SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits

Feng Shi
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Yiorgos Makris
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Abstract

We present SPIN-SIM, a logic and fault simulator for Speed-Independent Circuits, that extends the classical Eichelberger's method and overcomes its limitations. In order to improve simulation accuracy, SPIN-SIM adopts a 13-valued algebra, maintains the relative order of causal signal transitions, and unfolds time frames judiciously. In addition, complex gates are handled through replacement by pseudo-gate equivalents with regards to functionality, timing and faulty behavior. Experimental results show that SPIN-SIM incurs a negligible increase in computational time over Eichelberger's method, yet is much more accurate and achieves a significant improvement in fault coverage.

1 Introduction

Asynchronous circuits promise several advantages over their synchronous counterparts, including the potential for higher performance, lower power consumption and design reusability. At the same time, they avoid a key emerging challenge of traditional synchronous design, namely high-frequency clock distribution. As chip complexity increases, clock skew effects are significantly amplified, making the problem intractable. Consequently, interest in asynchronous circuits has been resurrected and several commercial products have already been designed.

For asynchronous circuits to take off, however, CAD solutions for assisting their design and test are urgently required. Among these issues, we focus on the problem of testing asynchronous circuits. Due to the lack of a global clock, asynchronous circuits operate independently and are sensitive to race conditions and hazards. Moreover, controlling and observing internal nodes in asynchronous circuits, even through the use of DFT, is not straightforward. To compound the problem, different assumptions on the underlying timing models have lead to a number of distinct classes of asynchronous circuits, each of which may require its own fault simulation and test generation methods.

A number of efforts have been recently devoted to fault modeling, fault simulation and test generation for asynchronous circuits. In order to exploit existing infrastructure, the problem of testing asynchronous circuits using available commercial testers for synchronous circuits has been studied by several researchers [1, 2, 3]. Since many asynchronous circuits operate in fundamental mode, test vectors can only be applied after the circuit has stabilized and fault effects have to be observed when the circuit is in a stable state.

Hazard analysis is critical in simulation and test generation of asynchronous circuits. Eichelberger [4] first used ternary logic simulation for detecting hazards in both combinational and sequential logic. He also proposed a method for simulation of asynchronous sequential circuits. Subsequently, multi-valued logic has been used in different contexts. For instance, Chakraborty et al. [5] used the 13-valued algebra for delay fault test generation. Fsimac, a gate-level fault simulator for stuck-at and gate-delay faults in extended burst mode machines was developed in [6], using min-max timing analysis [7] and 13-valued logic. Fsimac assumes fundamental mode of operation and simulates both the good circuit and the bad circuit in a time-unfolding manner until the primary outputs and state signals stabilize.

In this paper we propose SPIN-SIM, a logic and fault simulation algorithm for Speed-Independent circuits, which extends Eichelberger's method. We first briefly introduce asychronous circuits in section 2, then demonstrate the challenges in fault simulation of Speed-Independent circuits in section 3, and we describe our algorithm in sections 4 and 5. Experimental results are provided in section 6.

2 Classes of Asynchronous Circuits

Asynchronous circuits are classified into several categories based on their timing assumptions. *Delay-Insensitive* circuits [8] operate correctly under arbitrary gate and wire delays, hence are the most robust. Unfortunately, the class of such circuits built out of simple gates is rather limited. *Quasi-Delay-Insensitive* circuits are delay-insensitive except that "isochronic forks" are required to build practical circuits using simple gates and operators. An isochronic fork is a forked wire where all branches have exactly the same delay. *Timed* circuits [9] operate correctly under specific internal and/or environmental timing assumptions such

0-7803-8580-2/04 $20.00 Copyright 2004 IEEE

as bounded delays. *Speed-Independent* circuits [10] tolerate arbitrary gate delays, but assume negligible wire delays.

Alternatively, asynchronous circuits are divided into two main categories according to their design style, namely *Huffman* and *Muller* circuits. Huffman circuits [11] are designed using a traditional asynchronous state machine approach. The state is stored in combinational feedback loops and, thus, may need delay elements along the feedback path to prevent state changes from occurring too rapidly. Huffman circuits are typically designed under the bounded gate and wire delay model. In this model, circuits are guaranteed to work regardless of gate and wire delays, as long as a bound on these delays is known. In order to design correct Huffman circuits, it is also necessary to set constraints on the behavior of the environment, namely when inputs are allowed to change. Moreover, correctness of Huffman circuits relies on the assumption of "fundamental operation mode", which means that outputs and state variables stabilize before either new inputs or feed-back state variables arrive at the inputs. Violation of this assumption may result in a sequential hazard.

Muller circuits [11] are designed mainly based on state transition graphs (or Petri Nets) as the specification form. Under the unbounded gate delay model, circuits are guaranteed to work regardless of gate delays, assuming that wire delays are negligible. Muller circuit design requires explicit knowledge of the behavior protocol allowed by the environment. However, no restrictions are imposed on the order or the speed that inputs, outputs, and state signals change, except that they must behave according to the protocol. Muller circuits correspond to Speed-Independent circuits, and although the two terms are used interchangeably in the literature, we will only use the latter in the rest of the paper, in order to avoid confusion.

3 Simulating Speed-Independent Circuits

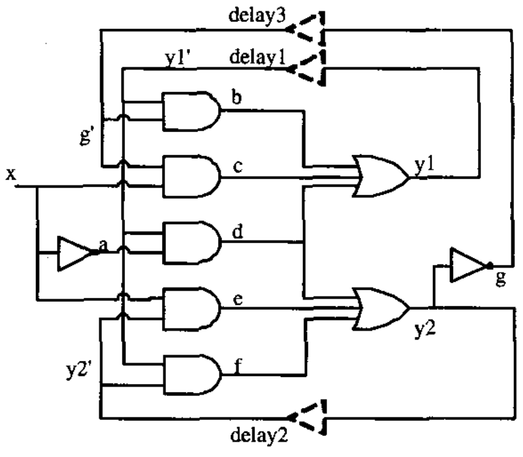
Simulation of asynchronous circuits is more complex than that of their synchronous counterparts, since it needs to deal with hazards/races and oscillations. Moreover, due to their particular timing model, simulation of Speed-Independent circuits presents an additional set of **challenges**, some of which are discussed in this section.

