# Fast Hazard Detection in Combinational Circuits

***Abstract*— In designing asynchronous circuits it is critical to ensure that circuits are free of hazards in the specified set of input transitions. In this paper, three new algorithms for detecting hazards in combinational circuits are proposed. These algorithms can be used to determine if the circuit is free of combinational hazards without exploring all the gates in the circuits, thus providing more efficient hazard detection. Experimental results indicate that the best new algorithm on average visits only 20% of the original gates, with an average runtime speedup of 1.69 and best speedup of 2.27 (for the largest example).**

## I. Introduction

Asynchronous design has been the focus of renewed interest and research activity because of the potential benefits of improved system performance, low power consumption, and modularity of designs. Several large-scale control and data-path circuits and microprocessors have been designed using asynchronous design [5], [7] and a number of companies are working on experimental asynchronous chips [3], [9].

In designing asynchronous circuits, it is extremely important for correctness to ensure that the circuits are free of hazards in the specified set of input transitions. Much of the original work on combinational hazards was initiated by Huffman, McCluskey, and Unger [10]. Theories of combinational hazards were further extended by Bredeson [2], Beister [1], and many other researchers. Among these efforts, Kung [4] introduced a multi-valued algebra which allows to simulate circuits to detect combinational hazards, both function and logic hazards. Given a circuit with $n$ gates and a set of $t$ input transitions, his method determines the hazard behavior of the circuit in $O(nt)$ time. But even if his method is asymptotically optimal, the algorithm still must traverse the entire circuit (i.e. $n$ gates) once for each of the $t$ specified input transitions, and runtimes may be excessive with large $n$ and $t$. Therefore, faster hazard detection is desirable.

The contribution of this paper is to introduce three fast algorithms to detect hazards in combinational circuits without exploring all the gates of the circuit. Experimental results indicate that the best new algorithm on average visits only 20% of the original gates, with an average runtime speedup of 1.69 and best speedup of 2.27 (for the largest example). These algorithms are the first to be proposed, of which we are aware, which provide systematic techniques for fast combinational hazard detection.

Interestingly, our initial results indicate that, in practice, synchronous CAD tools may introduce very few combinational hazards, and therefore may be a promising starting point in synthesizing asynchronous control circuits. In the future, we intend to develop "hazard repair" algorithms for the few cases where the synchronous flow introduces new hazards.

### A. Problem statement

**Problem 1** *Given a combinational circuit and a set of input transitions, determine if all the circuit outputs are free of hazards for the given input transitions.*

This problem is essentially identical to the problem of simulating the circuit using Kung algebra [4] where each input transition defines an assignment of Kung values to the primary inputs.

In this paper it is assumed that each gate in the circuit is one of AND, OR, and INVERTER. Also, for each input transition, it is assumed that transition types at the primary output gates are known in advance. That is, the user supplies a *transition type* ($0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$, $1 \rightarrow 1$) and the detector indicates if the transition is hazard-free. The goal of these algorithms is to support use of synchronous CAD tools in synthesis, where transition types are known in advance, and the detector must validate hazard-freedom.

### B. Organization of the paper

The remainder of this paper is organized as follows. Section II presents background material that is required to understand the ideas in this paper. Section III presents examples to give a motivation of the ideas used in the algorithms. Section IV presents some new theorems that are used in the algorithms. Our algorithms are described in detail in Section V and we present our experimental results in Section VI. Finally, Section VII presents conclusions and future work.

## II. Background

### A. Combinational circuits

Let $\mathsf{B} = \{0, 1\}$. A *Boolean function* $f$ with $n$ inputs and $m$ outputs is defined as a mapping $f : \mathsf{B}^n \rightarrow \mathsf{B}^m$. This paper considers combinational circuits implementing Boolean functions, which are represented by logic networks. A *logic network* is a directed acyclic graph, $G = (V, E)$, with $V$ partitioned into three subsets called *primary inputs* $V^I$, *primary outputs* $V^O$, and *internal vertices* $V^G$. A combinational Boolean function is associated with the internal vertices and a set of assignments of the primary outputs to internal vertices that denotes which variables are directly observable from output the network [6].

### B. Circuit and delay model

This paper considers combinational circuits having arbitrary finite gate and wire delays (an *unbounded wire delay model*) [8]. That is, the circuits must be glitch-free under all possible finite delay assignments to the gates and wires. A *pure delay model* is assumed as well. A pure delay can delay the propagation of a waveform, but does not otherwise alter it. That is, unlike the *inertial delay model*, this model conservatively assumes that glitches are not filtered out by delays on gates and wires.

### C. Multiple-input changes

An input state with $n$ input variables is represented by a vector $A = (a_1, \cdots, a_n)$ where $a_i \in \mathsf{B}$. An *input transition* where $k$ inputs change is represented by a pair of input states $(A, B)$ where values of two states disagree in $k$ of the $n$ variables [1], [8].

An input transition from input state $A$ to $B$ for a Boolean function $f$ is a *static transition* if $f(A) = f(B)$; it is a *dynamic transition* if $f(A) \neq f(B)$.

### D. Function hazards

A function $f$ which does not change monotonically during an input transition is said to have a *function hazard* in the transition.

**Definition 1** [1], [8] A function $f$ contains a *static function hazard* for the input transition from $A$ to $C$ if and only if $f(A) = f(C)$ and there exists some input state $B \in [A, C]$ such that $f(A) \neq f(B)$.

