Enhancing Simulation Accuracy through Advanced Hazard Detection in Asynchronous Circuits

Feng Shi, *Member*, *IEEE*, and Yiorgos Makris, *Senior Member*, *IEEE*

Abstract—A fast and accurate simulator with elaborate hazard detection capabilities is vital for asynchronous circuits, not only for the purpose of design validation through logic simulation, but even more importantly for the purpose of test validation through fault simulation. Toward this end, we developed SPIN-SIM, a logic and fault simulator built around Eichelberger's classical hazard detection method, yet extended in various ways in order to overcome its limitations. More specifically, in order to improve simulation accuracy and hazard detection, SPIN-SIM 1) employs a 13-valued algebra for which it adapts Eichelberger's method, 2) maintains partial orders of causal signal transitions through relative time stamps, and 3) unfolds time frames judiciously to distinguish between hazards and actual transitions. Experimental results demonstrate that, at the cost of a negligible increase in computational time over Eichelberger's method, if any at all, SPIN-SIM achieves significantly more accurate logic simulation and, by extension, drastically more efficient fault simulation. Furthermore, while the proposed method was developed and is presented for the class of Speed-Independent circuits, it is easily extendible to various other classes of asynchronous circuits.

Index Terms—Logic and fault simulation, asynchronous circuits, testing.

1 Introduction

Asynchronous circuits promise a wide range of advantages, including elimination of clock distribution networks and skew problems, low EMI, improved performance, reduced power consumption, and modularity [1], [2]. Nevertheless, and despite the recent release of several asynchronous products [3], adoption of the asynchronous design style has been rather limited, mainly because of the lack of EDA support. Indeed, asynchronous circuits present their own set of challenges, making the porting of computer-aided design and test methods from the synchronous domain neither straightforward nor always possible. Among the various issues, we focus on the problem of logic and fault simulation. Due to the lack of a global clock, components in asynchronous circuits operate independently and their interaction is sensitive to race conditions and hazards, making the use of existing synchronous circuit simulators insufficient, if at all possible. To compound the problem, varying assumptions on the underlying timing models have led to a number of distinct classes of asynchronous circuits, each of which may require its own simulation methods.

An accurate and efficient gate-level simulation method for asynchronous circuits is a crucial component in their design and test flow. A key attribute of achieving the required accuracy is the ability to detect hazards. One may argue that asynchronous circuits are often specified and implemented through formal methods that guarantee their hazard-free operation, thus reducing the need for a hazard-detecting simulator for the purpose of design validation. However, such guarantees are only provided for a fault-free circuit. In the presence of a manufacturing fault, it is possible that the circuit behavior deviates from its specification in ways that create hazards. As a result, generating and validating test vectors cannot be done without the use of a hazard-detecting simulator. Moreover, test vectors in asynchronous circuits have to be applied after the circuit has stabilized, and fault effects have to be observed when the circuit is in a stable state, conditions that synchronous fault simulators cannot guarantee.

Corroborating these points, it comes as no surprise that test technology for asynchronous circuits is still in its infancy. Indeed, most existing test generation and application methods for asynchronous circuits [4], [5], [6] employ scan-based approaches, which enable the use of synchronous test tools. Yet the area and performance overhead of these approaches is significant, and their effectiveness often limited, since they do not exercise the circuit in its native asynchronous mode. And while attempts to develop native-mode automatic test pattern generation (ATPG) methods [7], [8], [9] for asynchronous circuits have been made, they have been thwarted by the lack of efficient logic and fault simulation capabilities.

In this paper, we present SPIN-SIM, a logic and fault simulation method for asynchronous circuits built around Eichelberger's classical hazard detection method [10], which we extend in order to improve its accuracy. First, in Section 2, we briefly introduce the various classes of asynchronous circuits and we pinpoint the difficulties of dealing with hazards. Then, in Section 3, we discuss the key novelties of the proposed logic simulation method, which we formally present in Section 4. Section 5 explains how the proposed

- *F. Shi is with Skyworks Solutions, Inc., Irvine, CA 92617. E-mail: feng.shi@gmail.com.*
- *Y. Makris is with the Department of Electrical Engineering and Computer Science, Yale University, New Haven, CT 06520-8267. E-mail: yiorgos.makris@yale.edu.*

Manuscript received 29 Aug. 2007; revised 5 June 2008; accepted 25 June 2008; published online 6 Aug. 2008.
Recommended for acceptance by J. Lach.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-08-0439.
Digital Object Identifier no. 10.1109/TC.2008.141.

method handles complex gates, and Section 6 outlines its use for fault simulation. Experimental results demonstrating the efficiency of the proposed logic and fault simulation method are provided in Section 7, and its applicability to other classes of asynchronous circuits is discussed in Section 8.

2 Simulating Asynchronous Circuits

One of the key simulation complications arises from the plethora of different styles of asynchronous circuits, each of which comes with its own set of timing constraints and requirements. After briefly describing the most popular classes, we focus on the particularities of simulating the class of Speed-Independent circuits. While SPIN-SIM was initially developed for this class of circuits, we note that its underlying framework enables its straightforward extension to other classes, as we discuss in Section 8.

2.1 Classes of Asynchronous Circuits

Asynchronous circuits are classified into several categories based on their timing assumptions. *Delay-Insensitive* circuits [11] 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 are assumed to have exactly the same delay. *Timed* circuits [12] operate correctly under specific internal and/or environmental timing assumptions, such as bounded delays. *Speed-Independent* circuits [13] 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 [14] are designed using a traditional asynchronous state machine approach. The state is stored in combinational feedback loops and, thus, may require 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. Correctness of Huffman circuits relies on the assumption of "fundamental operation mode," which requires that outputs and state variables stabilize before either new inputs or changed feedback state variables arrive. Violation of this assumption may result in a sequential hazard and, thus, incorrect outputs.

Muller circuits [14] are designed mainly based on state transition graphs (or Petri Nets) as the specification form. Under the unbounded gate delay model, these 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 speed that inputs, outputs, and state signals change, except that they must comply with this protocol.