3.1 Dealing with Hazards

Similar to simulators for other types of asynchronous circuits, such as Fsimac [6] which handles Huffman circuits, a simulator for Speed-Independent circuits needs to detect hazard conditions. However, the particular timing model for Speed-Independent circuits requires a different logic simulation method than that for Huffman circuits, especially when handling feedback signals. As illustrated in Figure 1,

y1'y2' \ y1y2 x	0	1
00	00	10
01	00	01
11	11	01
10	11	10

(a)



(b)

Figure 1. Essential Hazard in Asynchronous Circuit

it is possible that *essential* hazards exist in an asynchronous sequential circuit. Figure 1 (a) shows the flow table of an asynchronous state machine, and (b) demonstrates its gate-level implementation. Essential hazards arise when some arrangement of circuit delays allows a state change to complete before the input change is fully processed [12]. For example, suppose that the circuit is initially in state $y_1y_2 = 00$ with input $x = 0$. If input x rises, the circuit will transition to state $y_1y_2 = 10$ according to its flow table. But if the inverter a is slow in comparison to other gates or delay elements on the feedback paths, as shown in Figure 2, c will rise, followed by $y1$, while a remains high. The rise of y_1 through the feedback path triggers the rise of b and d, since a is still high. Then $y2$ rises and g falls, which causes b and c to fall. The rise of $y2$ also causes e to rise. If now the inverter a finishes evaluation and a falls, then d will fall, causing y_1 to fall. But y_2 remains high since now e is high. So the final state of the circuit, in this arrangement of gate delays, is $y_1y_2 = 01$. Therefore, an essential hazard exists.

Such hazards are often avoided in Huffman circuits by inserting enough delays in the feedback lines to ensure that logic signals stabilize after the input transitions and before further transitions occur through the internal state variables.

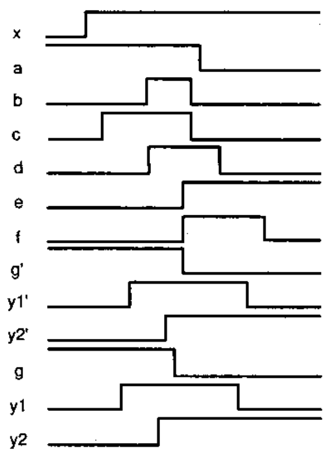


Figure 2. Waveform for the Hazard Condition

For example, if enough delay is inserted in the feedback paths in the circuit of Figure 1 (b), after x rises, c and y_1 rise and a falls, so in the end of the first time frame the state is $y_1y_2 = 10$. Then the circuit stabilizes in this state, which is consistent with its specification. Simulators for Huffman circuits, such as Fsimac [6], assume this "fundamental operation mode" and simulate the circuit in a simple time frame expansion manner. Hence they are not supposed to detect essential hazards. Although valid for simulation of Huffman circuits, this may lead to incorrect results when simulating Speed-Independent circuits, in which no delay elements are inserted in the feedback paths and, hence, essential hazards may exist. Therefore, simulation of Speed-Independent circuits requires a separate method.

In [4], Eichelberger developed an algorithm for detecting hazards in a sequential circuit, which can be adapted and used for simulation of Speed-Independent circuits. His method used $\frac{1}{2}$ to denote an unknown state, which we typically denote by X. Suppose each feedback line is cut, with one end denoted as a pseudo primary input (PPI) and the other as a pseudo primary output (PPO). The algorithm consists of two procedures: in Procedure A, all changing state signals are determined by setting changing primary inputs (PIs) to X and all other PIs and PPIs as originally specified, and then evaluating the PPOs. If there are any PPOs equal to X, the corresponding PPIs are changed to X and the process is repeated until no additional changes occur in PPOs. Then, Procedure B takes over to determine which value each state signal stabilizes to. With the changing PIs equal to their new values and all other PIs and PPIs equal to their values at the end of Procedure A, PPOs and primary outputs (POs) are evaluated. If one or more PPOs change from X to 1 or 0, the corresponding PPI is changed and the process is repeated until no more changes occur in PPOs.

However, Eichelberger's method is too conservative in many cases. Figure 3 gives such an example. In this simple circuit, there are two feedback paths; one is denoted as e while the other is inside the c-element. In Eichelberger's method, these feedback paths are cut into PPIs and PPOs. Let's denote the PPIs as e' and c', and the PPOs as e and c respectively. Table 1 lists the signal value on each node in every iteration of the method. Assume that the circuit state is $e = 0$ and $c = 0$, input b is high, and the stimulus to input a is a falling transition. In the first iteration of Procedure A, the changing input a is set to X, and then the circuit is evaluated. If there is an unknown value on any of the PPOs, such as e, the corresponding PPI, e', is set to X during the next iteration. Procedure A ends when no more Xs appear on the PPOs. Then Procedure B starts, input a is set to its new value of 1 in the fourth iteration, and the circuit is evaluated again. Procedure B ends after the fourth iteration since no X on the PPOs is resolved. Therefore, the final result indicates that the circuit falls into an undetermined state. This, however should not be the case. Initially $b = 1$, $c = 0$, $d = 1$, $a = 1$, $f = 0$ and $e = 0$. After a falls, f and e rise, so c also rises. Then d falls and so does e. But c doesn't change since b remains at 1. In the end, the output of the c-element becomes 1, which is fully determined. This conservativeness is a key problem when using Eichelberger's method on Speed-Independent circuits.