**Definition 2** [1], [8] A function $f$ contains a *dynamic function hazard* for the input transition from $A$ to $D$ if and only if (1) $f(A) \neq f(D)$ and (2) there exists a pair of input states, $B$ and $C$, such that (a) $B \in [A, D]$ and $C \in [B, D]$, and (b) $f(B) = f(D)$ and $f(A) = f(C)$.

If a transition has a function hazard, no implementation of the function is guaranteed to avoid a glitch during the transition.

### E. Logic hazards

If $f$ is free of function hazards for an input transition, an implementation may still have hazards due to possible delays in the logic realization.

**Definition 3** [1], [8] A circuit implementing $f$ contains a *static (dynamic) logic hazard* for the input transition from $A$ to $B$ if and only if $f(A) = f(B)$ $(f(A) \neq f(B))$ and for some assignment of delays to gates and wires, the circuit's output is not monotonic during the transition interval.

### F. Kung algebra

Kung algebra [4] is a multi-valued algebra which allows one to deal with Boolean values that change over time. A single Kung value represents an equivalence class of waveforms with the same basic behavior, i.e. same start value, same end Boolean value, and the same hazard behavior. For example, the Kung value $D+$ represents a waveform whose start value is 0, whose end value is 1, and includes at least one glitch during the value change, say $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$. Similarly, the the Kung value $S0$ represents a waveform whose start value is 0, whose end value is 0, and includes at least one glitch during the value change, say $0 \rightarrow 1 \rightarrow 0$. Figure 1 shows the set of

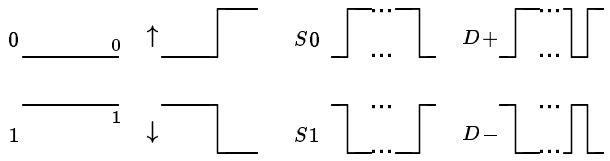


Fig. 1. Waveforms represented by Kung values

Kung values, and the corresponding waveforms represented by these values.[1]

Formally, Kung algebra is a quintuple $(\mathsf{K}, +, \cdot, 0, 1)$ where $\mathsf{K} = \{0, 1, \uparrow, \downarrow, S0, S1, D+, D0\}$ and two operations $+$ and $\cdot$ are defined using a partial order $\leq$ on $\mathsf{K}$. Intuitively, given $a, b \in \mathsf{K}$, $a \leq b$ means that "$b$ is more hazardous than $a$" and $a+b$ returns the maximal (with respect to $\leq$) Kung value which represents a possible waveform generated by "Boolean $+$" over any waveforms of $a$ and $b$.

A Kung value is *static* if it is either 0 or 1. Otherwise, it is called *non-static*. If an output wire of a gate $u$ is an input wire of a gate $v$, we say $v$ is a *parent gate* or an *output gate* of $u$.

A *path* in a combinational circuit is defined as an alternating sequence of gates and wires where consecutive gate and wire are incident in the circuit. A *static path* is defined to be a path where all the wires have static values. And a *non-static path* is a path where all the wires have non-static Kung values.

### G. Constant propagation

Constant propagation is a common technique for simplifying the circuit using constant inputs. It consists of detecting constant operands and pre-computing the results of operations assuming that constant operand [6]. In this paper constant propagation is extended to work with Kung algebra, as follows. In *static constant propagation*, a circuit is simplified by propagating only static input values, 0 and 1, and this propagation is identical to that of Boolean constant propagation. As a result of static constant propagation, a reduced circuit may be obtained. Additionally, *non-static constant propagation* is defined as the technique that reduces a circuit by propagating only non-static input values. A reduced circuit may also be obtained in this case.

## III. Motivational Examples

In this section, several examples are presented that illustrate some of the key ideas of the proposed hazard detection algorithms. Examples show how hazard detection can be simplified without exploring all the gates and wires.

### A. Example 1: Forward algorithm for static transitions

The "forward" algorithm reduces a circuit through constant propagation first. For static transitions (i.e. $0 \to 0$, $1 \to 1$), only one step is required: a single pass of constant propagation. If the circuit is reduced to a single wire, then a theorem in Section IV confirms that the circuit is hazard-free. However, if the circuit is not reduced to a single wire, then the theorem states that the circuit is hazardous.

Figure 2 shows a circuit marked with primary input values specified by a input transition $(A, B) = ((0, 1, 0, 1), (0, 0, 1, 1))$ with four input variables $a, b, c$, and $d$. The circuit implements the Boolean function $f = ab + ac + d$, where $f(A) = 0$ and $f(B) = 0$. By applying constant propagation, the circuit on the left side is reduced to a single wire with Kung value 1, shown on the right. Thus, by the theorem, it can be concluded that the circuit is hazard-free.

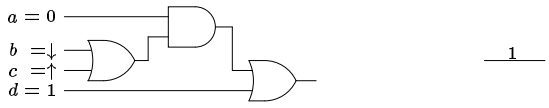


Fig. 2. Example 1: Forward algorithm for static transitions

[1]Kung value set also includes a symbolic value, $*$, which represents arbitrary behavior. This value is not needed in the constrained problem in this paper.

### B. Example 2: Forward algorithm for dynamic transitions

When a circuit has a dynamic output transition for a given input transition, static constant propagation cannot reduce the circuit to a single wire. The forward algorithm for the dynamic case has two distinct steps: first (as above), to reduce the circuit; second, to perform hazard detection on the reduced circuit.