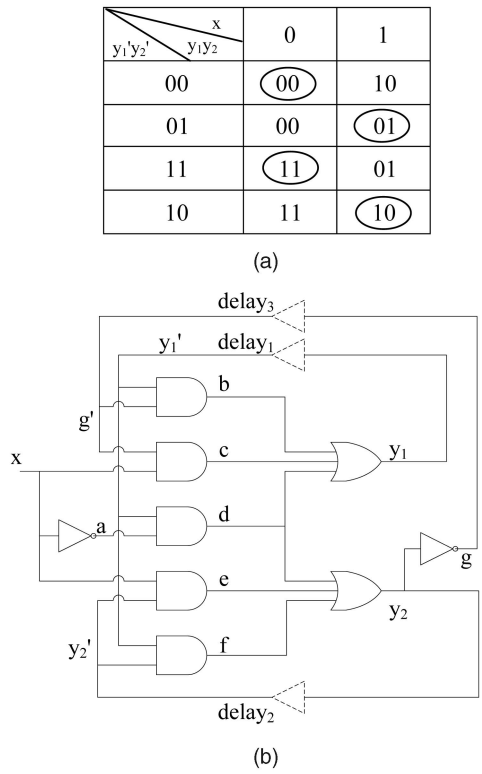


Fig. 1. Asynchronous circuit example.

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 this paper.

2.2 Hazards in Speed-Independent Circuits

Simulation of asynchronous circuits needs to deal with hazards, races, and oscillations, which synchronous circuit simulation is not concerned with. Simulation of Speed-Independent circuits, in particular, presents a set of unique challenges due to their specific timing model. Similar to simulators for other types of asynchronous circuits, such as Fsimac [15] which handles Huffman circuits, a simulator for Speed-Independent circuits needs to detect hazard conditions. However, the particular timing model of Speed-Independent circuits calls for a different logic simulation method than that for Huffman circuits, especially when handling feedback signals.

As illustrated in Figs. 1 and 2, it is possible that *essential hazards* exist in an asynchronous sequential circuit. Fig. 1a shows the flow table of an asynchronous state machine, where the circled entries indicate stable states, and Fig. 1b 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 [16]. 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 through the sequence of signal transitions shown in Fig. 2a. But if the inverter generating a is slow in comparison to other gates or delay elements on the feedback paths, an incorrect result will occur, as shown in Fig. 2b. Specifically, c will rise, followed by y_1, while a remains high.

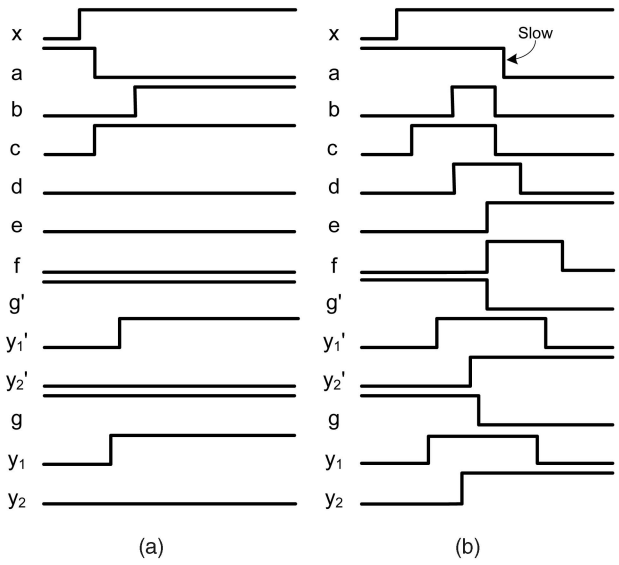


Fig. 2. Essential hazard example in circuit of Fig. 1.

The rise of y_1 through the feedback path triggers the rise of b and d, since a is still high. Then, y_2 rises and g falls, which causes b and c to fall. The rise of y_2 also causes e to rise. If, now, the inverter generating 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 delay in the feedback lines to ensure that logic signals stabilize after the input transitions and before further transitions occur through the internal state variables. For example, if enough delay is inserted in the feedback paths in the circuit of Fig. 1b, 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 [15], assume this "fundamental operation mode" and simulate the circuit in a simple time frame unfolding manner. Hence, they are not required 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 different approach.

One possible solution is to use the traditional algorithm developed by Eichelberger [10], which employs ternary

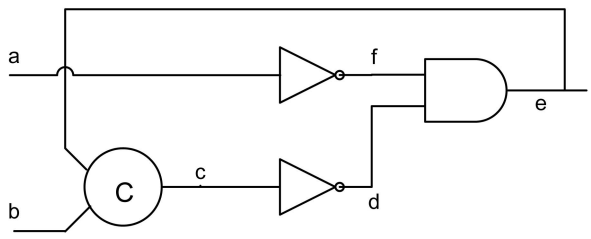


Fig. 3. Conservativeness of Eichelberger's method.

TABLE 1
Simulation of the Circuit of Fig. 3

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

logic simulation for detecting hazards in both combinational and sequential logic, as well as in asynchronous circuits. This method uses $\frac{1}{2}$ to denote an unknown state, which we typically denote by X. Suppose each feedback line is cut, with one end denoted as a pseudoprimary input (PPI) and the other as a pseudoprimary output (PPO). Eichelberger's 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. Fig. 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.[1] In Eichelberger's method, these feedback paths are cut into PPIs and PPOs. Let us 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 new X's appear on the PPOs (third iteration). 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 does not 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.

1. A Muller C-element is a sequential component whose output reflects the value of its inputs when all inputs match and remains in this state until all inputs transition to the opposite state.

TABLE 2
Simulation Using Adapted Eichelberger's Method

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

3 Proposed Simulation Method

SPIN-SIM, the proposed simulator, is developed based on Eichelberger's hazard detection method. However, in order to overcome its conservativeness, SPIN-SIM extends this method in several ways. First, it employs a 13-valued algebra to represent signal transitions more accurately and adapts Eichelberger's method accordingly. Second, it uses time stamps to maintain partial orders of causal signal transitions. Third, it unfolds time frames judiciously, distinguishing between hazards and actual transitions. The details of these enhancements are described in the following three sections.

3.1 Eichelberger's Method with 13-Valued Algebra

Multivalued algebras have been widely used in tasks that require hazard detection, such as simulation of asynchronous circuits and test generation for path delay faults [10], [15], [17], [18], [19], [20], [21]. The 13-valued algebra has been particularly explored, since it can accurately describe signal transitions. Moreover, it is compact and it avoids unnecessary event proliferation by abstracting the details of multi-transition waveforms. Another reason for using 13-valued logic is that it facilitates the expression of functions of complicated gates, such as C-elements, latches, and complex domino gates, which are used in asynchronous circuits. The 13-valued waveforms are listed below, adopting the interpretation and notation in [19], wherein the three symbols of a 3-tuple correspond to the initial, intermediate, and final signal value, respectively:

- **Constant:** $\langle 1,1,1\rangle$, $\langle 0,0,0\rangle$.
- **Transition:** $\langle 0,\uparrow,1\rangle$, $\langle 1,\downarrow,0\rangle$.
- **Hazard:** $\langle 0,X,0\rangle$, $\langle 0,X,1\rangle$, $\langle 1,X,0\rangle$, $\langle 1,X,1\rangle$.
- **Stabilizing:** $\langle X,X,0\rangle$, $\langle X,X,1\rangle$.
- **Destabilizing:** $\langle 0,X,X\rangle$, $\langle 1,X,X\rangle$.
- **Undefined:** $\langle X,X,X\rangle$.

