Fast Hazard Detection in Combinational Circuits

Cheoljoo Jeong Steven M. Nowick*
Department of Computer Science, Columbia University
New York, NY, 10027, USA
{cjeong,nowick}@cs.columbia.edu

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, two new algorithms are proposed to determine if a combinational circuit is hazard-free without exploring all its gates, thus providing more efficient hazard detection. Experimental results indicate that the best new algorithm on average visits only 20.7% of the original gates, with an average runtime speedup of 1.69 and best speedup of 2.27 (for the largest example).

Categories and Subject Descriptors: B.6.1 [Design Styles]: Combinational logic
General Terms: Algorithms, Design, Verification.
Keywords: Logic Design, Asynchronous circuits, Hazards.

1. 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 [6].

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 [9]. This work was further extended by several other researchers: a multilevel hazard analysis technique was proposed by Beister [1] and hazard simulation techniques were proposed based on multi-valued algebra, including 3-valued algebra [3, 10] and 5-valued algebra [5]. 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 two 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 20.7% 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.

An important motivation of this work is that hazard detection is critical to a recent approach for asynchronous synthesis: using *synchronous* (i.e. potentially hazardous) CAD tools (e.g. [2]). Traditional approaches to asynchronous multilevel combinational logic synthesis are highly restrictive since only *hazard-preserving transformations* are allowed [4,9]. However, depending on the actual set of input transitions, more powerful *non-hazard-preserving transformations* may lead to better multilevel implementations without introducing any hazards. Furthermore, currently there is a lack of widely available asynchronous (i.e. hazard-free) technology mapping tools. In sum, an asynchronous CAD flow can use existing synchronous multilevel synthesis and technology mapping tools to produce possibly better hazard-free multilevel logic. Hazard detection is then critical to verify the correctness of the resulting circuit.

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

The hazard detection problem can be formulated as follows.

Hazard Detection Problem *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's 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* ($0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$, $1 \rightarrow 1$) at the primary output gates are known in advance. The goal of the 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.

The remainder of this paper is organized as follows. Section 2 presents some background material and section 3 presents examples to give a motivation of the ideas used in the algorithms. Section 4 presents a theorem and a corollary that are used in the algorithms. Our proposed algorithms are described in detail in Section 5 and experimental results are presented in Section 6. Finally, Section 7 presents conclusions and future work.

2. BACKGROUND

Combinational Circuits and Hazards. Let $\mathsf{B} = \{0,1\}$ and $Y = \{0,1,*\}$. A *Boolean function* f with n inputs and m outputs is defined as a mapping $f : \mathsf{B}^n \rightarrow Y^m$. 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*, *primary outputs*, and *internal vertices* [7].

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.

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 $f(A) \neq f(B)$. A function f which does not change monotonically during an input

*This work was supported by NSF ITR Award No. NSF-CCR-0086036 and by a grant from the New York State Microelectronics Design Center.

DAC 2004, June 7–11, 2004, San Diego, California, USA.

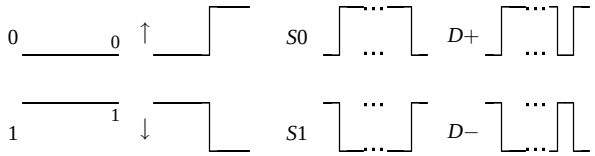

Figure 1: Waveforms represented by Kung values.

transition is said to have a *function hazard* in the transition [1].

If a transition has a function hazard, no implementation of the function is guaranteed to avoid a glitch during the transition. However, 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; such hazards are called *logic hazards* [1].

Kung's Algebra. Kung's algebra [4] is a multi-valued waveform 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 $0 \to 1 \to 0 \to 1$. 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 $0 \to 1 \to 0$. Figure 1 shows the set of Kung values, and the corresponding waveforms represented by these values.

Formally, Kung's algebra is a quintuple $(\mathsf{K}, +, \cdot, 0, 1)$ where $\mathsf{K} = \{0, 1, \uparrow, \downarrow, S0, S1, D+, D-\}$ and two operations $+$ and $\cdot$ are defined using a partial order $\leq$ on 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 that represents a possible waveform generated by "Boolean $+$" over any waveforms of a and b. $a \cdot b$ can be defined similarly. 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.

3. 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.

Example 1: Forward algorithm for static transitions. The "forward" algorithm reduces a circuit through constant propagation first, which is a common technique for simplifying the circuit using constant inputs [7]. In particular, the Kung input values 0 and 1 are propagated to simplify the circuit.