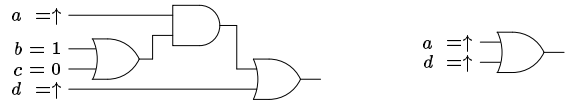


Fig. 3. Example 2: Forward algorithm for dynamic transitions

Figure 3 shows a combinational circuit marked with primary input values specified by a dynamic input transition $(A, B) = ((0, 1, 0, 0), (1, 1, 0, 1))$. First, static constant propagation reduces the left circuit to the right one. Second, a hazard detection algorithm is then run on the resulting reduced circuit: any such algorithm can be used, such as the original Kung algorithm or one of the new fast hazard detection algorithms. (In this paper, the "CORE" algorithm of Section V-B will be applied to the reduced circuit.)

More details on the forward algorithm, for both static and dynamic transitions, are presented in Section V-A.

### C. Example 3: CORE algorithm

The "CORE" algorithm is a recursive algorithm where the flow starts at the primary outputs towards the primary inputs. When the flow hits primary inputs, actual Kung input values are collected and propagated back towards the primary outputs. The actual Kung operations are performed during this forward flow.

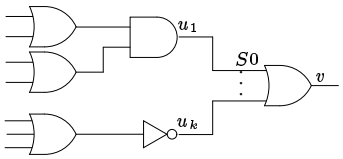


Fig. 4. Example 3: CORE algorithm

During the backward flow of the CORE algorithm, constraints can be propagated to the primary inputs which are used to accelerate hazard detection. As an example, suppose that the circuit shown in Figure 4 makes a $0 \to 1$ dynamic output transition at node $v$ for a given input transition. Then, when visiting node $v$, a *constraint* $C$ is generated (called HF-constraint) at the node output. Intuitively, this constraint captures the *desired* output behavior, i.e. when the output is hazard-free. This constraint can be described as a set of Kung values, in this case $\{\uparrow\}$. This constraint is also propagated backwards in the recursive flow.

Now, assume the flow hits the primary inputs, and then passes Kung values back towards the primary output. In particular, assume the algorithm eventually returns an $S0$ value at node $u_1$. At this point, node $v$ can be *partially evaluated* based on this single input value. The resulting value of $v$ is also a set of Kung values, in this case $\{1, S0, D+, D-, S1\}$. This set excludes the possibility of three hazard-free transitions, $\uparrow, \downarrow, 0$, since the $S0$ input now excludes these hazard-free output transitions.

Note that this partial value set is *disjoint* from the original HF-constraint $C$ at node $v$. Therefore, even though only one of the inputs of node $v$ has been evaluated, it can be concluded immediately that the output of $v$ is hazardous, due to a *constraint violation*: the partial value and the constraint are contradictory.

To conclude, constraint propagation and partial evaluation together can be used to deduce hazard properties in the circuit with only partial information, and thereby help to prune the search space.[2] Details of the CORE algorithm are presented in Section V-B.

## IV. Theorems on Constant Propagation

In this section, theorems and corollaries that are used to simplify hazard detection are presented based on constant propagation, which

[2]Another set of constraints, called RF-constraints, is also proposed and will be shown to be useful.

is a common technique for circuit simplification.[3] These theorems will be used in two of the proposed algorithms: the forward algorithm and the hybrid algorithm.

**Theorem 1** *Given combinational circuit $G$ and input transition $t$, let $G'$ be the circuit reduced from $G$ through static constant propagation. If $G$ makes a static output transition in $t$, $G$ is free of a hazard in $t$ if and only if either $G'$ is a constant* 0 *or a constant* 1.

*Proof:* IF part of the theorem is trivially true and only the ONLY IF part of the theorem is proved. To the contrary, assume that $G$ is free of a static hazard but $G'$ has at least one gate. Let $v$ be the primary output gate in this *non-empty* $G'$. Then $v$'s gate output should be a non-static value since otherwise $v$ would be removed in constant propagation. This is contradictory to the assumption that $G$ is free of static hazards and completes the proof. ■

Example 1 of Section III illustrates the theorem. That is, if a circuit makes a hazard-free output transition then there should exist *static* paths from the primary input to the primary output which *determine* the circuit outputs.

One useful observation resulting from Theorem 1 is that when Kung simulation is performed on the reduced circuit $G'$ *all the remaining wires in $G'$ have non-static values* in the given input transition. This is because all the wires in $G$ with static values in the given input transition is in static constant propagation.

**Corollary 1** *Let a combinational circuit $G$ and an input transition $t$ be given. If $G$ is reduced into $G'$ through static constant propagation and $G$ has a dynamic output transition in $t$, $G'$ is free of dynamic logic hazards in $t$ if and only if no wire in $G'$ glitches in $t$.*

*Proof:* Only the IF part of the theorem is proved since ONLY IF part of the theorem is trivially true. Note that no wire in $G'$ has a static value since all the remaining primary inputs have non-static values. And if any gate glitches in its output, this glitch cannot be filtered at a later gate since there exist no static signals which can suppress this glitch. Thus, any glitch at a gate implies a glitch at the circuit output. ■

While the above theorem and corollary are related to static constant propagation, another theorem is related to non-static constant propagation.

**Theorem 2** *Let a combinational circuit $G$ and an input transition $t$ be given. Then, $G$ is free of static hazards in $t$ if there exist no non-static paths from primary inputs to the primary output.*

*Proof:* If there exist no non-static paths to the primary output then all the paths to the primary output should end with static values. Since all the inputs to the primary output have static values, the primary output value is static and the circuit makes a hazard-free static output transition in $t$. ■

Thus, when all the *non-static* values are blocked in the middle of the circuit during non-static constant propagation it can be concluded that the circuit is *hazard-free*.

