two examples of circuits which are considered equivalent with this notion are shown in Figure 1. This notion is different from

*Synopsys Inc., Mountain View, CA 94043
†Cadence Berkeley Labs, Berkeley, 94701
‡Dept. of ECE, University of Colorado, Boulder
‡Department of EECS, University of California, Berkeley, CA 94720

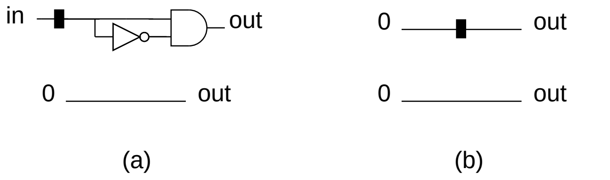

Figure 1: Example of circuits which are steady-state equivalent.

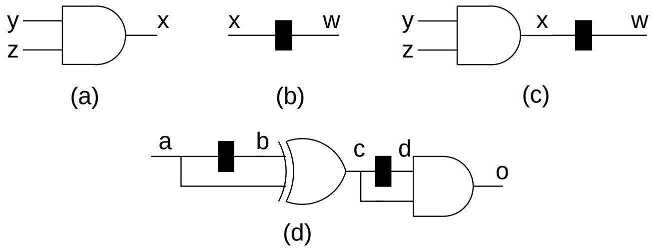

Figure 2: Some acyclic circuits.

3-valued equivalence [5] (it distinguishes both the circuit pairs in Figure 1), and exact 3-valued equivalence [15] (it distinguishes the circuits in Figure 1b).

In the next section we present our technique to derive a combinational representation of sequential circuits. In Section 4 we apply it to sequential circuits without feedback and add feedback in Section 5.

3 From Sequential to Combinational Representation

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 use the following extensions of regular Boolean functions.

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

3.1 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 its outputs. This functionality is given in terms of input values in multiple (but finite) clock cycles.

Definition 3 *A* **clocked Boolean function** *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., $s(t) = f_g(y_1(t), y_2(t), \ldots, y_n(t))$, where $y_1, y_2, \ldots, y_n$ 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. 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,

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

Consider the circuit given in Figure 2(d). The output function is given as:

$$\begin{array}{rcl} o(t) & = & c(t)d(t) \\ d(t) & = & c(t-1) \\ c(t) & = & b(t) \oplus a(t) \\ b(t) & = & a(t-1) \\ o(t) & = & (a(t-1) \oplus a(t))(a(t-2) \oplus a(t-1)) \end{array} \tag{1}$$

Essentially, the output function depends on the value of input a in three different clock cycles. By generating three different inputs for different time instants, we can rewrite the (1) as

$$o = (a_{-1} \oplus a_0)(a_{-2} \oplus a_{-1})$$

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 a finite number of clock cycles. This notion is very similar to the notion of *Timed Boolean Function* given in [8] and recurrence equations in [4]. In [8], 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. In [4], synchronous recurrence equations were proposed for modeling and optimization of sequential circuits. However, their usage in representing and verifying the functionality of sequential circuits has not been seen before.

3.2 Event driven Boolean function

First we define some notation.

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

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.

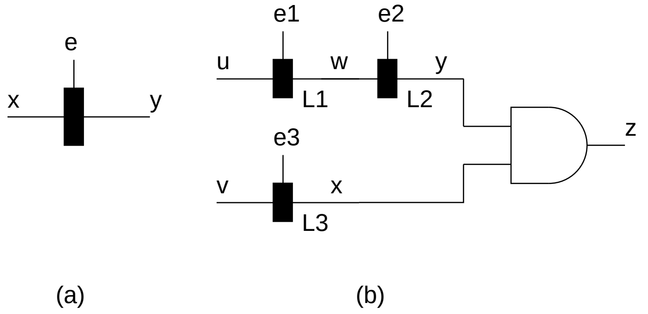

Figure 3: Combinational functionality in the presence of enabled latches: two illustrations.

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

$$\eta([\,]) \;=\; t \quad \text{empty event denotes the current time}$$

$$\eta([p_1, p_2, \ldots, p_n]) \;=\; \begin{cases} -\infty \text{ if } A([p_1, p_2, \ldots, p_n]) = \emptyset \\ \max_\tau \{\tau \in A([p_1, p_2, \ldots, p_n])\} \\ \quad \text{otherwise} \end{cases}$$

where

$$A([p_1, p_2, \ldots, p_n]) \;=\; \{\tau < \eta([p_2, p_3, \ldots, p_n]) : p_1(\tau)\}$$

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 a regular Boolean function.