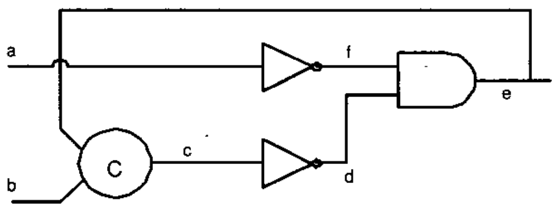


Figure 3. Example Circuit for Demonstrating the Conservativeness of Eichelberger's Method

Node	1st iter	2nd iter	3rd iter	4th iter
a	X	X	X	1
b	1	1	1	1
c'	0	0	X	X
c	0	X	X	X
d	1	X	X	X
e'	0	X	X	X
e	X	X	X	X

Table 1. Simulation of the Circuit of Figure 3

3.2 Preserving Relative Transition Order

Multi-valued algebras have been widely used in tasks that require hazard detection, such as simulation of asynchronous circuits and test generation for delay path faults [4, 5, 7, 6]. The 13-valued algebra has been particularly explored, since it can accurately describe signal transitions. Moreover, it is compact and it avoids unnecessary event

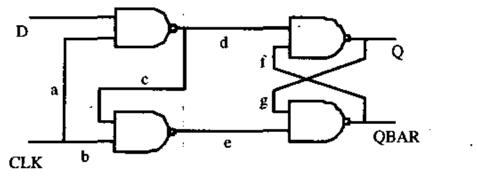


Figure 4. Gate-Level Schematic of a D-Latch

proliferations by abstracting the details of multi-transition waveforms. Another reason for using 13-valued logic is that it facilitates the expression of functions of some complicated gates, such as C-elements, latches and complex domino gates, which are widely used in asynchronous circuits. The 13-valued waveforms are listed below, adopting the interpretation and notation of [7]:

Constant: $< 1,1,1 >, < 0,0,0 >$

Transition: $< 0,\uparrow,1 >, < 1,\downarrow,0 >$

Hazard: $< 0,X,0 >, < 0,X,1 >, < 1,X,0 >, < 1,X,1 >$

Stabilizing: $< X,X,0 >, < X,X,1 >$

Destabilizing: $< 0,X,X >, < 1,X,X >$

Undefined: $< X,X,X >$

The extension of gate functions from the 3-valued logic to the 13-valued logic is not difficult and is detailed in the above references.

However, 13-valued algebra cannot express the relative order of signal transitions, which, in some cases, may lead to false indications of hazard conditions. Figure 4 gives such an example. Suppose that the initial values of the PIs of the D-Latch are $CLK = 1$ and $D = 1$, the values of the internal nodes are $abcdefg = 1100101$, and the values of the POs are $Q = 1$ and $QBAR = 0$. Also suppose that CLK falls to 0 and D doesn't change, or in 13-valued logic, $CLK =< 1,\downarrow,0 >$ and $D =< 1,1,1 >$. Therefore, $b =< 1,\downarrow,0 >$, $c =< 0,\uparrow,1 >$ and $e =< 1,X,1 >$ by gate evaluation, and a glitch on e is reported. However, the circuit assumes that a and b fall simultaneously because they are branches of an isochronic fork. Since the gate delay for c is positive, c rises after a falls, hence after b falls, so there is no glitch on e. In fact, $e =< 1,1,1 >$. In this case, simulation based on 13-valued algebra leads to a false indication of hazard. In order to achieve higher accuracy, the *relative order of signal transitions* needs to be considered during gate evaluation.

3.3 Handling Complex Gates

Another difficulty in simulating asynchronous circuits stems from complex gates. Complex gates are commonly used to extend the limited class of Delay-Insensitive circuits that can be built out of simple gates and operators. By extension, complex gates such as C-elements and complex domino gates are also frequently used in the design of Speed-Independent circuits. While it is possible to treat them as simple gates, as in [6], the number of types of complex gates is not small. Hence, allocating a dedicated truth table for each of them is a considerable memory cost. In addition, the number of inputs of a complex gate is often high, and since the number of entries in the truth table is exponential to the number of inputs, this memory cost is amplified. Moreover, the utilization of multi-valued algebras augments the memory cost significantly by increasing the base number. An alternative solution would be to replace the complex gates with their simple gate implementations for the purpose of simulation. This method incurs no additional memory cost, but unfortunately some Speed-Independent circuits do not longer remain Speed-Independent when complex gates are replaced by their simple gate equivalent, since internal gate delays are now considered. Therefore, an efficient mechanism for handling complex gates is necessary in order to simulate accurately Speed-Independent circuits.

4. Proposed Simulation Method

SPIN-SIM, the proposed simulator, is developed based on Eichelberger's method for hazard detection. However, SPIN-SIM extends this method in several ways to overcome its conservativeness. In addition to adopting a 13-valued algebra to represent signal transitions more accurately, SPIN-SIM preserves the relative order of causal signal transitions, performs judicious time-frame unfolding, and handles complex gates through pseudo-gate replacement.

4.1 Adapted Eichelberger's Method

Since Eichelberger's method is able to detect essential and other hazards in a sequential circuit, it can be adapted into a simulation algorithm for Speed-Independent circuits. To achieve this, the unknown value, X, of the typical 3-valued algebra is replaced by appropriate values in a 13-valued algebra which carry more information about transition waveforms. As an example, Table 2 illustrates the simulation results for the circuit of Figure 1 using the adapted Eichelberger's method. During every simulation iteration, each of the feedback paths y_1, y_2 and g is cut into two ends. The ends that feed gate inputs are denoted by y_1', y_2' and g', while the other ends are denoted by y_1, y_2 and g, respectively. The initial values of the feedback paths are $y_1'y_2'g' = 000$, and the stimulus on input x is $< 0,\uparrow,1 >$. The simulation results for the first, second and third simulation iterations are listed in the second, third and fourth columns respectively. The value on y_1' is replaced by $< 0,X,X >$ in the second iteration since Eichelberger's method sets an unknown value, X, on a PPI if the value on the corresponding