## V. Hazard Detection Algorithms

In this section three hazard detection algorithms are presented: the forward, CORE, and hybrid algorithms. The forward and hybrid algorithms are based on constant propagation, and the CORE algorithm is a recursive algorithm which includes constraint propagation. The goal of these algorithms is to determine the circuit output and its hazard behavior by visiting a small number of vertices in the circuit. Two of these algorithms, forward and hybrid, make use of the theorems presented in Section IV.

### A. Forward algorithm

The "forward" algorithm propagates Kung values in a single pass in a topological order of vertices to determine the circuit output. It starts by reducing the circuit through static constant propagation. If the circuit has a static transition for the given input transition, the circuit output value can be directly determined using Theorem 1: the circuit is (static) hazard-free *if and only if* constant propagation reduces the circuit to a single wire.

If the circuit has a dynamic transition for the given input transition, a second step is required since static constant propagation alone cannot determine the hazard behavior of the circuit output. In this case, a hazard detection algorithm is applied to the reduced circuit.

Note that, from Corollary 1, the circuit has a hazard when we encounter any hazardous Kung value in the reduced circuit. Therefore, in performing hazard detection on the reduced circuit, a special *simplified* detection algorithm can be used, which terminates as soon as any hazardous (non-static) Kung value is produced.

```
FORWARD(G, t)
 1  // reduce the circuit using static constant propagation
 2  G' ← FORWARDPROPAGATE(G, t)
 3  if (static output transition)
 4    then if (G' is empty)
 5           then return "Hazard-free"
 6           else return "Hazardous"
 7    else  // optional step for dynamic output transition
 8          return HAZARDDETECT(G', t)

FORWARDPROPAGATE(G = (V, E), t)
 1  I ←  the set of primary inputs with static values;
 2         for input transition t;
 3  V' = V;  E' = E;
 4  while (I ≠ ∅)
 5  do select a primary input v in I;
 6     if out(v) = ∅
 7       then I ← I − {v};
 8                 // pick a gate having primary input v;
 9     select a gate w in out(v);
10     out(v) ← out(v) − {w};  // remove w from v's fanout set
11     while (val(v) determines w's gate output)
12     do if (w is the primary output)
13           then return G' = (∅, ∅);  // circuit reduced to wire
14           else  V' ← V' − {w};  // delete traversed gate
15                 v ← w;
16                 select a gate w in out(v);
17                 out(v) ← out(v) − {w};
18                 E' ← E' − {⟨v, w⟩};
19     return G' = (V', E')
```

Fig. 5. Outline of the forward algorithm. In the algorithm, $out(v)$ is the set of output gates of a $v$ and $val(v)$ is the Kung value of $v$.

Figure 5 shows an outline of the forward algorithm. FORWARD function takes a circuit and an input transition and determines the hazard behavior of the circuit using FORWARDPROPAGATE function. In FORWARD, HAZARDDETECT can be CORE algorithm presented in next section or straightforward Kung simulation algorithm. FORWARDPROPAGATE function returns either an empty or a reduced circuit through constant propagation. Paths from static primary inputs to the primary output are explored to get a reduced circuit.

Line 5 selects a primary input and performs a depth-first forward constant propagation until either the circuit is reduced to a wire (in line 13) or no further propagation is possible. Nodes and edges are deleted throughout this process. If nodes remain, again some primary input (possibly the same one) is selected (line 5) and the process continues.

*1) Biasing the direction of forward flow:* In forward algorithm, search direction is biased in the hope that chosen path will simplify the circuit faster than arbitrary paths. When selecting a path in line 6 and 13 of Algorithm FORWARDPROPAGATE, two heuristics are used for biasing the direction of forward flow. First, paths with shorter distance to the circuit output are preferred in the hope that this path will determine the circuit output with smaller cost of path exploration.

Second, paths starting with a node which is contained in a larger block are preferred, where a block is defined to be a sub-circuit with no fanout gate except at the boundary (A more detailed explanation of block is given in Section V-D.2). Rationale behind this decision is that path explorations through larger blocks can proceed farther without encountering multi-fanout gates since all the vertices which are not the root of the block have fanout 1.

*2) Static vs. non-static constant propagation:* In the forward algorithm, only static constant propagation can be used safely. When a forward flow hits a gate in non-static constant propagation, it has to be determined whether the gate output is non-static or not: we can continue propagation only if gate output is non-static.

But in non-static constant propagation only non-static values are propagated and, thus, it has to be determined whether the gate

[3]To simplify exposition, theorems and optimization techniques discussed in the paper assume that the circuits have single primary outputs. But these theorems and techniques can be easily extended to support circuits with multiple outputs.

output is non-static using only the non-static inputs. But any forward algorithm has an one-sided error in this determination: since "non-staticness" of gate output cannot be determined without looking at *all* inputs (including static inputs) there's a possibility that non-static constants are not propagated properly.

But in static constant propagation, it can be determined whether a gate output is static or not using only the static inputs. Thus, when the forward algorithm is used, static constant propagation is used always.

### B. CORE: A recursive algorithm

The "CORE" algorithm is a recursive algorithm for combinational hazard detection, where the control flow now starts at the primary output. During the *backward* flow, several *constraints* are computed and propagated down towards the primary inputs: these constraints are used to simplify Kung computation in the final forward flow of the algorithm.