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

Using the 13-valued algebra, we adapt Eichelberger's method which is able to detect essential and other hazards in a sequential circuit, in order to perform simulation of Speed-Independent circuits. To achieve this, the unknown value X of the typical 3-valued algebra is replaced by appropriate values in the 13-valued algebra, which carry more information about transition waveforms. As an example, Table 2 illustrates the simulation results for the circuit of Fig. 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 $\langle 0,\uparrow,1\rangle$. The simulation results for the first, second, and third simulation iterations are listed in the second, third, and fourth column, respectively. The value on y_1' is replaced by $\langle 0,X,X\rangle$ in the second iteration since Eichelberger's method sets an unknown value, X, on a PPI if the value on the corresponding PPO is unstable in the previous iteration. As a result, the values on y_1 and y_2 become destabilizing 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 delay 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.

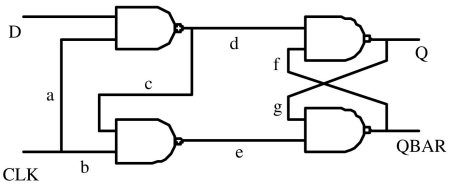


Fig. 4. Gate-level schematic of a D-Latch.

While the use of the 13-valued algebra improves hazard detection, it cannot express the relative order of signal transitions. In some cases, this may lead to false indication of hazard conditions. Fig. 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 does not change, or in 13-valued logic, $CLK = \langle 1,\downarrow,0\rangle$ and $D = \langle 1,1,1\rangle$. Therefore, $b = \langle 1,\downarrow,0\rangle$, $c = \langle 0,\uparrow,1\rangle$, and $e = \langle 1,X,1\rangle$ 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, $e = \langle 1,1,1\rangle$. In this case, simulation based on 13-valued algebra leads to a false indication of a hazard. In order to achieve higher accuracy, the *relative order of signal transitions* needs to be considered during gate evaluation.

3.2 Maintaining Relative Transition Order

Since Speed-Independent circuits assume the unbounded gate delay model [13] (i.e., delay elements are attached only to gate outputs and the delay magnitude is positive and finite but unknown), min-max timing analysis based on the bounded delay model, as in [15] and [19], is not necessary. As in [15], we mainly assume the *pure delay* type, i.e., waveforms are shifted in time but do not change shape.

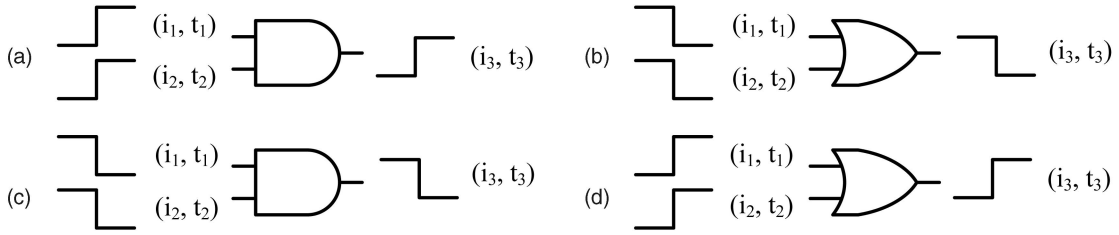
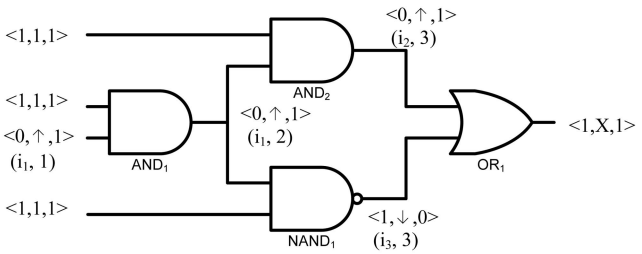


Fig. 5. Examples of maintaining time stamps.

However, as shown above, 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, SPIN-SIM maintains 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; 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 as the transition propagates. 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 $\langle 1, \downarrow, 0\rangle$ with a group ID of i and a time field of t, then the output is $\langle 0, \uparrow, 1\rangle$ with the same group ID i and a time field of $t+1$. Notice that we only maintain time stamps for transitions, that is, $\langle 0, \uparrow, 1\rangle$ and $\langle 1, \downarrow, 0\rangle$. The time stamps for other signal values are not kept, since they are typically not necessary. While this might sacrifice some accuracy, it helps moderate 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 equals 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, Fig. 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 i_3 is assigned to the output and this new group is denoted as a successor of both groups i_1 and i_2. Transitions in a predecessor group precede transitions in its successor groups. SPIN-SIM keeps track of this relation by maintaining *group masks*. Each group i of a total of n groups is assigned a group mask $\mathbf{m}_i = (m_{i1}, m_{i2}, \ldots, m_{in})$, within which every group j is assigned a corresponding bit m_{ij}. A bit m_{ij} is set to 1 if its corresponding group j is a predecessor of the current group i; otherwise, it is set to 0. In the previous example, the two bits corresponding to groups i_1 and i_2 are set to 1 in the mask of group i_3. SPIN-SIM also sets the time field of the output transition to the maximal of those of the input transitions, incremented by 1. In cases c and d, the output transition takes place after either one of the input transitions takes place. The input transition that 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 interest of simulation speed, we decided to avoid representing this information in SPIN-SIM and so we 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. Fig. 6 gives such an example. Since one input to gate AND_1 is $\langle 1, 1, 1\rangle$ and the other is a transition $\langle 0, \uparrow, 1\rangle$ with group ID i_1 and time field 1, the transition propagates to the output of AND_1 and the waveform is $\langle 0, \uparrow, 1\rangle$ with the same group ID and an incremented time field of 2. This output fans out to both gates AND_2 and $NAND_1$, so the transition propagates to their outputs since, for both of them, the other input is $\langle 1, 1, 1\rangle$. Thus, the output waveform of gate AND_2 is $\langle 0, \uparrow, 1\rangle$ and that of gate $NAND_1$ is $\langle 1, \downarrow, 0\rangle$. 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 i_2 to the output transition of gate AND_2 and another new group ID i_3 to that of gate $NAND_1$. Both of their time fields are set to 3, which is the time field of the input transition incremented by 1. The signal group i_1 is set to be the predecessor of both signal groups i_2 and i_3 in their group masks, respectively, which confirms the fact that the output



Group ID: Group mask:

	i_1	i_2	i_3	...
i_1	1	0	0	...
i_2	1	1	0	...
i_3	1	0	1	...
...	...			