Definition 4 *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., $s(\eta(E)) = f_g(y_1(\eta(E)), y_2(\eta(E)), \ldots, y_n(\eta(E)))$, where $y_1, y_2, \ldots, y_n$ are the fanin signals of G and f_g represents its functionality.*
- *If s is the output of a latch with fanin signal y and 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 an 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 3(a), the value of signal y, can be represented as $y(\eta([e]))$, since the value of y is equal to the value of x at the time at which e was last active. In Figure 3(b), the functionality of signal z associated with an event E can be obtained as follows:

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

Eqn. 2 indicates that the value of z is equal to the AND of the value of u which has been propagated through both latches L_1 and L_2 and of v which has been propagated through L_3.

In the next section we show how we make use of CBFs and EDBFs to obtain combinational functions for sequential circuits.

4 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 and acyclic circuits with latches trapped within a combinational block (e.g., Figure 2d). We first explain our technique for circuits with regular latches (no load-enable signal) and then describe the case with load-enabled latches.

4.1 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 the primary outputs, we recursively obtain the CBF for each signal as shown in Figure 4. The result of the CBF computation routine is a Boolean formula for each of the outputs in terms of the values of the 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.

Figure 5 conceptually captures the functionality of the algorithm given in Figure 4.

Definition 5 *For an acyclic sequential circuit C, the* **sequential depth** *d is equal to the largest delay for which an input affects the output. Note that d can be less than the topological latch depth (maximum number of latches along a path between an input and an output) due to false dependencies.*

Lemma 4.1 *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. ■

4.1.1 Canonicity of the Formula

Theorem 4.1 (Canonicity of CBF) *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 steady-state as defined in Section 2.2, and equivalence between the CBFs is combinational.*

```
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;
    }
    Cache the result of f(x, d);
    return f(x, d);
}
```

Figure 4: Computing the CBF for the outputs of a feedback free circuit.

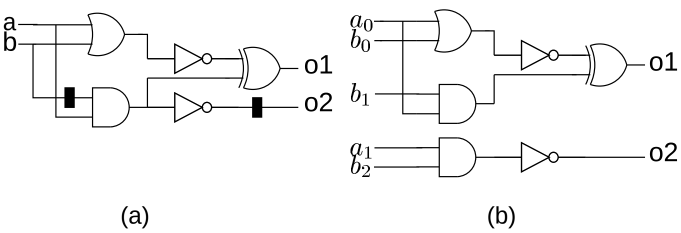

Figure 5: Generating equivalent combinational equivalence problems.

```
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;
    }
    return F(x, E);
}
```

Figure 6: Computing EDBF for the outputs of a circuit.

Proof:

$\Leftarrow$

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.

4.2 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 [10], 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 3.2. 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 6.[1]

[1]Though this algorithm was developed independently, a very similar algorithm was presented in [2]. For completeness sake, we produce it here.

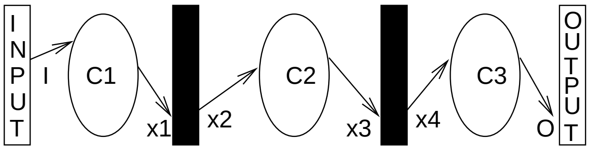

Figure 7: Topological arrangement of latches (black boxes) and combinational blocks (ovals).

4.2.1 Canonicity of the Formula

Lemma 4.2 (Latch count and enable sequence) *Given an acyclic sequential circuit with load-enabled latches and an input/output pair, for each* true *path between the pair, the number of latches and the event associated with the sequence of enabling signals of the latches along the path is invariant during retiming (ala [10]) 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 jump over 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 the illustration in Figure 7. Since 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 consider the path $I \to x_1 \to x_2 \to x_3 \to x_4 \to O$, 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 4.2 (Canonicity of EDBF) *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 6. Then $F_1 \equiv F_2 \Leftrightarrow C_1 \equiv C_2$.*

Proof:

$\Rightarrow$

Assume $C_1 \neq C_2$. 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, $C_{1_k}(\pi) \neq \perp$ and $C_{1_k}(\pi) \neq C_{2_k}(\pi)$. Now, since $C_{1_k}(\pi) \neq \perp$, it implies that for 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 the same in C_1 and C_2 (from Lemma 4.2), $C_{2_k}(\pi) \neq \perp$. Hence using π, we can create an EDBF minterm m, such that $F_1(m) \neq F_2(m)$.

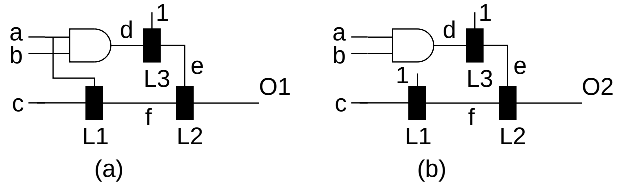

Figure 8: EDBF can lead to false negatives: illustration I.