Figure 6 shows an outline of the CORE algorithm. The CORE function takes a gate and its current constraints as its arguments, and it returns the Kung output value of the gate after recursive computation. When invoked with a gate $v$ as its argument, it selects one gate from the set of $v$'s input gates, evaluate the selected gate recursively, and eventually updates $v$'s partial value based on this value. Based on this updated partial value, the evaluation of gate $v$ can either be terminated or else the recursive search can continue to explore $v$'s other input fanin cones. To determine the hazard behavior of a circuit's primary output, the top-level CORE function is invoked with the primary output gate as its argument.

```
CORE(v, C, R)
 1  // C is a HF-constraint, R is an RF-constraint
 2  if (val(v) is already known or v is a primary input)
 3     then return val(v);
 4     else  I ← set of input gates;
 5           K ← initial partial value of v;
 6           while (I ≠ ∅)
 7           do select u from I;
 8              C' ← UPDATEHFCONSTRAINT(C, v);
 9              R' ← GENERATERFCONSTRAINT(R, K, v);
10              k ← CORE(u, C', R');
11              if k ≠ "Useless value"
12                 then // Only re-evaluate v if input useful
13                      K ← UPDATEPARTIALVALUE(k, K);
14                      if (|K| = 1)  // early termination
15                         then return K;
16                      if (K ∩ C = ∅)  // HF-constraint check
17                         then exit "Hazardous circuit";
18                      if (K ∩ R = ∅)  // RF-constraint check
19                         then return "Useless value";
20           return K
```

Fig. 6. Outline of the CORE algorithm.

*1) Partial value computation:* The process of evaluating a gate can be viewed as the process of *refining* the set of possible output values of the gate, as the values of one gate input after another are computed. This process stops when the cardinality of the set of possible output values becomes 1 (i.e. the gate output is fully evaluated to a single Kung value). The *partial value* of a gate $v$ is defined to be the set of possible Kung output values at a specific point in time during the recursive evaluation. As each new input $u$ to node $v$ is evaluated (to value $k$ in line 10), a new partial evaluation of node $v$ is performed (line 13), resulting in an updated partial value set $K$ for node $v$. If an input indicates that its value will not aid in further refinement of the value of node $v$ (line 11), then this input is skipped and the next input is checked.

As an optimization to this evaluation step, a decoupled three-part update is performed: to compute the *start value*, *end value*, and *hazard behavior* of the node. This triple of partial information is equivalent to, and captures, the corresponding Kung set $K$ of the node, and can be efficiently computed incrementally, as new inputs arrive.

*2) Early termination:* This is analogous to *short circuit evaluation* in evaluating programming language expressions. While evaluating a gate, if the value of the gate out is already final it is safe to stop the evaluation.

That is, when the partial value of a gate contains a single value, we can terminate gate evaluation since the partial value cannot be further refined. This happens when a controlling input to a gate has been encountered.

*3) Constraints propagation:* There are two kinds of constraints that are propagated during the backward flow of the algorithm: HF-constraints and RF-constraints. An *HF-constraint* is a global constraint that should be satisfied at each gate to make the circuit output free of hazards while *RF-constraint* is a local constraint that should be satisfied at each gate to make its output useful in refining the partial value of its output gate. When an HF-constraint is not satisfied the entire circuit evaluation can be aborted and it can be concluded that the circuit is hazardous. And when an RF-constraint is not satisfied the gate evaluation can be aborted skipping exploration of remaining input cones.

More precisely, an HF-constraint passed to a gate $v$, $\mathrm{HF}(v)$, is a set of Kung values that $v$ can accept as its input to make the circuit output free of a hazard (an example of HF-constraint is presented in Example 3 of Section III). Since output transition types of primary output gates are assumed to be known in advance, we can generate HF-constraint for primary output gates. Furthermore, this HF-constraint can be propagated down towards the primary inputs as follows. Given two adjacent gates $v$ and $w$, where $w$ is the output gate of $v$,[4] the HF-constraint passed to gate $v$, called $\mathrm{HF}(v)$, is defined as

$$\mathrm{HF}(v) = \bigcup_{k \in \mathrm{HF}(w)} h(gt(w), k),$$

where $gt(v)$ is the *gate type* of node $v$ (e.g. AND, OR, etc.); $h$ is a deterministic mapping function (for which lookup tables are precomputed); and $HF(v)$ is computed through this mapping on each Kung value $k$ in the given constraint set $HF(w)$.

An RF-constraint, passed from a parent gate $w$ to one of its input gates $v$, denoted by $\mathrm{RF}(v)$, is a set of possible Kung values that gate $v$ can provide to parent $w$ that can be *useful* to $w$ in further refining its partial value. Intuitively, node $w$ passes a constraint set $RF(v)$ to each of its input nodes $v$, indicating desired input values which may facilitate further partial evaluation of $w$. During recursive evaluation of node $v$, the partial value of $v$ is incrementally updated. The RF-constraint, $RF(v)$, is used to *test* whether node $v$'s partial value will be potentially useful in refining parent $w$'s value. If the partial value of $v$ has *no common Kung value* with the constraint set $\mathrm{RF}(v)$, then $v$'s final Kung value will be useless to parent $w$'s further evaluation, and therefore computation at node $v$ can be aborted without affecting the evaluation result of $w$.