Fig. 6. Example of time stamps and group masks.

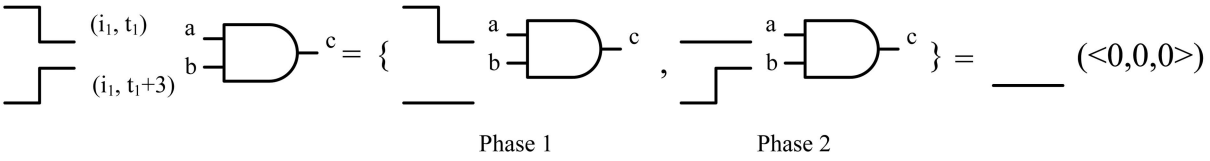


Fig. 7. Example of gate evaluation with time stamps.

transition of AND_1 precedes those of both gates AND_2 and $NAND_1$. The group masks of these signal groups are also illustrated in Fig. 6. Notice that the bit corresponding to group i_1 in the group mask of both i_2 and i_3 is set to 1. Also, 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 AND_2 and $NAND_1$ reach gate OR_1, they belong to different groups. This implies that there is no relative order between them and, therefore, the output waveform of gate OR_1 is $\langle 1, X, 1\rangle$.

The gate evaluation process is also 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 relative timing during gate evaluation. Yet, this method would expand the truth table and would need 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 Fig. 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 $\langle 1, \downarrow, 0\rangle$ and the signal on input b takes its pretransition value, that is, $\langle 0, 0, 0\rangle$. The output is $\langle 0, 0, 0\rangle$ according to the original gate evaluation method. Then, in the second phase, the signal on input a takes its posttransition value, that is, $\langle 0, 0, 0\rangle$, the signal on input b is correspondingly $\langle 0, \uparrow, 1\rangle$, and the output is $\langle 0, 0, 0\rangle$. SPIN-SIM computes the final output as the concatenation of the outputs of the two phases which, in this case, is $\langle 0, 0, 0\rangle$. Note that this is the correct result, which is different from the original truth table result of $\langle 0, X, 0\rangle$.

The complete algorithm for deriving the waveform of the output signal and its time stamp under any possible input combination is formally presented in Section 4.

3.3 Judicious Time Frame Unfolding

SPIN-SIM unfolds time frames carefully. 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. Then, if a transition is detected on any PPO, the corresponding PPI is set accordingly, the circuit is reevaluated, and any hazards are handled as previously described. This process is repeated until no more hazards and transitions occur on PPOs, or until the number of iterations exceeds a predefined limit. Such a limit is necessary to break the infinite loop that occurs when the circuit oscillates, in which case the first terminating condition will never be 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.

We demonstrate this by simulating again the circuit of Fig. 3 using SPIN-SIM. The initial condition is $c = 0$ and $e = 0$. The stimulus is $a = \langle 1, \downarrow, 0\rangle$ and $b = \langle 1, 1, 1\rangle$. 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 and the time stamp on e' with those on e, which are $\langle 0, \uparrow, 1\rangle$ and (0, 3), 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,

TABLE 3
SPIN-SIM Results on the Example Circuit of Fig. 3

Node	1st iter	2nd iter	3rd iter	4th iter
a	$\langle 1, \downarrow, 0\rangle$ $(0, 1)$	$\langle 1, \downarrow, 0\rangle$ $(0, 1)$	$\langle 1, \downarrow, 0\rangle$ $(0, 1)$	$\langle 1, \downarrow, 0\rangle$ $(0, 1)$
b	$\langle 1, 1, 1\rangle$	$\langle 1, 1, 1\rangle$	$\langle 1, 1, 1\rangle$	$\langle 1, 1, 1\rangle$
c'	$\langle 0, 0, 0\rangle$	$\langle 0, 0, 0\rangle$	$\langle 0, \uparrow, 1\rangle$ $(0, 4)$	$\langle 0, \uparrow, 1\rangle$ $(0, 4)$
c	$\langle 0, 0, 0\rangle$	$\langle 0, \uparrow, 1\rangle$ $(0, 4)$	$\langle 0, \uparrow, 1\rangle$ $(0, 4)$	$\langle 0, \uparrow, 1\rangle$ $(0, 4)$
d	$\langle 1, 1, 1\rangle$	$\langle 1, \downarrow, 0\rangle$ $(0, 5)$	$\langle 1, \downarrow, 0\rangle$ $(0, 5)$	$\langle 1, \downarrow, 0\rangle$ $(0, 5)$
e'	$\langle 0, 0, 0\rangle$	$\langle 0, \uparrow, 1\rangle$ $(0, 3)$	$\langle 0, \uparrow, 1\rangle$ $(0, 3)$	$\langle 1, \downarrow, 0\rangle$ $(0, 6)$ $[0, 3]$
e	$\langle 0, \uparrow, 1\rangle$ $(0, 3)$	$\langle 1, \downarrow, 0\rangle$ $(0, 6)$ $[0, 3]$	$\langle 1, \downarrow, 0\rangle$ $(0, 6)$ $[0, 3]$	$\langle 1, \downarrow, 0\rangle$ $(0, 6)$ $[0, 3]$
f	$\langle 0, \uparrow, 1\rangle$ $(0, 2)$	$\langle 0, \uparrow, 1\rangle$ $(0, 2)$	$\langle 0, \uparrow, 1\rangle$ $(0, 2)$	$\langle 0, \uparrow, 1\rangle$ $(0, 2)$

since there is no signal timing information loss. After evaluation, the result is a rising transition on c and a falling transition on e. Given the inputs $f = \langle 0, \uparrow, 1\rangle(0, 2)$ and $d = \langle 1, \downarrow, 0\rangle(0, 5)$, the result on e would be $\langle 0, X, 0\rangle$. 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; the time stamp of the previous one is still kept, as shown in brackets, to indicate the starting point of the current waveform. This time stamp is called *starting* time stamp. In the third iteration, SPIN-SIM corrects the value on c' to $\langle 0, \uparrow, 1\rangle(0, 4)$ but does not change the value on e' to $\langle 1, \downarrow, 0\rangle(0, 6)$, since by doing so, it would discard the old waveform information on e'. Thus, this is a true time frame unfolding.[2] 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 state values 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, an upper limit of simulation iterations is experimentally set (in our experiments, a limit of 50 sufficed to obtain maximal simulation accuracy). When the number of iterations exceeds this limit, SPIN-SIM assumes that the circuit is oscillating and sets the switching signals to destabilizing values.