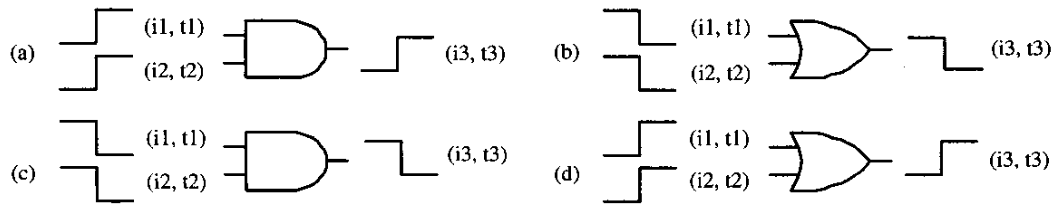


Figure 5. Examples of Maintaining the Time Stamps

Node	1st iter	2nd iter	3rd iter
x	$<0,\uparrow,1>$	$<0,\uparrow,1>$	$<0,\uparrow,1>$
a	$<1,\downarrow,0>$	$<1,\downarrow,0>$	$<1,\downarrow,0>$
b	$<0,0,0>$	$<0,X,X>$	$<0,X,X>$
c	$<0,\uparrow,1>$	$<0,\uparrow,1>$	$<0,X,X>$
d	$<0,0,0>$	$<0,X,X>$	$<0,X,X>$
e	$<0,0,0>$	$<0,0,0>$	$<0,X,X>$
f	$<0,0,0>$	$<0,0,0>$	$<0,X,X>$
g	$<1,1,1>$	$<1,X,X>$	$<1,X,X>$
y_1	$<0,\uparrow,1>$	$<0,X,X>$	$<0,X,X>$
y_2	$<0,0,0>$	$<0,X,X>$	$<0,X,X>$
y_1'	$<0,0,0>$	$<0,X,X>$	$<0,X,X>$
y_2'	$<0,0,0>$	$<0,0,0>$	$<0,X,X>$
g'	$<1,1,1>$	$<1,1,1>$	$<1,X,X>$

Table 2. Adapted Eichelberger's Method

PPO is unstable in the previous iteration. As a result, the values on y_1 and y_2 become destablizing in the following iterations and the simulation terminates because the values on the PPOs are consistent with those on the PPIs. Since the final state of either y_1 or y_2 is destabilizing, the essential hazard described previously is detected by the adapted Eichelberger's method. In contrast, simulation algorithms for Huffman circuits assume sufficient delays on feedback paths and, thus, they use the final states of the PPOs as the values of the PPIs for the next time frame. Hence they are unable to detect this essential hazard.

4.2 Preserving Relative Transition Order

SPIN-SIM maintains the relative order of signal transitions by keeping a simple time stamp during gate evaluation. Since Speed-Independent circuits assume the unbounded gate delay model [10], (i.e. delay elements are attached only to gate outputs and the delay magnitude is positive and finite but unknown), min-max timing analysis for asynchronous circuits based on the bounded delay model, as in [6, 7], is not necessary. As in [6], we mainly assume the *pure delay* type, i.e. waveforms are shifted in time but do not change shape. However, as shown before, the relative order of causal signal transitions is necessary in many cases for correct simulation. In order to keep track of the relative order of causal signal transitions, we keep a time stamp for every transition. The time stamp is simple and only includes a signal group ID and a time. The group ID is used to indicate causal transitions since signal transitions with a causal relation are assigned the same group ID. The relative order of the causal transitions is recorded in the time field, which is incremented with the propagation of the transition. Hence, for two transitions with the same group ID, the one with the smaller time field precedes the other. For instance, if the input to an inverter is $<1,\downarrow,0>$ with a group ID of i and a time field of t, then the output is $<0,\uparrow,1>$ with the same group ID i and a time field of $t+1$. Notice that we only maintain time stamps for transitions, that is, $<0,\uparrow,1>$ and $<1,\downarrow,0>$. The time stamps for other signal values are not kept, since they are typically not necessary. While this might sacrifice some accuracy, it saves significantly in computational time.

Maintaining time stamps for the output of a multi-input gate is more complicated than for an inverter. In the simplest case, the output transitions as a result of changes on input signals that belong to the *same* group, while the remaining inputs are stable. In this case the group ID of the output transition is the same as that of the input transitions and the time field is that of the triggering input transition incremented by 1. It is possible, however, that the output is the result of multiple input transitions that do not all belong to the same group. For instance, Figure 5 illustrates four such cases. In cases (a) and (b), the output transition takes place only after both of the input transitions have taken place. In order to represent this relation, a new group ID $i3$ is assigned to the output and this new group is denoted as a successor of both groups $i1$ and $i2$. Transitions in a predecessor group precede transitions in its successor groups. SPIN-SIM keeps track of this relation by maintaining *group masks*. Each group is assigned a group mask, within which every group is assigned a corresponding bit. A bit is set to 1 if its corresponding group is a predecessor of the current group, otherwise it is set to 0. In the previous example, the two bits corresponding to groups $i1$ and $i2$ are set to 1 in the mask of group $i3$. SPIN-SIM also sets the time field of the output transition to the maximal of those of the input transition plus 1. In cases (c) and (d), the output transition takes place after either of the input transitions takes place. The input transition which takes place first precedes the output transition. However, since it is not known which transition occurs first, this relation is difficult to express. In the inter-

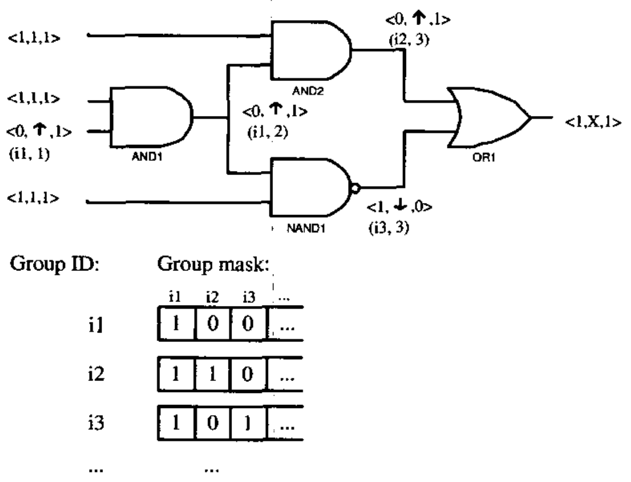


Figure 6. Maintaining Time Stamps

est of simulation speed, we decided to avoid representing this information in SPIN-SIM and set both mask bits to 0, because the degradation of simulation accuracy by doing so is seldom noticeable.