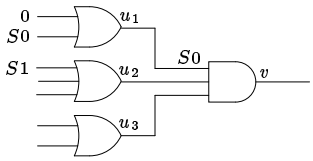


Fig. 7. Gate evaluation example

Figure 7 shows an example of gate evaluation using RF-constraints. Let us assume that the control of the algorithm has just reached the AND gate $v$. When $u_1$ is visited first and it produces a Kung value $S0$, a check if the value is a controlling value for this gate is performed. Since $S0$ is not a controlling input to the AND gate, exploration continues after refining the partial value of $v$ into $\{0, S0\}$. Then, the second input gate $u_2$ is visited with an RF-constraint, $\{0\}$. Let's consider the point when $u_2$ is visited and its first input has just yielded $S1$, making the partial value of $u_2$ into $\{S1, 1\}$. Since $\{S1, 1\} \cap \{0\} = \emptyset$, it is obvious that $u_2$ cannot produce a value that can refine $v$'s partial value. Thus, exploration of the *rest of the inputs of* $u_2$ can be skipped and we can determine the output of $v$.

*4) Biasing the search direction:* In the backward flow there may exist multiple choices on which input gate to visit first. As in the forward algorithm, exploration of the circuit is biased to maximize of effectiveness of the optimizations discussed above.

[4]for simplicity of the explanation, we assume no multiple fanout gates; the definition can be generalized to multiple fanout gates as well

Two heuristics are used for this purpose: *cone size heuristic* as a primary heuristic and *fanin heuristic* as a secondary heuristic. Input gates with smaller cone size are preferred in the hope that a controlling input will be encountered with smaller circuit exploration cost. When there's a tie, an input gate with larger fanin is preferred in the hope that exploration of *more* input cones can be skipped when gate evaluation stops by early termination or by conflicts in RF-constraints.

### C. A hybrid algorithm

Finally, a hybrid algorithm is proposed combining the forward and CORE algorithm (say, "backbone" of the algorithm is forward – performing topological-order traversal from the primary inputs to the primary outputs. The new idea, however, is that when the forward flow is stalled, a call to recursive CORE is performed at the gate where the flow stalled to determine whether the flow should continue or not.

Hybrid algorithm is also based on forward propagation of Kung values starting from the primary inputs towards the primary output in a topological order of vertices. But during this forward traversal of the circuit, there may exist sub-flows which are recursive 'CORE' flow. It is also based on constant propagation like forward algorithm but either static or non-static constant propagation can be used.

When static constant propagation is used, an additional step is required when the circuit output transition is dynamic, as in the case of forward algorithm. When non-static constant propagation is used, no additional step is required. Non-static constant propagation reduces the circuit into a non-static Kung constant when the circuit output makes a dynamic transition or a hazardous static transition. When the reduced circuit is not a constant, it can concluded that the circuit output makes a hazard-free static transition using Theorem 2.

```
HYBRID(G, t)
1  K ← HYBRIDPROPAGATE(G, t)
2  if (K equals "Hazard-free")
3     then return "Hazard-free"
4     else if (K ∈ {↑, ↓})
5             then return "Hazard-free"
6     else if (K ∈ {D+, D−, S0, S1})
7             then return "Hazardous"

HYBRIDPROPAGATE(G, t)
 1  I ← {v ∈ V^I : val(v) ≠ 0 and val(v) ≠ 1};
 2  V' = V; E' = E;
 3  while (I ≠ ∅)
 4  do select a gate v from I;
 5     I ← I − {v};
 6     select a gate w in out(v);
 7     out(v) ← out(v) − {w};
 8     while (val(v) is non-static)
 9     do v ← w;
10        select a gate w in out(v);
11        out(v) ← out(v) − {w};
12        K ← HAZARDDETECT(v, t);
13        if (v is the primary output)
14           then return K;
15     return K;
16  return "Hazard-free"
```

Fig. 8. Outline of the hybrid algorithm with non-static constant propagation.

Figure 8 shows an outline of the hybrid algorithm with non-static constant propagation. It is assumed that HAZARDDETECT algorithm in the outline returns the Kung value of the gate in the given input transition.

HYBRID function takes a circuit and an input transition and determines the hazard behavior of the circuit using HYBRIDPROPAGATE function. HYBRIDPROPAGATE function returns the Kung output value of the primary output gate Paths from primary inputs to the primary output are explored to get a Kung value but, unlike FORWARDPROPAGATE, we use HAZARDDETECT function to proactively determine the gate output of the gate encountered in the path being explored.

As discussed above, difference between the hybrid algorithm and the forward algorithm approach lies in what happens when a gate is visited in a forward flow. In the case of the forward algorithm, when the gate output cannot be determined immediately, it stops traversal and initiates another path exploration from the primary input. In the case of the hybrid algorithm, when the gate output cannot be determined it proactively determine its gate value (say, using the CORE algorithm), and decide whether to continue or not.

A major advantage of this approach is that now non-static constant propagation can be used safely. This is because by forcibly evaluating each gate at the time the gate is visited *all* non-static constants can be propagated towards the primary output. Note that in the forward algorithm, there is a possibility that some non-static constants may not be propagated. And hybrid algorithm also uses the biasing technique presented in Section V-A.1.

### D. Preprocessing

Several preprocessing steps are executed before hazard detection algorithm. These steps are used to expedite hazard detection either by reducing the size of the circuit or by gathering information that is useful for faster hazard detection.

*1) Circuit transformation:* Preprocessing step used in the algorithms transforms the circuit into one with fewer gates. Here, only hazard-preserving transformations transformations [4] should be used since otherwise the hazard behavior of the circuit may be changed. In this step, associative laws are used to merge adjacent AND gates and adjacent OR gates, which is a hazard-preserving transformation that does not change the hazard behavior of the circuit.