For static transitions (i.e. $0 \to 0$, $1 \to 1$), *only one step is required:* a single pass of constant propagation of static Kung values (0, 1). If the circuit is reduced to a single wire, then a theorem in Section 4 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.

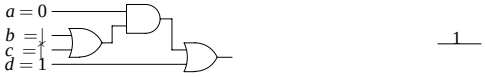

Figure 2: Forward algorithm for static transitions.

Figure 2 shows a circuit marked with primary input values specified by an 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.

Example 2: Forward algorithm for dynamic transitions. When a circuit has a dynamic output transition for a given input transition, 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.

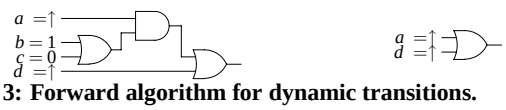

Figure 3: Forward algorithm for dynamic transitions.

Figure 3 shows a combinational circuit marked with input values specified by a dynamic input transition $(A, B) = ((0,1,0,0), (1,1,0,1))$. First, 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 5.2 will be applied to the reduced circuit.)

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 Kung values at each gate are computed during this forward flow.

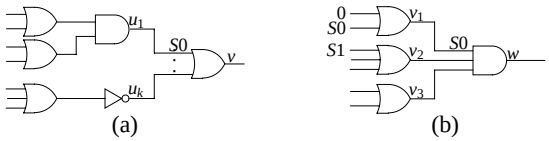

Figure 4: CORE algorithm

During the backward flow of the CORE algorithm, constraints are propagated to the primary inputs which are used to accelerate hazard detection. As an example, suppose that the circuit shown in Figure 4(a) makes a $0 \to 1$ dynamic output transition at node v for a given input transition. Then, when visiting node v, a *constraint* 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 at node v: $\uparrow, \downarrow$, and 0. Note that this partial value set is *disjoint* from the original HF-constraint at node v, $\{\uparrow\}$. 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 actual partial value and the hazard-free constraint are contradictory.

4. THEOREMS ON HAZARD DETECTION

In this section, a theorem and a corollary that are used to simplify hazard detection are presented based on constant propagation. The theorem will be used in the forward algorithm and precomputation.

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

One 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. Therefore, if any glitch occurs in G', it cannot be suppressed and thus G' must be hazardous.

Corollary 1 *Let a combinational circuit G and an input transition t be given. If G is reduced into G' through 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.*

5. HAZARD DETECTION ALGORITHMS

In this section two hazard detection algorithms are presented: the forward and the CORE algorithms. The forward algorithm is 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.

5.1 Forward algorithm

The "forward" algorithm propagates Kung values in a single pass in a topological order of vertices from circuit inputs to outputs. It starts by reducing the circuit through constant propagation. If the

circuit output makes a static transition for the given input transition, the circuit output value can be immediately 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 output makes a dynamic transition for the given input transition, a second step is required since 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 any hazardous Kung value is encountered in the reduced circuit. Therefore, a special *simplified* detection algorithm can be used here, which terminates as soon as any hazardous (non-static) Kung value is produced.

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

Figure 5: Outline of the forward algorithm.

Figure 5 shows an outline of the forward algorithm, where $out(v)$ is the set of output gates of v and $val(v)$ is the Kung value of v. The FORWARD function takes a circuit and an input transition and determines the hazard behavior of the circuit. The 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 4 selects a primary input and performs a depth-first forward constant propagation until either the circuit is reduced to a wire (in line 11) 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 13) and the process continues.

Once the FORWARDPROPAGATE is complete, if the output transition is static, by Theorem 1, the algorithm is done and answers if the circuit is hazard-free. However, if the output transition is dynamic, the reduced circuit must still be simulated using HAZARDDETECT which can be special simplified version of the CORE algorithm or the straightforward Kung simulation algorithm.

Biasing the Direction of Forward Flow. In the forward algorithm, the search direction is biased such that, at each node encountered in the traversal, paths with shorter distance to the circuit output are preferred, as the path is grown from left to right. As a tie-breaker, a path starting with a node in the immediate fanout which is contained in the largest block is preferred, where a block is a sub-circuit which is fanout-free except at its boundary.

5.2 CORE: A recursive algorithm

The "CORE" algorithm is a recursive algorithm for 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 are used to simplify Kung evaluation in the subsequent *forward* flow of the algorithm.

Figure 6 shows an outline of the CORE algorithm. The CORE function takes a gate and constraints as its arguments, and returns the Kung output value of the gate after recursive computation. When invoked with a gate v as its argument, it selects a 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 the recursive search can continue to explore v's other input fanin cones. To determine the hazard behavior of a circuit's output, the function is invoked with the primary output gate as its argument.