A transition triggered by another transition is typically within the same group as the first one. However, there are exceptions. Figure 6 gives such an example. Since one input to gate $AND1$ is $< 1,1,1 >$ and the other is a transition $< 0,\uparrow,1 >$ with group ID $i1$ and time field 1, the transition propagates to the output of $AND1$ and the waveform is $< 0,\uparrow,1 >$ with the same group ID and an incremented time field of 2. This output fans out to both gates $AND2$ and $NAND1$, so the transition propagates to their outputs since, for both of them, the other input is $< 1,1,1 >$. Thus, the output waveform of gate $AND2$ is $< 0,\uparrow,1 >$ and that of gate $NAND1$ is $< 1,\downarrow,0 >$. However, neither of the two output transitions inherits the group ID of the input transition. This happens because although both transitions are triggered by a common input transition, which definitely precedes them, they exercise different gate delays, hence their relative order is undetermined. If both of them inherited the group ID of the input transition, their relative order would be falsely set. Therefore, SPIN-SIM assigns a new group id $i2$ to the output transition of gate $AND2$ and another new group id $i3$ to that of gate $NAND1$. Both of their time fields are set to 3, which is the time field of the input transition incremented by 1. The signal group $i1$ is set to be the predecessor of both signal groups $i2$ and $i3$ in their group masks respectively, which confirms the fact that the output transition of $AND1$ precedes both of those of gates $AND2$ and $NAND1$. The group masks of these signal groups are also illustrated in figure 6. Notice that the bit corresponding to group $i1$ in the group mask of both $i2$ or $i3$ is set to 1. By convention, in the mask of every group, the bit corresponding to the group itself is always set to 1. When the output transitions of gate $AND2$ and $NAND1$ reach gate $OR1$, they belong to different groups. This implies that there is no relative order between them and, therefore, the output waveform of gate OR is $< 1,X,1 >$.

The gate evaluation process is adapted after considering the relative timing of the signal transitions. If there are fewer than two input transitions on the inputs or if there is no relative order between any two transitions, the gate function is evaluated through the original truth table. However, if there is a relative order between any two input transitions, the output might be different. One method to address this would be to store the outputs under all possible input and relative order combinations into the truth table, and index by both input values and their timing during gate evaluation. Yet this method expands the truth table and needs additional memory space. Instead, SPIN-SIM adopts an alternative method which keeps the original truth table but splits the evaluation process into several phases. For example, in Figure 7 the two input transitions of the AND gate are within the same group and, therefore, have a relative order. Since the time field of the transition on input a is smaller than that on input b, the transition on input a precedes that on input b. In the first phase, the signal on input a is $< 1,\downarrow,0 >$ and the signal on input b takes its pre-transition value, that is, $< 0,0,0 >$. The output is $< 0,0,0 >$ according to the original gate evaluation method. Then in the second phase, the signal on the input a takes its post-transition value, that is, $< 0,0,0 >$, and the signal on input b is correspondingly $< 0,\uparrow,1 >$, and the output is $< 0,0,0 >$. SPIN-SIM computes the final output as the concatenation of the outputs of the two phases which, in this case, is $< 0,0,0 >$. Note that this is the correct result, which is different from the original truth table result of $< 0,X,0 >$.

4.3 Judicious Time Frame Unfolding

SPIN-SIM carefully unfolds time frames. Unlike in Procedure A of Eichelberger's algorithm, which treats transitions and hazards in the same way, if there is a hazard but no transition detected on any PPO after evaluation, the corresponding PPI is set to the destabilizing value, and the process is repeated until no more hazards are detected. If a transition is detected on any PPO, the corresponding PPI is set accordingly, the circuit is re-evaluated and any hazards are handled as described previously. This process is repeated until no more hazards and transitions occur on PPOs, or until the number of iterations exceeds the pre-specified maximum limit. Setting such a limit on the maximum number of iterations is necessary to break the infinite loop that occurs when the circuit oscillates, in which case the first terminating condition will be never satisfied. After this step, a procedure similar to Procedure B of Eichelberger's algorithm takes place to determine the stabilizing values on the POs and the state signals.

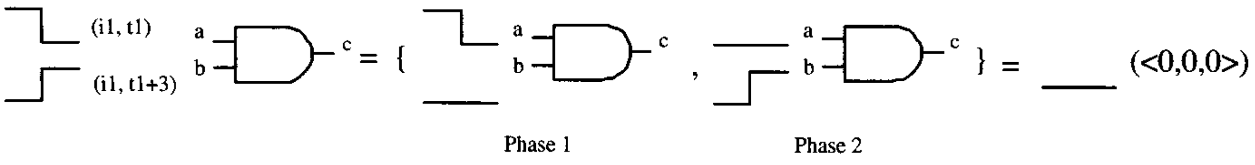


Figure 7. Example of Gate Evaluation with Time Stamps

Node	1st iter	2nd iter	3rd iter	4th iter
a	$<1,\downarrow,0>$ $(0,1)$	$<1,\downarrow,0>$ $(0,1)$	$<1,\downarrow,0>$ $(0,1)$	$<1,\downarrow,0>$ $(0,1)$
b	$<1,1,1>$ NA	$<1,1,1>$ NA	$<1,1,1>$ NA	$<1,1,1>$ NA
c'	$<0,0,0>$ NA	$<0,0,0>$ NA	$<0,\uparrow,1>$ $(0,2004)$	$<0,\uparrow,1>$ $(0,2004)$
c	$<0,0,0>$ NA	$<0,\uparrow,1>$ $(0,1004)$	$<0,\uparrow,1>$ $(0,1004)$	$<0,\uparrow,1>$ $(0,1004)$
d	$<1,1,1>$ NA	$<1,\downarrow,0>$ $(0,1005)$	$<1,\downarrow,0>$ $(0,1005)$	$<1,\downarrow,0>$ $(0,1005)$
e'	$<0,0,0>$ NA	$<0,\uparrow,1>$ $(0,1003)$	$<0,\uparrow,1>$ $(0,1003)$	$<1,\downarrow,0>$ $(0,2006)$ $[0,1003]$
e	$<0,\uparrow,1>$ $(0,3)$	$<1,\downarrow,0>$ $(0,1006)$ $[0,3]$	$<1,\downarrow,0>$ $(0,1006)$ $[0,3]$	$<1,\downarrow,0>$ $(0,1006)$ $[0,3]$
f	$<0,\uparrow,1>$ $(0,2)$	$<0,\uparrow,1>$ $(0,2)$	$<0,\uparrow,1>$ $(0,2)$	$<0,\uparrow,1>$ $(0,2)$