The complete simulation algorithm and the verification method for conservative time frame unfolding are formally presented in Section 4.

4 Extended 13-Valued Algebra Theory

In this section, we formalize the proposed extension to the 13-valued algebra which enables it to deal with timing constraints in the circuit. First, we introduce notation and definitions, as well as lemmas that capture the causal relationship between input and output transitions and are, thus, used for assigning appropriate time stamps to output transitions during gate evaluation. Then, we examine the necessary conditions for performing conservative time frame unfolding and we prove a theorem that enables an easy method for checking whether a time frame unfolding is conservative. We remind that this capability is crucial, since arbitrary time frame unfolding may overlook possible sequential hazards. Subsequently, we describe the rules for performing gate evaluation in the extended 13-valued algebra, including generation of the output waveform, assignment of appropriate time stamp and, when necessary, verification that any time frame unfolding is conservative. Finally, we describe how these rules are incorporated in Eichelberger's method to enhance simulation accuracy through advanced hazard detection in asynchronous circuits.

4.1 Notation and Definitions

In order to capture the timing constraints used in asynchronous circuits, we extend the 13-valued algebra with time stamps and represent a signal w by a tuple $\{v, \hat{t}, t\}$, where

- v is one of the values in the 13-valued algebra. We denote the set of all the 13-valued waveforms as $\mathcal{V}$, the set of constants in $\mathcal{V}$ as $\mathcal{C} = \{\langle 0, 0, 0\rangle, \langle 1, 1, 1\rangle\}$, the set of transitions as $\mathcal{T} = \{\langle 0, \uparrow, 1\rangle, \langle 1, \downarrow, 0\rangle\}$, and the set of unknown waveforms as $\mathcal{U} = \{v | v \in \mathcal{V} \setminus (\mathcal{C} \cup \mathcal{T})\}$. Also, we denote the initial value of a 13-valued waveform v as $In(v)$ and the final value of v as $Fi(v)$. For instance, $In(\langle 1, \downarrow, 0\rangle) = 1$ and $Fi(\langle 0, X, X\rangle) = X$.
- $\hat{t}$ indicates the start time of v and is valid only if v is a transition ($v \in \mathcal{T}$); otherwise, it is meaningless and is assigned an empty value, which we denote by E. Even when v is a transition, $\hat{t}$ might be undefined if the start time is the simulation initialization, in which case $\hat{t}$ is assigned an E. We also note that a signal $\{v, \hat{t}, E\}$, $\hat{t} \neq E$ is invalid.
- t indicates the time that a transition in v occurs and is also valid only if $v \in \mathcal{T}$; otherwise, if $v \in \mathcal{C} \cup \mathcal{U}$, t is meaningless and is assigned the empty value E.

We call a time stamp t or $\hat{t}$ *trivial* if and only if its value is E. Also, for convenience, we define as $WV(w)$ the 13-valued waveform of the extended waveform $w = \{v, \hat{t}, t\}$, $ST(w)$ the start time of w, and $TT(w)$ the transition time of w. Obviously, $WV(w) = v$, $ST(w) = \hat{t}$, and $TT(w) = t$.

The reason for introducing and assigning time stamps to transitions is to express order between signals. Since the delay of each gate in a Speed-Independent circuit is arbitrary, $\hat{t}$ and t are expressed in the form of relative rather than absolute time. We denote $t_1 \prec t_2$ or $t_2 \succ t_1$ ($t_1 \preceq t_2$ or $t_2 \succeq t_1$) if a time stamp t_1 precedes (does not succeed) another time stamp t_2, $t_1 = t_2$ if t_1 and t_2 are simultaneous, and $t_1 \| t_2$ if there is no relative order between t_1 and t_2, where t_1, t_2 may be either start time stamps or transition time stamps. It is easy to see that the relative signal order is transitive, i.e., if $t_1 \prec (\preceq) t_2$ and $t_2 \prec (\preceq) t_3$, then $t_1 \prec (\preceq) t_3$.

Similar to ternary and 13-valued logic, not all values in the extended 13-valued algebra are exclusive. For instance, $\{\langle X, X, X\rangle, E, E\}$ represents an arbitrary waveform, hence includes all other values. This relationship is represented by imposing a partial ordering between waveforms. We denote $v_1 < v_2$ if the set of the waveforms that v_1 represents is a subset of the waveforms that v_2 represents. In 13-valued algebra, the partial ordering $<$ is defined as illustrated in the Hasse diagram in Fig. 8. In the extended 13-valued algebra, for any two signals $w_1 = \{v_1, \hat{t_1}, t_1\}$ and $w_2 = \{v_2, \hat{t_2}, t_2\}$, where $\hat{t_1} = \hat{t_2}$, the order between them is defined as follows: if $t_1 = t_2$, or $t_1 = E$, or $t_2 = E$, the same order exists as between v_1 and v_2. In addition, for any $w = \{v, \hat{t} \neq E, t \neq E\}$, $w < \{\langle X, X, Fi(v)\rangle, E, E\}$. More importantly, for any ternary function F that is extended to 13-valued or extended

2. Arbitrary time frame unfolding may conceal hazard conditions as discussed in Section 4. Although a verification method for time frame unfolding is developed in Section 4, it might lead to conservative results. Therefore, in order to improve simulation accuracy, SPIN-SIM always performs signal corrections before time frame unfolding.

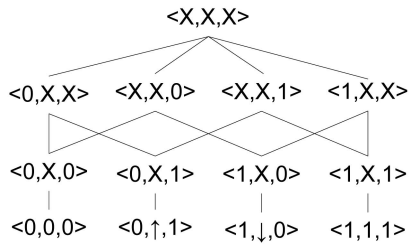


Fig. 8. Partial orders of 13-valued algebra.

13-valued algebra, if $w_1 \leq w_2$, then it holds true that $F(w_1) \leq F(w_2)$. The above property is often used in deriving and proving the following lemmas and gate evaluation rules.