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

Figure 6: Outline of the CORE algorithm.

Partial Value Computation. Gate evaluation 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. The process starts with the gate output value as unknown, i.e. the complete Kung set K, and 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 is defined to be the set of possible Kung values at the gate output, at a specific point in time during the recursive evaluation. Partial values will be used in the optimizations below to prune the search space.

Early Termination. While evaluating a gate, if the value of the gate output is already final it is safe to stop the evaluation. That is, when the partial value of a gate is a singleton, gate evaluation can be terminated. This occurs when an input is controlling.

Constraint Propagation. Two kinds of constraints are propagated during the backward flow of the algorithm: HF-constraints and RF-constraints. An *HF-constraint* is a global constraint that must be satisfied at each gate to make the circuit output free of hazards. An *RF-constraint* is a local constraint that must be satisfied at each gate to make its output "useful" in further 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. When an RF-constraint is not satisfied at one of the inputs, v, to a gate w, v's evaluation can immediately be aborted and all unexplored input cones of v can be skipped. In this case, v's evaluation cannot help *refine* the Kung value of w.

More precisely, an HF-constraint passed to a gate v, $\mathrm{HF}(v)$, is a set of Kung values that v can accept at its input to make the circuit's primary output hazard-free (see Example 3 of Section 3). Since output transition types of primary outputs are assumed to be known in advance, HF-constraints for primary output gates can always be generated. Furthermore, such HF-constraints can be propagated down towards the primary inputs as follows. Given two gates v and w, where v is an input to gate w, the HF-constraint passed from w to v, $\mathrm{HF}(v)$, 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 mapping function (for which lookup tables are precomputed); and $\mathrm{HF}(v)$ is computed through this mapping over all Kung values k in $\mathrm{HF}(w)$.

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. Figure 4(b) shows an example. Assume that the control of the algorithm has just reached the AND gate w. When input gate v_1 is visited first and it produces a Kung value $S0$, the partial value of w is refined to $\{0, S0\}$, which are now the only possible values that can be produced on w's output. Then, the second input gate v_2 is visited with an RF-constraint, $\{0\}$. Consider the point when v_2 is visited and its first input has just produced $S1$, and

the partial value of v_2 is refined to $\{S1,1\}$. Since $\{S1,1\} \cap \{0\} = \emptyset$, v_2 cannot produce a value that can further refine v's partial value. Thus, exploration of the *rest of the inputs of* v_2 can be skipped.

Biasing the Search Direction. Two heuristics are used to choose the search direction: input gates with smaller cone size are preferred, and, when there is a tie, an input gate with larger fanin is preferred.

5.3 Preprocessing and Precomputation

A preprocessing step is used in both algorithms to transform the initial circuit into one with fewer gates; the simplified circuit is valid for all input transitions. Here, only hazard-preserving 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.

Precomputation is another technique for reducing the size of the circuit. Unlike preprocessing where the circuit reduction is valid over all input transitions, precomputation reduces the size of the circuit for subsets of input transitions. A set of input transitions can be partitioned into partition blocks such that input transitions in the same partition block have *similar* input assignments. For example, for a circuit of 3 input variables, x_1, x_2, x_3, assume that three input transitions in a partition block define the following Kung input assignments: $(0,1,\uparrow),(0,1,\downarrow)$, and $(0,1,0)$. In this case, a reduced circuit can be precomputed by propagating two common constants $x_1 = 0$ and $x_2 = 1$, which are shared by the three transitions. The hazard detection algorithm can be then executed on the same reduced circuit for each input transition in this partition block.

6. EXPERIMENTAL RESULTS

The two algorithms proposed in Section 5 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 circuits in BLIF format and input transition sets in BTR-ANS format as their input. The circuits used for experiments were synthesized from large Burst-mode asynchronous controller specifications [8] using the Minimalist and the SIS toolsets. In particular, for single-output implementations, state minimization (`min-states` in Minimalist) and state assignments (`assign-states -s -O -P` in Minimalist) were performed by asynchronous tools while two-level (`espresso -Dso`) and multilevel synthesis (`script-rugged` in SIS) as well as technology mapping (`map` in SIS) were performed using non-hazard-free synchronous tools. Multi-output implementations were synthesized using the following steps: `min-states`, `assign-states -P`, `espresso`, `script-rugged`, and `map`.