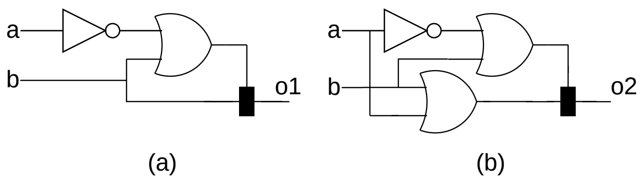

Figure 9: EDBF can lead to false negatives: illustration II.

$\Leftarrow$

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 4.2), 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. ■

The Theorem 4.2 is necessary to validate the algorithm given in [2]. 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 8, two sequentially equivalent circuits are presented. However, their EDBFs would be 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:

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

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,

$$
\begin{aligned}
p(\tau) \geq q(\tau) &\Rightarrow \\
\eta[p(\tau), q(\tau-1)] &= \eta[q(\tau), q(\tau-1)] \\
&= \eta[q(\tau)] \quad (5)
\end{aligned}
$$

Applying (5), on (3) (with $p(\tau) = a(\tau)$ and $q(\tau) = a(\tau)b(\tau)$) and (4) (with $p(\tau) = 1$ and $q(\tau) = a(\tau)b(\tau)$), we get,

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

In [2], a similar rule was presented to minimize the false negatives. This rewriting rule extends the applicability of our technique. However, this rule is not complete, as shown by the next example. In Figure 9, (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:

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

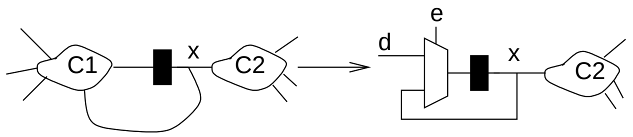

Figure 10: Modeling feedback using a latch with enable and data signals.

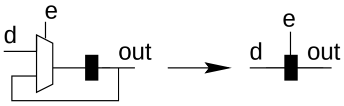

Figure 11: Modeling an enabled latch with extra logic.

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. Note however that our technique produces exact result if only retiming and resynthesis has been performed on the circuit.

5 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 a latch with feedback path with appropriate enable and data signals as shown in Figure 10. Next we derive the conditions under which this modeling is feasible.

Lemma 5.1 (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 10 if and only if $F(x)$ is positive unate in x. Note that e and d are independent of x.*

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

$$F_{\bar{x}} \quad \subseteq \quad d \subseteq F_x \tag{6}$$

can be used as the data signal. On the other hand, using simple Boolean algebra we can show that the value of e is unique and is given as $\bar{F}_x + F_{\bar{x}}$.

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 10 is that a latch fed by a multiplexer can be thought of as an enabled latch as shown in the Figure 11. This gets rid of the feedback path and for our purposes the circuit becomes acyclic. Now we can apply the analysis techniques developed in Section 4.2 for acyclic circuits 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 e as don't care. 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 9, where the decomposition of the next-state function $ax + b$ is different for the two circuits. This can be handled in following ways:

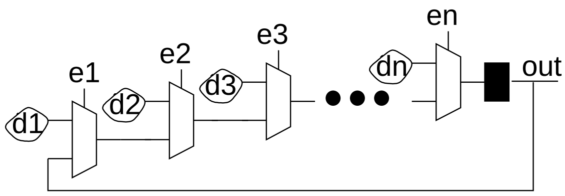

Figure 12: Conditional updating of the latch content.

(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) By performing a canonical decomposition of the enable and data signals. The lemma 5.2 gives a sufficient condition for such a decomposition.

Lemma 5.2 (Data-enable decomposition) *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).*

The feedback modeling as derived in Figures 10 and 11 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 12.

The latches with feedback paths, for which we cannot derive the enabled latch model, are handled in the following way. We find a minimum number of latches that need to be exposed, i.e., need to be made observable, in order to remove the feedback path for these latches. By exposing latches, we treat their outputs as primary inputs and hence the feedback paths are broken, i.e., we cut the latches from the circuit.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.

6 Experimental Setup

6.1 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 output 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. In the current work we have not implemented the technique to identify the latches with feedback paths that satisfy the criterion mentioned in the previous section. Instead, we obtain a minimal set of latches to expose such that the circuit becomes acyclic. The problem of finding the minimum set of vertices to make the circuit acyclic corresponds to the "minimum feedback vertex set problem", which is NP-complete. We used a modified version of the heuristics given in [9].

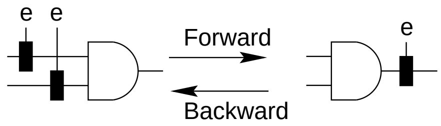

Figure 13: Retiming enabled-latch across gates.