*2) Block graph construction:* Given a combinational circuit $G = (V, E)$, we build a block graph $B_G = (V_B, E_B)$ where $V_B$ is the set of vertices in $V$ with multiple fanout and $B_G$ is a subgraph of $G$ induced by $V_B$. A vertex $v \in V$ is said to be *contained in* $v_B \in V_B$ if and only if $v_B$ is the nearest vertex $w$ in $V$ such that the cone rooted at $w$ contains $v$.

Block graphs are used for two purposes in forward and hybrid algorithms: they are used for biasing the search direction and eliminating dead paths. A *dead path* in these algorithms is defined to be a path to a gate whose gate output has already been determined and it is safe to exclude in path exploration.

### E. Precomputation

Precomputation is another technique for reducing the size of the circuit. Unlike circuit transformation where the circuit reduction is valid over all input transitions, precomputation reduces the size of the circuit for a subset of input transitions.

A set $T$ of input transitions can be partitioned into $k$ partition blocks $\{T_i : 1 \leq i \leq k\}$ such that input transitions in the same partition block have *similar* input assignments. In *precomputation*, the circuit is partially evaluated with the primary input values that are common to all the input transitions in a partition block to obtain a reduced circuit.

For example, for a circuit of 4 input variables, $x_1, \cdots, x_4$, assume that three input transitions in a partition block defines the following three assignments of Kung values to the input variables: $(0, 1, 1, \uparrow)$, $(0, 1, 0, \downarrow)$, and $(0, 1, \uparrow, 0)$. Then, the first two inputs $x_1$ and $x_2$ have the same values for three input transitions. In this case, the circuit can be reduced by propagating two constants $x_1 = 0$ and $x_2 = 1$, and hazard detection algorithm can be executed on the reduced circuit for each input transition in this partition block.

For precomputation, the set of input value assignments is sorted in lexicographic order of Kung values. And for a fixed integer $k$, the set of input transitions is partitioned into partition blocks with $k$ consecutive input transitions in the sorted set. There is a trade-off in determining the value of $k$. If $k$ is large, the number of common values in a partition block gets smaller and the circuit reduction ratio gets smaller. If $k$ is small, the extra overhead required for reducing the circuit with common values gets relatively large. With several experiments, the value of $k$ has been decided to be 5.

## VI. EXPERIMENTAL RESULTS

Algorithms proposed in Section V were implemented and hazard detection experiments were performed. Programs were written in C and experiments were conducted on a 800MHz Celeron machine with 256MB RAM running Redhat Linux 7.3. The programs take combinational circuits in Berkeley BLIF format and input transition sets in BTRANS format as their input. The circuits used for experiments were synthesized from large Burst-mode asynchronous specifications [8] using the Minimalist toolset and the SIS toolset. In particular, state minimization and state assignments were performed by asynchronous

| name | # inputs∗ | # outputs∗ | # gates | # input transitions | Forward algorithm | CORE algorithm | Hybrid algorithm | total # hazardous outputs∗∗ | % hazardous outputs |
|---|---|---|---|---|---|---|---|---|---|
| p1 | 13 (32) | 14 (33) | 295 | 84 | 20.7 | 37.2 | 18.2 | 18 | 0.64 |
| p2 | 8 (18) | 12 (22) | 132 | 56 | 18.9 | 46.0 | 21.4 | 10 | 0.81 |
| m3 | 19 (34) | 10 (25) | 337 | 104 | 17.9 | 26.1 | 16.0 | 21 | 0.80 |
| pscsi | 10 (34) | 5 (29) | 784 | 124 | 24.9 | 31.9 | 22.8 | 112 | 3.11 |
| iccd-scsi | 9 (31) | 5 (27) | 851 | 186 | 21.3 | 24.3 | 20.0 | 136 | 2.70 |
| **average percentage of gates visited** | | | | | 20.7 | 33.1 | 20.0 | | 1.61 |

(a) single-output implemented circuits

| name | # inputs∗ | # outputs∗ | # gates | # input transitions | Forward algorithm | CORE algorithm | Hybrid algorithm | total # hazardous outputs∗∗ | % hazardous outputs |
|---|---|---|---|---|---|---|---|---|---|
| p1 | 13 (17) | 14 (18) | 193 | 84 | 26.2 | 43.0 | 25.6 | 38 | 2.51 |
| p2 | 8 (12) | 12 (16) | 75 | 56 | 21.5 | 47.2 | 28.4 | 23 | 2.56 |
| m3 | 19 (23) | 10 (14) | 156 | 104 | 20.8 | 29.0 | 21.1 | 37 | 2.54 |
| pscsi | 10 (15) | 5 (10) | 290 | 124 | 23.3 | 37.6 | 31.9 | 165 | 13.31 |
| iccd-scsi | 9 (13) | 5 (9) | 276 | 186 | 23.2 | 33.5 | 29.0 | 169 | 10.00 |
| **average percentage of gates visited** | | | | | 23.0 | 38.0 | 27.2 | | 6.18 |

(b) multi-output implemented circuits

∗ The numbers in parentheses in the '# inputs' column and '# outputs' column counts also the state variables after state minimization and assignments.
∗∗ If an output is hazardous for $k$ distinct input transitions, it contributes $k$ to the total number of hazardous outputs. Hence, "% hazardous outputs" column counts total hazardous outputs as a percentage of $k \times m$, where $m$ is the total number of primary outputs.