Table 3. SPIN-SIM Results on the Example Circuit

We demonstrate this by simulating again the circuit of Figure 3 using SPIN-SIM. The initial condition is $c = 0$ and $e = 0$. The stimulus is $a =< 1, \downarrow, 0 >$ and $b =< 1, 1, 1 >$. The signal values for each node in the four simulation iterations are listed in the second through fifth column in Table 3, respectively. The time stamps of signal transitions are also listed in parenthesis when applicable. The first number in the parenthesis is the signal group ID and the second number is the time field value. Since SPIN-SIM uses the 13-valued algebra, it is not necessary to set the changing inputs to undefined values in the first simulation iteration in procedure A. The input transition on a is assigned a group ID of 0 and a time stamp of 1. The circuit is then evaluated and the value on PPO e turns out to be a rising transition with a time stamp $(0, 3)$. Unlike Eichelberger's method, in which the corresponding PPI e' is set to the destabilizing value in the next iteration, SPIN-SIM replaces the value on e' with the previous value on e, which is $< 0, \uparrow, 1 >$, since this is a transition and not a hazard. Note that this kind of replacement of a stable value with a transition resembles more a signal value correction than a time frame unfolding, since there is no signal timing information loss. In the second iteration, the time stamp of e' becomes $(0, 1003)$. This, however, does not imply that the transition on e' follows the one on e in the first iteration. SPIN-SIM uses this method to just indicate that the transition has propagated over a feedback path. After evaluation, the result is a rising transition on c and a falling transition on e. Given the inputs $f =< 0, \uparrow, 1 > (0, 2)$ and $d =< 1, \downarrow, 0 > (0, 1005)$, the result on e would be $< 0, X, 0 >$. However, SPIN-SIM identifies from their time stamps that there are two transitions in the result and the former has been already stored in e', so only the latter transition is reported, but the time stamp of the previous one is still kept, as shown in brackets, to indicate the starting point of the current waveform. In the third iteration SPIN-SIM corrects the value on c' to $< 0, \uparrow, 1 > (0, 2004)$ but does not change the value on e' to $< 1, \downarrow, 0 > (0, 2006)$, since by doing so, it would discard the old waveform information on e'. Thus, this is a time frame unfolding. SPIN-SIM always preforms signal corrections before time frame unfolding, in order to make sure that no hazard conditions exist. Since no hazard is detected in the end of the third iteration, the time frame unfolding on e' is performed and the time stamp of the old transition is kept as its new starting point. Since the evaluation results in no additional signal changes, Procedure A terminates.

Procedure B is not executed because the next states on the PPOs coincide with those on the PPIs and the simulation finishes. The final result confirms the previous analysis which indicates that the final value should be $c = 1$. Note that if any hazards are detected on PPOs, the corresponding PPIs are set to destabilizing values, as in Eichelberger's method. The starting points of waveforms are maintained so that evaluation of unfolded signals will produce a correct result with a proper starting point. Time frame unfolding might not terminate in case of oscillations. Hence, a maximal number of allowable simulation iterations is set, so that when the number of iterations exceeds this limit SPIN-SIM assumes that the circuit is in oscillation and sets the switching signals to destabilizing values.

4.4 Handling Complex Gates

Many complex gates are commonly used in Speed-Independent circuits. Instead of storing a truth table for each of them, SPIN-SIM uses simple gates to represent each of these complex gates that have simple gate level equivalent implementations. However, a naive replacement often introduces additional hazards which result in a circuit that is no longer Speed-Independent. The key challenge is to design such gate level equivalent implementations that

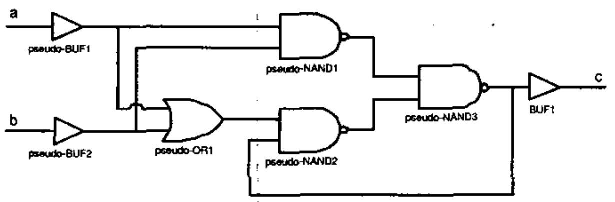


Figure 8. C-element: Pseudo Gate Implementation

mimic the original complex gate in both functionality and timing, employing both pseudo gates and real gates. Figure 8 gives an example of a pseudo-gate implementation of a c-element. All gates in this example are pseudo-gates, except for the output-stage buffer. SPIN-SIM assumes no delay when a signal propagates through a pseudo-gate, hence its time stamp does not increase. Therefore, the only delay is contributed by the buffer on the output, which mimics successfully the timing property of a complex c-element. Pseudo-buffers are also inserted in the inputs to make internal fan-out invisible to the outside of the complex gate. To support SPIN-SIM, we have developed a library of gate and pseudo-gate equivalent implementations for the most commonly used complex gates by hand. In this way, SPIN-SIM simulates complex gates accurately and efficiently, without incurring additional memory cost.

5 Fault Simulation

Fault simulation of Speed-Independent circuits is similar to that of synchronous circuits. The required stuck-at fault list may be either user-defined or generated automatically, including all stuck-at faults on inputs or output of gates, except those inside a complex gate. The fault list may be pruned through equivalent fault collapsing to speed up fault simulation. Each test vector is then simulated on the good circuit and on each bad circuit, where a single stuck-at fault from the fault list is injected. The output values of the faulty circuit are compared to those of the good circuit and if the fault is detected it is dropped from the fault list. This process is repeated until all the test vectors are simulated.

In order to discuss fault collapsing in Speed-Independent circuits, we adopt the definitions of [13]. We refer to fault equivalence/dominance in a single gate as g-equivalence/dominace, in a combinational circuit as c-equivalence/dominace, and in a synchronous sequential circuit as s-equivalence/dominance. The corresponding extensions to asynchronous sequential circuits are defined as follows. A fault y is said to a-dominate another fault x in an asynchronous sequential circuit if and only if every test sequence for x is also a test sequence for y. Two faults, x and y, are said to be a-equivalent in an asynchronous sequential circuit if and only if x a-dominates y and y a-dominates x. It is obvious that the equivalence relationship in a single gate remains valid in a Speed-Independent circuit. Unfortunately, a c-equivalent pair of faults might not be a-equivalent in a Speed-Independent circuit, since the circuit under each fault might have different hazard conditions that lead to different undetermined states. Therefore, a test for one fault in a c-equivalent pair might be invalid for the other. For the same reason, as well as due to self-hiding and delayed reconvergence [13], a c-dominant and c-dominated pair of faults might not be an a-dominant and a-dominated pair. Therefore, SPIN-SIM collapses conservatively, i.e. only g-equivalent faults.