6.2 Retiming

Retiming was done using *Minaret* [12]. This tool only supports the constant delay model (we could not find any efficient public domain retiming tools, that 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 retiming tool which could handle latches with enable signals as proposed in [10] and shown in Figure 13. Hence, in the current work we show results on retiming of circuits with regular latches.

6.3 Combinational Optimization

We used SIS [14] to obtain a minimum delay circuit. A modified version of "script.delay" was used. The modifications were 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 the 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 of each gate to four. The delay models and the fanout limitation changes were achieved by appropriately modifying the library.

6.4 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 as shown in Figure 5. The combinational circuit Figure 5(b) represents the CBF for the sequential circuit Figure 5(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 [13, 7].

7 Experimental Results

Our experiment consisted of the following steps

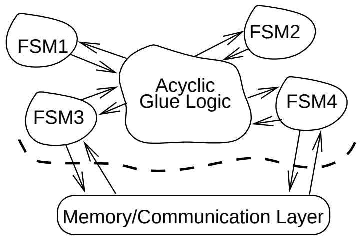

Figure 14: Feedback paths due to interaction with memory and communication layer.

1. Given the sequential circuit (A), modify it appropriately to satisfy constraints 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 the 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 10).

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 the original circuit (A) to obtain circuit (D).

4. We also compared the savings 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. Combinational circuits (H and J) were created (as described in Section 6.4) from circuits (B) and (C) respectively.

6. 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 paths exist due to interaction with the memory and the communication layer as shown in Figure 14. Typically, designers want to keep the boundary between the design and memory/communication layer 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 12. A more detailed functional analysis (based on the next-state function of the latches as explained in Section 5) would lead to fewer number of exposed latches.

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

Table 1: Results on sequential optimization and verification.

A,B,C,D,E,H,J: As described in Section 7
L,%: Latches, Percentage of latches exposed in B
Area/δ: Area (normalized against D) / Delay of the circuit
H vs J: CPU time (in minutes:seconds) for combinational verification between H and J

Example	# Latches	# Exposed	%
ex1	2157	934	43
ex2	100	16	16
ex3	146	56	38
ex4	1437	835	58
ex5	672	305	45
ex6	412	250	61
ex7	453	81	18
ex8	968	470	48
ex9	783	15	2
ex10	634	174	27
ex11	792	369	46
ex12	2206	691	31

Table 2: Number of latches exposed for some industrial circuits.

7.1 Analysis

By analyzing the data given in Tables 1 and 2 we make the 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 pure combinational optimization. In some cases delay values reduce 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 a 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. 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% of the latches; and sometimes as few as 2% of the latches were exposed. As mentioned, these numbers will decrease when positive unateness is used.

8 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 retiming transformations while performing fast verification, as indicated by our experimental results.

Availability of a retiming tool which can move enabled latches across combinational blocks is critical for sequential optimization of industrial circuits. To make our approach exact for arbitrary sequential optimizations, we need to develop a complete technique to distinguish events and combination of events and signals.

References

[1] P. Ashar, A. Gupta, and S. Malik. Using Complete-1-Distinguishability for FSM Equivalence Checking. In *Proc. IEEE/ACM International Conference on*

Computer-Aided Design, Nov. 1996.

[2] G. P. Bischoff, K. S. Brace, S. Jain, and R. Razdan. Formal Implementation Verification of the Bus Interface Unit for the Alpha 21164 Microprocessor. In *Proc. IEEE/ACM International Conference on Computer Design*, 1997.

[3] O. Coudert and J. C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 126–9, Nov. 1990.

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

[5] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen. On Verifying the Correctness of Retimed Circuits. In *Proceedings. The Great Lakes Symposium on VLSI*, pages 277–80, 1996.

[6] 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.

[7] A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts and Heaps. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 263–8, June 1997.

[8] W. Lam. *Algebraic Methods for Timing Analysis and Testing in High Performance Designs*. PhD thesis, University of California Berkeley, Apr. 1993. Memorandum No. UCB/ERL M94/19.

[9] D. H. Lee and S. M. Reddy. On Determining Scan Flip-Flops in Partial-Scan Designs. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 322–5, 1990.

[10] 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.

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

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

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

[14] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[15] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The Validity of Retiming Sequential Circuits. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 316–21, June 1995.

[16] D. Stoffel and W. Kunz. A Structural Fixpoint Iteration for Sequential Logic Equivalence Checking Based On Retiming. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1997.

[17] 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. IEEE/ACM International Conference on Computer-Aided Design*, pages 130–133, Nov. 1990.