In order to derive the correct relative order between input and output signals during gate evaluation, it is important to know the causal relationship between them. A transition at the output of a combinational gate F is caused by transitions at one or more inputs of the gate. An input waveform v_i *leads to* the output transition v_o when v_i is the necessary condition for v_o. We denote $v_i \rightarrow v_o$ if and only if $v_o \in \mathcal{T}$, $F(v_1, \ldots, v_i, \ldots, v_n) = v_o$, and $F(v_1, \ldots, \langle In(v_i), In(v_i), In(v_i)\rangle, \ldots, v_n) = \langle In(v_o), In(v_o), In(v_o)\rangle$.

Since an output transition must occur no earlier than the input transition that leads to it, the following lemma holds:

Lemma 1. *Given an arbitrary combinational logic gate $F(w_1, \ldots, w_i, \ldots, w_n)$, assume that $w_i = \{v_i, \hat{t}_i, t_i\}$ and $w_o = F(w_1, \ldots, w_i, \ldots, w_n) = \{v_o, \hat{t}_o, t_o\}$. If $v_i, v_o \in \mathcal{T}$ and $v_i \rightarrow v_o$, then $t_i \preceq t_o$.*

For example, suppose that the input waveforms to a 2-input NAND gate are $w_{i_1} = \{\langle 1, 1, 1\rangle, E, E\}$ and $w_{i_2} = \{\langle 1, \downarrow, 0\rangle, E, t_{i_2}\}$, respectively, and the output waveform is $w_o = \{\langle 0, \uparrow, 1\rangle, E, t_o\}$. Obviously, $WV(w_{i_2}) \rightarrow WV(w_o)$, and $t_{i_2} \preceq t_o$. Note that $t_{i_2} = t_o$ only if the NAND gate is ideal and no delay is associated with the output.

Another property of causal relationships between input and output signals, which is important for deriving correctly their relative order, is that an output transition can only be caused by an input transition, as stated in the following lemma:

Lemma 2. *Let v_i and v_o be 13-valued algebra waveforms on an input and the output of an arbitrary combinational gate $F(v_1, \ldots, v_i, \ldots, v_n)$. If $v_o \in \mathcal{T}$, but $v_i \notin \mathcal{T}$, then $v_i \not\rightarrow v_o$.*

Not all output transitions have a unique input transition that leads to them. For example, suppose that the input waveforms to a 2-input NAND gate are $w_{i_1} = \{\langle 1, \downarrow, 0\rangle, E, t_{i_1}\}$ and $w_{i_2} = \{\langle 1, \downarrow, 0\rangle, E, t_{i_2}\}$, respectively, and that the output waveform is $w_o = \{\langle 0, \uparrow, 1\rangle, E, t_o\}$. Here, neither $WV(w_{i_1})$ nor $WV(w_{i_2})$ leads to $WV(w_o)$, despite the fact that either one of them causes the output transition.

In general, an output transition must be caused by a nonempty set of input transitions, as the following lemma states:

Lemma 3. *If $F(v_1, \ldots, v_i, \ldots, v_n) = v_o$ and $v_o \in \mathcal{T}$, then $\exists m\ (m > 0)$ input transitions denoted as $v_{j_k} \in \mathcal{T}$, where $j_k = 1, 2, \ldots, n$, $j_p < j_q$, $p < q$, k, p, $q = 1, 2, \ldots, m$, s.t. $F(v_1, \ldots, v_{j_1 - 1}, \langle In(v_{j_1}), In(v_{j_1}), In(v_{j_1})\rangle, \ldots, \langle In(v_{j_m}), In(v_{j_m}), In(v_{j_m})\rangle, \ldots, v_n) = \langle In(v_o), In(v_o), In(v_o)\rangle$, and $\forall k = 1, 2, \ldots, m$, $F(v_1, \ldots, \langle In(v_{j_1}), In(v_{j_1}), In(v_{j_1})\rangle, \ldots, v_{j_k}, \ldots, \langle In(v_{j_m}), In(v_{j_m}), In(v_{j_m})\rangle, \ldots, v_n) = v_o$.*

4.2 Conservative Time Frame Unfolding

The start time stamp in the extended 13-valued algebra is introduced to support correct gate evaluation in asynchronous sequential circuits. Unlike their synchronous counterparts, wherein a global clock signal indicates the boundaries of time frames within which gate evaluation is safely performed, transitions on feedback lines in asynchronous circuits may happen in arbitrary order and at arbitrary speed. Therefore, the boundaries of time frames are unclear and, thus, time frame unfolding has to be performed carefully during simulation. Otherwise, potential hazards may be overlooked, leading to incorrect evaluation results. For example, suppose that the signal to input 1 of a gate is a sequence of transitions happening at $t_{11}, t_{12}, \ldots, t_{1m}$, respectively, the signal to input 2 is a sequence of transitions happening at $t_{21}, t_{22}, \ldots, t_{2n}$, and that there is no relative order between the two sequences of input transitions. After the first two transitions t_{11} and t_{21} are evaluated, if the value of input 1 reaches t_{12} through time frame unfolding the old value t_{11} is discarded. Thus, it will not be evaluated with any of the transitions $t_{22}, t_{23}, \ldots, t_{2m}$ that have yet to appear on input 2. However, since no relative order is defined between t_{11} and these transitions, they may meet in practice. As a result, evaluation through time frame unfolding may be incorrect and potential hazards may be ignored.

In the following, we discuss the conditions under which a time frame unfolding is conservative, and we prove a theorem which enables a simple method for assessing whether a time frame unfolding is safe to perform during gate evaluation.

Lemma 4. *Let the transition sequence to input 1 of a gate be $t_{11}, t_{12}, \ldots, t_{1m}$ and the transition sequence to input 2 be $t_{21}, t_{22}, \ldots, t_{2n}$. Also assume that the value of input 1 in the previous simulation time frame reaches t_{1i}, $i = 1, 2, \ldots, m-1$ and the value of input 2 in the previous simulation time frame reaches t_{2j}, $j = 0, 1, \ldots, n$, where t_{20} denotes the time before transition t_{21} and $t_{2(n+1)}$ denotes the time after the last transition t_{2n} on input 2. Finally, assume that, $\forall i$, $t_{1i} \preceq t_{2(n+1)}$. Unfolding of the current time frame which replaces t_{1i} on input 1 with $t_{1(i+1)}$ is conservative if $t_{1i} \preceq t_{2(j+1)}$.*

Proof. Due to transitivity, since $t_{1i} \preceq t_{2(j+1)}$ and $t_{2(j+1)} \preceq t_{2(j+2)}, t_{2(j+3)}, \ldots, t_{2n}$, we have $t_{1i} \preceq t_{2(j+1)}, t_{2(j+2)}, \ldots, t_{2n}$, which means that t_{1i} will not meet any of the transitions that have yet to appear on input 2. Thus, no resulting waveform is overlooked by unfolding transition t_{1i} to $t_{1(i+1)}$, so this time frame unfolding is conservative. □

During gate evaluation in an asynchronous sequential circuit, any time frame unfolding that satisfies the condition of Lemma 4 is conservative and the evaluation results are guaranteed to detect any hazard. If it does not satisfy the above condition, however, gate evaluation has to report unknown waveforms to avoid neglecting hazards. Unfortunately, using the condition in Lemma 4 to assess conservativeness of time frame unfolding is problematic in practice.