Although efficient parallel fault simulation techniques have been devised for synchronous circuits [14], similar techniques for asynchronous circuit are yet to be developed. The main reason is that 13-valued algebra requires more bits to represent a value, and, more importantly, that the basic gate functions in 13-valued algebra deviate from common bitwise logic operators. Therefore, similar to Fsimac [6], SPIN-SIM performs fault simulation serially.

6 Experimental Results

SPIN-SIM has been developed in C, based on the simulation engine of HOPE [14]. The input circuit netlist is in ISCAS89 format and the stuck-at fault list can be defined through a file or generated automatically by the tool. We experimented with SPIN-SIM on a set of Speed-Independent circuits synthesized by Petrify [15]. Complex gates like Muller C-elements are handled as described in section 4.4. In each benchmark, a reset input is assumed to be connected to every memory element to appropriately initialize the circuit. Experiments were performed on a workstation with dual Xeon 1.7GHz processors and 1 gigabyte of RAM. For each benchmark, we fault simulated ten thousand random test vectors generated through the method described in [16]. A fault is reported detected only if the generated test patterns are guaranteed to detect it assuming any possible combination of gate delays. This, however, does not imply that we are limited to patterns that will never cause undetermined states or oscillations in the circuit. In some cases a circuit state may be undetermined or in oscillation after applying a pattern, but the faulty circuit may still be detected because at least one output will be stable and will have a value different than the expected good circuit response. In order to demonstrate the efficiency of SPIN-SIM, we compare our results to those of Eichelberger's method, which we implemented as in [4], except that we adopted a 13-valued algebra during the simulation. For each method and each circuit we repeated the experiment 20 times and we report the average of the obtained values.

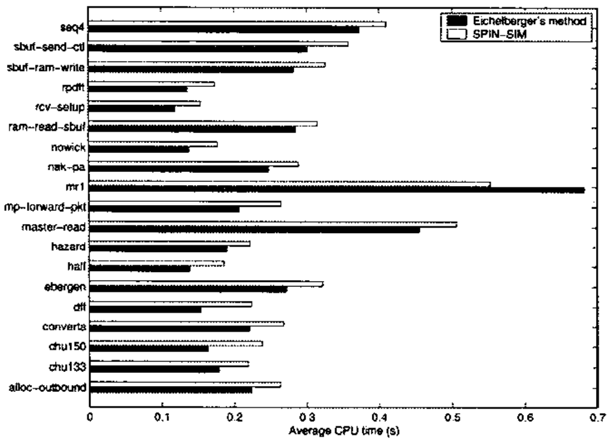


Figure 9. Simulation CPU Time

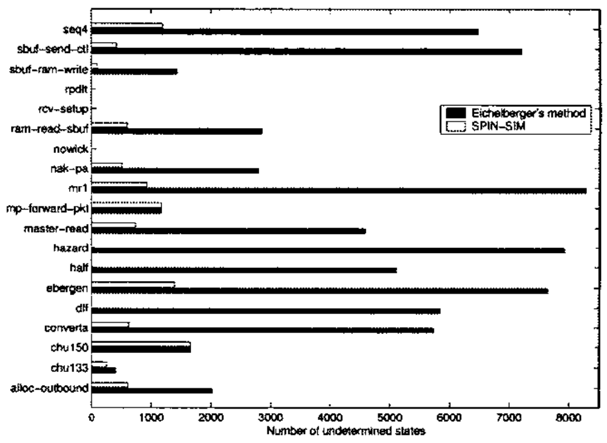


Figure 10. Number of Undetermined States

We first compare in Figure 9 the time that each method spent on logic simulation of the test vectors on the good circuit. For some circuits, such as *chu150* and *dff*, SPIN-SIM spends about 45% more CPU time than Eichelberger's method. This additional computational effort is mainly spent to maintain time stamps during gate evaluation, through which SPIN-SIM provides improved accuracy. For some circuits, such as *master-read*, *ram-read-sbuf*, and *seq4*, SPIN-SIM spends only about 10% more CPU time than Eichelberger's method. Finally, for some circuits such as *mr1*, SPIN-SIM spends less CPU time than Eichelberger's method. This happens mainly because in some cases the conservativeness of Eichelberger's method leads to undetermined states which may cause additional simulation iterations. Overall, SPIN-SIM spends around 21% more CPU time than Eichelberger's method. Yet, the accuracy of simulation is significantly improved. To demonstrate this, Figure 10 shows the number of undetermined states for each circuit during the simulation by each of the two methods. Evidently, the techniques employed in SPIN-SIM result in a simulation where circuit states are resolved much better than in Eichelberger's method, even when the latter employs a 13-valued algebra.

By overcoming the conservativeness of Eichelberger's method, SPIN-SIM also achieves a significant improvement in fault coverage. Table 4 illustrates the fault simulation results of both SPIN-SIM and Eichelberger's method. The number of stuck-at faults in each circuit before and after fault collapsing is listed in the second and third column respectively, along with the fault collapsing rate in the fourth column. Although we only collapsed g-equivalent faults, the fault collapsing rate is still considerable, averaging at 38.7%. The fifth and sixth columns provide the average number of detected faults and the average fault coverage of the randomly generated vectors for each circuit using Eichelberger's simulation method. An average fault coverage of 75.6% is achieved across all circuits. Similarly, the eighth and ninth columns list the average number of detected faults and the average fault coverage of the same randomly generated vectors for each circuit using SPIN-SIM. Overall, SPIN-SIM achieves an average fault coverage of 97.1% across all benchmark circuits, which means an average improvement of 21.5% over Eichelberger's method.