Fig. 9. Experimental result: percentages of gates visited.

tools while two-level and multilevel synthesis as well as technology mapping were performed using non-hazard-free synchronous tools.[5]

| name | # gates | Baseline | Forward | CORE | Hybrid |
|---|---|---|---|---|---|
| p2 | 132 | 1 | 0.882 | 0.794 | 0.911 |
| p1 | 295 | 1 | 0.712 | 0.613 | 0.643 |
| m3 | 337 | 1 | 0.629 | 0.555 | 0.574 |
| pscsi | 784 | 1 | 0.589 | 0.557 | 0.531 |
| iccd-scsi | 851 | 1 | 0.483 | 0.440 | 0.448 |
| **average runtime ratio** | | 1 | 0.658 | 0.591 | 0.621 |
| **average speedup** | | 1 | 1.52 | 1.69 | 1.61 |

Fig. 10. Experimental result: relative runtime ratio and speedup for single-output implemented circuits when the runtime of the straightforward Kung simulation program is taken to be 1.

Each program starts by preprocessing the input circuit. After preprocessing, the set of input transitions is partitioned into partition blocks of size 5. For each partition block, precomputation is applied, and the circuit is reduced using static constant propagation with the common input values. Then, the corresponding hazard detection algorithm is applied to the reduced circuit for each input transition in the partition block.

Figure 9 shows a table with the percentage of gates visited by the proposed algorithms. For gate visit counts, both the forward algorithm and the hybrid algorithm visited less than 30% of the gates.

Figure 10 shows the resulting runtimes. Runtime in the table was measured using UNIX `time` program as the sum of user time and system time required for program execution and includes not only the circuit search time but also the fixed overheads of reading in a circuit, initialization, and updating information in a data structure.

Of the three program implementations, the CORE algorithm had the best average speedup of 1.69 having a speedup as high as 2.27 in the largest example. Proposed algorithms used all the optimizations discussed in the paper and were compared to the baseline algorithm which is essentially a recursive Kung simulation program. Two versions of a straightforward Kung simulation program were developed as potential baselines: one is a recursive program and another is one that visits the circuit in topological order from the primary inputs to the primary outputs. The baseline program was taken to be the recursive one which ran consistently faster than the latter1.

It can be observed from the table that, as the number of gates increases, the runtime ratio improves, approaching the actual percentage of gates visited. This trend is promising, indicating greater improvements on larger circuits. The runtime consists of two parts: the time for reading circuits, parsing, and several initializations and the time required for circuit exploration. The reason that the performance was not good in small circuits can be attributed to the fact that the latter time was not a dominant portion of the runtime in small circuits. As the number of input transitions and the number of gates sufficiently large, the time required for circuit exploration will dominate the runtime, the relative portion occupied by initialization functions will become negligible, and therefore, the relative runtime ratio tends to approach the percentage of gates visited.

[5]Both the singly-implemented circuits and multi-output-implemented circuits were synthesized. Single-output implementation examples are generated using state minimization with no fedback outputs (`min-states` in Minimalist), state assignment with output implemented singly with optimal encoding (`assign-states -s -O -P` in Minimalist), two-level logic minimization with outputs implemented singly (`espresso -Dso` in SIS), multilevel logic minimization (`script-rugged` in SIS), and technology mapping (`map` in SIS). This flow highlights the use of synchronous CAD tools in the synthesis of asynchronous circuits. Multi-output implementation examples are generated using state minimization (`min-states`), state assignment with multi-output implementation (`assign-states -P`), two-level logic minimization (`espresso`), multilevel logic minimization (`script-rugged`), and technology mapping (`map`).

## VII. Conclusions

This paper presented three new algorithms for detecting hazards in combinational circuits. These algorithms detect hazards visiting only a small portion of the circuit and works faster than the straightforward Kung simulation algorithm. Experimental results are promising: the best new algorithm on average visits only 20% of the original gates, with an average runtime speedup of 1.69 and best speedup of 2.27 (for the largest example).

In the future, we plan to further improve the implementation of the three new algorithms, to better approach the speedup potential suggested by the dramatic reduction in gate visitation counts. Furthermore, our initial results indicate that, in practice, synchronous CAD tools may introduce very few combinational hazards, and therefore may be promising in synthesizing asynchronous control circuits. We intend to develop "hazard repair" algorithms to fix the few cases where the synchronous flow introduces new hazards.

## References


[1] J. Beister. A unified approach to combinational hazards. *IEEE Transactions on Computers*, C-23(6):566–575, June 1974.

[2] J. G. Bredeson. Synthesis of multiple input-change hazard-free combintional switching circuits without feedback. *Int. J. Electron.*, 39(6):615–624, 1975.

[3] B. Coates, J. Ebergen, J. Lexau, S. Fairbanks, I. Jones, A. Ridgway, D. Harris, and I. Sutherland. A counterflow pipeline experiment. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[4] D. S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *Proceedings of ICCAD' 92, IEEE International Conference on Computer Aided Design*, pages 631–634, 1992.

[5] A. J. Martin, M. Nystroöm, and C. G. Wong. Three generations of asynchronous microprocessors. *To appear in IEEE Design & Test of Computers, special issu on Clockless VLSI Design*, 2003.

[6] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[7] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura. TITAC: Design of a quasidelay-sensitive microprocessor. *IEEE Design and Test of Computers*, 11:50–63, 1994.

[8] S. M. Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers*. PhD thesis, Stanford University, December 1995.

[9] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kil, C. Dike, M. Roncken, and B. Agapiev. RAPPID: An asynchronous instruction-length decoder. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[10] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, 1969.