The reason is that, in order to examine the unfolding to $t_{1(i+1)}$, Lemma 4 needs to check whether $t_{1i} \preceq t_{2(j+1)}$. However, $t_{2(j+1)}$ is not yet available. To resolve this limitation, we prove the following theorem to examine conservativeness of time frame unfolding:

Theorem 1. *Assume that a gate is completely evaluated through time frame unfolding and that no hazard is detected. This gate evaluation is conservative if for each new waveform w_1 that appears at an input of the gate, either $TT(w_1) \succeq ST(w_2)$ or $ST(w_2) = E$, where w_2 is the current waveform on the other input of the gate.*

Proof. Assume that the transition sequence to input 1 of a gate is $t_{11}, t_{12}, \ldots, t_{1m}$, the transition sequence to input 2 is $t_{21}, t_{22}, \ldots, t_{2n}$, and the gate is completely evaluated through time frame unfolding with Theorem 1 satisfied. Assume also that for each new time frame, w_1, which satisfies $TT(w_1) = t_{1(i+1)}$, $ST(w_1) = t_{1i}$, $i = 1, 2, \ldots, m-1$, the previous waveform on input 2 is w_2', and $TT(w_2') = t_{2j}$ (as a special case, if w_2' is the waveform before transition t_{21}, then $j = 0$ and $TT(w_2') = t_{20} = E$). We prove that during the timing frame unfolding, it must hold that $t_{1i} = ST(w_1) \preceq t_{2(j+1)}$. If w_2' is the last transition on input 2 and $j = n$, the above is obviously correct. Otherwise, since the time frames are completely unfolded, let us consider the step when w_2 is unfolded from t_{2j} to $t_{2(j+1)}$. At that time, it holds that $TT(w_2) = t_{2(j+1)} \succeq ST(w_1')$, where w_1' is the waveform on input 1. Since $TT(w_2') = t_{2j}$ in the step immediately before w_1 changes from t_{1i} to $t_{1(i+1)}$, it must be true that $ST(w_1') \succeq t_{1i}$ when w_2 changes from t_{2j} to $t_{2(j+1)}$. Therefore, $t_{2(j+1)} \succeq t_{1i}$ for any step in time frame unfolding. Thus, according to Lemma 4, the gate is conservatively evaluated through this time frame unfolding. □

In order to use Theorem 1 for checking whether a time frame unfolding is conservative during a gate evaluation, we define the following two functions. Function δ checks whether the input waveform has changed into a new transition since the last time that the gate was evaluated. Using function δ and Theorem 1, function ξ examines whether two input waveforms were obtained through conservative time frame unfolding:

$$\delta_w = \begin{cases} 1 & : \text{ if } w \in \mathcal{T}, \text{ and } w \text{ has changed,} \\ 0 & : \text{ otherwise,} \end{cases}$$

$$\xi_{w_1,w_2} = \begin{cases} 1, & \text{if } \delta_{w_1} = 0, \delta_{w_2} = 0, \\ 1, & \text{if } \delta_{w_1} = 1, \delta_{w_2} = 1, \\ & ST(w_2) = E \text{ or } TT(w_1) \succeq ST(w_2), \\ & ST(w_1) = E \text{ or } TT(w_2) \succeq ST(w_1), \\ 1, & \text{if } \delta_{w_1} = 1, \delta_{w_2} = 0, \\ & ST(w_2) = E \text{ or } TT(w_1) \succeq ST(w_2), \\ 1, & \text{if } \delta_{w_1} = 0, \delta_{w_2} = 1, \\ & ST(w_1) = E \text{ or } TT(w_2) \succeq ST(w_1), \\ 0, & \text{otherwise.} \end{cases}$$

4.3 Rules for Gate Evaluation

In the extended 13-valued algebra, signals are annotated with time stamps to capture relative signal order. Thus, the rules for gate evaluation used during simulation need to be extended accordingly. In this section, we first develop evaluation rules for gates with two inputs. Then, we extend the rules for evaluation of multiple-input gates. In order to make this section succinct, we only present evaluation rules for a few representative cases; the rest are derived similarly.

In a two-input gate, new evaluation rules are needed for each possible combination of values of start and transition time stamps of the two input waveforms. Suppose that the combinational function of the gate under evaluation is F, the two inputs are $w_1 = \{v_1, \hat{t_1}, t_1\}$ and $w_2 = \{v_2, \hat{t_2}, t_2\}$, and the output is $w_3 = \{v_3, \hat{t_3}, t_3\}$. When all time stamps are trivial, i.e., when $\hat{t_1} = t_1 = \hat{t_2} = t_2 = E$, the evaluation rules reduce to those used in the traditional 13-valued algebra. However, when the time stamps are nontrivial, new evaluation rules are necessary to deal with them:

1. First, we consider the case that $ST(w_1) = ST(w_2) = TT(w_2) = E$, but $TT(w_1) \neq E$. Suppose that $w_1 = \{v_1, E, t_1\}$, and $w_2 = \{v_2, E, E\}$. Then, the following evaluation rules hold:

$$\begin{aligned} v_3 &= WV(F(w_1, w_2)) = F(v_1, v_2), \\ \hat{t_3} &= ST(F(w_1, w_2)) = E, \\ t_3 &= TT(F(w_1, w_2)) = \begin{cases} t_1 & : \quad v_3 \in \mathcal{T}, \\ E & : \quad \text{otherwise.} \end{cases} \end{aligned} \tag{1}$$