In the seventh and tenth column of Table 4, we also list the average CPU time needed by each method to fault simulate the ten thousand random test vectors on each benchmark circuit. For several circuits, such as *chu150* and *rpdft*, SPIN-SIM spends 14–70% more CPU time, although both methods are equally effective in terms of fault coverage. This is attributed to the higher computational complexity of SPIN-SIM that keeps time-stamps and other information to increase simulation accuracy. However, SPIN-SIM spends significantly less CPU time, even up to tens of times less in some cases, on circuits such as *ebergen* and *mr1*. At the same time, it provides a much better fault coverage, since it alleviates the conservativeness of Eichelberger's method. Interestingly, this improved accuracy is the exact reason for the CPU time savings. Eichelberger's method is unable to detect a considerable number of faults and simulates each of them for every test vector, while SPIN-SIM detects many of them in early stages and drops them from the fault list. Thus, SPIN-SIM performs fewer simulations and saves a lot of CPU time. The normalized CPU time savings for these experiments were 33.9%.

7 Conclusion

Logic simulation of Speed-Independent circuits requires consideration of hazard conditions caused by changes of feedback lines before the circuit stabilizes. In addition, fault

				Eichelberger's method			SPIN-SIM		
Circuit Name	No. of Faults	No. of faults After collapsing	Collapsing Rate (%)	Average No. of Detected faults	Average fault Coverage(%)	CPU time (seconds)	Average No. of Detected faults	Average fault Coverage(%)	CPU time (seconds)
alloc-outbound	70	41	41.4	41	100	0.012	41	100	0.009
chu133	54	32	40.7	31	96.9	0.261	31	96.9	0.365
chu150	56	34	39.3	33	97.1	0.247	33	97.1	0.418
converta	54	37	31.5	21	56.8	1.478	34	91.9	0.656
dff	44	28	36.4	6	21.4	1.313	24	85.7	0.772
ebergen	74	46	37.8	22	47.8	3.250	44	95.7	0.747
half	22	15	31.8	6	40.0	0.563	15	100	0.006
hazard	48	33	31.2	29	87.9	0.562	32	97.0	0.383
master-read	144	86	40.3	56	65.1	5.135	84	97.7	1.119
mp-forward-pkt	60	34	43.3	34	100	0.007	34	100	0.008
mr1	152	93	38.8	10	10.8	31.184	87	93.5	2.320
nak-pa	82	48	41.5	48	100	0.016	48	100	0.008
nowick	56	28	50.0	28	100	0.007	28	100	0.007
ram-read-sbuf	90	55	38.9	55	100	0.063	55	100	0.011
rcv-setup	40	25	37.5	25	100	0.006	25	100	0.007
rpdft	62	34	45.2	34	100	0.006	34	100	0.007
sbuf-ram-write	110	69	37.3	69	100	0.042	69	100	0.029
sbuf-send-ctl	94	59	37.2	35	59.3	2.485	56	94.9	0.962
seq4	96	63	34.4	34	54.0	4.263	60	95.2	1.044
Average			38.7		75.6			97.1	

Table 4. Fault Simulation Results

simulation of Speed-Independent circuits necessitates that faults be detected through signals that are in a stable state. To address these issues, we developed SPIN-SIM, a logic and fault simulation algorithm for stuck-at faults in Speed-Independent circuits by extending Eichelberger's method. In addition to adopting a 13-valued algebra, SPIN-SIM maintains the relative order of causal signal transitions in the circuit, unfolds time frames carefully and handles complex gates through a pseudo-gate replacement technique. Experimental results indicate that SPIN-SIM achieves much higher logic and fault simulation accuracy, yet incurs only a slight increase in computation time, if any at all.

References

[1] S. Banerjee, S. T. Chakradhar, and R. K. Roy, "Synchronous test generation model for asynchronous circuits," in *Proceedings of the 9th International Conference on VLSI Design*, 1996, pp. 178–85.

[2] M. A. Breuer, "The effects of races, delays, and delay faults on test generation," *IEEE Transactions on Computers*, vol. C-23, no. 10, pp. 1078–1092, 1974.

[3] O. Roig, J. Cortadella, M. A. Peiia, and E. Pastor, "Automatic generation of synchronous test patterns for asynchronous circuits," in *Proceedings of the 34th Design Automation Conference*, 1997, pp. 620–625.

[4] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM Journal of Research and Development*, vol. 9, no. 2, pp. 90–99, 1965.

[5] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "Delay fault models and test generation for random logic sequential circuits," in *Proceedings of the 29th Design Automation Conference*, 1992, pp. 165–172.

[6] S. Sur-Kolay, M. Roncken, K Stevens, P. P. Chaudhuri, and R. Roy, "Fsimac: A fault simulator for asynchronous sequential circuits," in *Proceedings of the 9th Asian Test Symposium*, 2000, pp. 114–119.

[7] S. Chakraborty, D. Dill, and K. Yun, "Min-max timing analysis and an application to asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 332–346, 1999.

[8] W. A. Clark, "Macromodular computer systems," in *Proceedings of the Spring Joint Computer Conference, AFIPS*, 1967, vol. 30, pp. 335–336.

[9] C. J. Myers and T. Meng, "Synthesis of timed asynchronous circuits," *IEEE Transactions on VLSI Systems*, vol. 1, no. 2, pp. 106–119, 1993.

[10] D. Muller and W. Bartky, "A theory of asynchronous circuits," *In Annals of Computing Laboratory of Hardward University*, pp. 204–243, 1959.

[11] C. J. Myers, *Asynchronous Circuit Design*, John Wiley and Sons, Inc., New York, 2001.

[12] S. H. Unger, *Asynchronous Sequencial Switching Circuits*, Wiley-Interscience, New York, 1969.

[13] J. E. Chen, C. L. Lee, and W. J. Shen, "Single-fault fault-collapsing analysis in sequential logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 12, pp. 1559–1568, 1991.

[14] H. K. Lee and D. S. Ha, "Hope: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.

[15] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.

[16] F. Shi and Y. Makris, "Fault simulation and random test generation for speed-independent circuits," in *Proceedings of the 2004 Great Lakes Symposium on VLSI*, 2004, pp. 127–130.