name	# inputs/# outputs∗/ # gates/# transitions	Forward algorithm	CORE algorithm	# hazards (% hazards)
p1	32/33/295/84	20.7	37.2	18 (0.64%)
p2	18/22/132/56	18.9	46.0	10 (0.81%)
m3	34/25/337/104	17.9	26.1	21 (0.80%)
pscsi	34/29/784/124	24.9	31.9	112 (3.11%)
iccd-scsi	31/27/851/186	21.3	24.3	136 (2.70%)
average % of gates visited		20.7	33.1	(1.61%)

(a) single-output implemented circuits

name	# inputs/# outputs∗/ # gates/# transitions	Forward algorithm	CORE algorithm	# hazards (% hazards)
p1	17/18/193/84	26.2	43.0	38 (2.51%)
p2	12/16/75/56	21.5	47.2	23 (2.56%)
m3	123/14/156/104	20.8	29.0	37 (2.54%)
pscsi	15/10/290/124	23.3	37.6	165 (13.31%)
iccd-scsi	13/9/276/186	23.2	33.5	169 (10.00%)
average % of gates visited		23.0	38.0	(6.18%)

(b) multi-output implemented circuits

∗ # inputs and outputs count also the state variables after state assignments.

Figure 7: Experimental results: gates visited and # hazards.

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 constant propagation with the common input values. Then, the corresponding hazard detection algorithm is applied to the reduced circuit for each of the three input transition in the partition block.

Figure 7 shows a table with the percentage of gates visited by the proposed algorithms, and the "% hazards" indicates the hazards introduced by synchronous CAD tools. If an output is hazardous for k distinct input transitions, it contributes k to the total number of hazards. Hence, "% hazards" counts total hazards as a percentage of $t \times m$, where t is the number of input transitions and m is the total number of primary outputs. On average, approximately 2% of single-output and 6% of multi-output transitions were hazardous. For gate visit counts, on average, the forward algorithm visited 20.7% (single-output) and 23.0% (multi-output) of the gates.

name	# gates	Baseline (time in sec)	Forward algorithm	CORE algorithm
p2	132	1 (0.063)	0.882	0.794
p1	295	1 (0.078)	0.712	0.613
m3	337	1 (0.259)	0.629	0.555
pscsi	784	1 (0.427)	0.589	0.557
iccd-scsi	851	1 (0.628)	0.483	0.440
average runtime ratio		1	0.658	0.591
average speedup		1	1.520	1.690

Figure 8: Experimental results: runtime.

Figure 8 shows the resulting runtime ratios. 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. Actual runtimes are listed for the baseline while runtime ratios are listed for the two algorithms.

Of the two 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.

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 reason that the performance was not as good in small circuits can be attributed to the fact that the actual time for circuit exploration was not a dominant portion of the runtime in small circuits; rather the fixed overheads (indicated above) tended to dominate.

Additional experiments were also performed on MCNC examples (*alu4*, *k2*, *pair*, and *des*), which are not asynchronous control circuits, with 10000 randomly generated input transitions. The forward algorithm visited 42.5% of the gates on average with an average speedup of 1.42. Respective runtimes, in seconds, of the straightforward Kung simulations were 19.2, 25.8 58.7, and 95.1. Even if these results on the synchronous benchmarks showed less improvement than the results on the asynchronous controller examples of Figure 7, the absolute runtime reductions were significant (up to tens of seconds) since the runtimes were far larger than those of Minimalist examples.

7. CONCLUSIONS

This paper presented two new algorithms for detecting hazards in combinational circuits. These algorithms detect hazards visiting only a small portion of the circuit and run faster than the straightforward Kung simulation algorithm. 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 hazards. Also, there are recent asynchronous CAD flows [2] which use synchronous technology mapping and Kung-style hazard detection. We plan to incorporate the proposed optimized hazard detection tool into this flow.

8. REFERENCES

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

[2] T. Chelcea and S. M. Nowick. Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In *Proc. DAC'02*, 2002.

[3] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Develop.*, 9:90–99, 1965.

[4] D. S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *Proc. ICCAD'92*, pages 631–634, 1992.

[5] D. W. Lewis. Hazard detection by a quinary simulation of logic devices with bounded propagation delays. In *Proceedings of DAC'72*, pages 157–164, 1972.

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

[7] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill.

[8] S. M. Nowick. Automatic synthesis of burst-mode asynchronous controllers. Technical Report CSL-TR-95-686, Standford University, December 1995.

[9] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley, 1969.

[10] M. Yoeli and S. Rinon. Application of ternary algebra to the study of static hazards. *J. ACM*, 11(1):84–97, January 1964.