   When $v_3 \in \mathcal{T}$, it must hold that $TT(w_3) = t_1$ for the following reason. According to Lemma 3, if $v_3 \in \mathcal{T}$, it must be true that $v_1 \rightarrow v_3$, since $v_2 \notin \mathcal{T}$. Therefore, $t_3 \succeq t_1$. Thus, when evaluating ideal gates with no output delay, $t_3 = t_1$. Extension of the evaluation rules for practical gates with associated output delay is straightforward, as we describe at the end of this section.
2. Second, we consider the case that $ST(w_1) = ST(w_2) = E$, but neither $TT(w_1)$ nor $TT(w_2)$ is E. Assume that $w_1 = \{v_1, E, t_1\}$, $t_1 \neq E$, and $w_2 = \{v_2, E, t_2\}$, $t_2 \neq E$. Also assume that $v_1 = \langle b_1, \updownarrow, \bar{b_1} \rangle$ and $v_2 = \langle b_2, \updownarrow, \bar{b_2} \rangle$, where b_1 and b_2 are either 0 or 1. In the following, we consider the cases that $t_1 \| t_2$ and $t_1 \prec t_2$, respectively; the remaining cases are similarly derived:

   a. If $t_1 \| t_2$, the following evaluation rules hold:

$$\begin{aligned} v_3 &= WV(F(w_1, w_2)) = F(v_1, v_2), \\ \hat{t_3} &= ST(F(w_1, w_2)) = E, \\ t_3 &= \begin{cases} t_1, & \text{if } v_3 \in \mathcal{T}, v_1 \rightarrow v_3, \\ & \quad v_2 \not\rightarrow v_3, \\ t_2, & \text{if } v_3 \in \mathcal{T}, v_1 \not\rightarrow v_3, \\ & \quad v_2 \rightarrow v_3, \\ Late(t_1, t_2), & \text{if } v_3 \in \mathcal{T}, v_1, v_2 \rightarrow v_3, \\ \| t, \forall t \neq t_3, & \text{if } v_3 \in \mathcal{T}, v_1, v_2 \not\rightarrow v_3, \\ E, & \text{otherwise.} \end{cases} \end{aligned}$$

Note that function $Late(t_1, t_2)$ in the above equations returns the later one of t_1 and t_2. When both v_1 and v_2 lead to v_3, v_3 succeeds both of them, i.e., it does not precede the later one. Such a case is illustrated in the example of Figs. 5a and 5b.

b. If $t_1 \prec t_2$, the evaluation procedure takes place in three phases. In the first phase, w_1 is evaluated with $w_2' = \{\langle b_2, b_2, b_2\rangle, E, E\}$ according to rule (1). Assume that the result is $w_3' = \{v_3', E, t_3'\}$. Second, $w_1'' = \{\langle \bar{b_1}, \bar{b_1}, \bar{b_1}\rangle, E, E\}$ is evaluated with w_2. Assume that the result is $w_3'' = \{v_3'', E, t_3''\}$. In the last phase, w_3' and w_3'' are combined into the final waveform as follows:

i. If $v_3' \in \mathcal{U}$ or $v_3'' \in \mathcal{U}$, then

$$\begin{aligned} WV(w_3) &= v_3 = \langle In(v_3'), X, Fi(v_3'')\rangle, \\ ST(w_3) &= \hat{t_3} = E, \\ TT(w_3) &= t_3 = E. \end{aligned}$$

ii. Otherwise, if both v_3' and $v_3'' \in \mathcal{C}$, then

$$\begin{aligned} WV(w_3) &= v_3 = v_3' = v_3'', \\ ST(w_3) &= \hat{t_3} = E, \\ TT(w_3) &= t_3 = E. \end{aligned}$$

Such an evaluation case is illustrated in the example of Fig. 7.

iii. Otherwise, if only $v_3'(v_3'') \in \mathcal{T}$, then

$$\begin{aligned} WV(w_3) &= v_3 = v_3'(v_3''), \\ ST(w_3) &= \hat{t_3} = E, \\ TT(w_3) &= t_3 = t_3'(t_3''). \end{aligned}$$

iv. Otherwise, both v_3' and $v_3'' \in \mathcal{T}$ and the following equations hold:

$$\begin{aligned} WV(w_3) &= v_3 = v_3'', \\ ST(w_3) &= \hat{t_3} = t_3', \\ TT(w_3) &= t_3 = t_3''. \end{aligned}$$

Note that, in this case, since a waveform in the extended 13-valued algebra can only express one transition, the previous transition is discarded and its transition time stamp is used as the start time stamp of the temporary result before the postprocessing step which we describe later in the section.

3. Third, we consider the case that one input waveform has nontrivial start and transition time stamps, while the other has trivial start but nontrivial transition time stamps, for instance, $w_1 = \{v_1, \hat{t_1}, t_1\}$, $\hat{t_1}$, $t_1 \neq E$ and $w_2 = \{v_2, E, t_2\}$, $t_2 \neq E$, and $\hat{t_1} \prec t_2$. Since w_1 and w_2 might be new waveforms that have changed since the last time the gate was evaluated, the evaluation process first needs to verify whether time frame unfolding is conservative by using Theorem 1:

$$\begin{aligned} v_3' &= WV(F(\{v_1, E, t_1\}, \{v_2, E, t_2\})), \\ WV(w_3) = v_3 &= \begin{cases} v_3', & \text{if } v_3' \notin \mathcal{U}, \\ & \xi_{w_1,w_2} = 1, \\ \langle X, X, Fi(v_3')\rangle, & \text{otherwise}, \end{cases} \\ \hat{t_3'} &= ST(F(\{v_1, E, t_1\}, \{v_2, E, t_2\})), \\ ST(w_3) = \hat{t_3} &= \begin{cases} \hat{t_3'} & : \ \hat{t_3'} \neq E, v_3 \notin \mathcal{U}, \\ \hat{t_1} & : \ \hat{t_3'} = E, v_3 \notin \mathcal{U}, \\ E & : \ \text{otherwise}, \end{cases} \\ t_3' &= TT(F(\{v_1, E, t_1\}, \{v_2, E, t_2\})), \\ TT(w_3) = t_3 &= \begin{cases} E, & \text{if } v_3 \in \mathcal{U}, \\ t_3', & \text{if } v_3 \notin \mathcal{U}, \\ & t_3' \succ \hat{t_1} \text{ or } t_3' = E, \\ \succ \hat{t_3}, & \text{otherwise}. \end{cases} \end{aligned}$$

Note that, in the last case of the above equation, we force $t_3 \succ \hat{t_3}$ because it is possible that $t_3' \| t$, $\forall t \neq t_3'$ according to the previous evaluation rules. There may be no relative order between t_3 and t_1 or t_2, but t_3 must succeed $\hat{t_3}$.

4. Fourth, we consider the case that both of the two input waveforms have nontrivial start and transition time stamps. Assume that $w_1 = \{v_1, \hat{t_1}, t_1\}$, $\hat{t_1}$, $t_1 \neq E$, and $w_2 = \{v_2, \hat{t_2}, t_2\}$, $\hat{t_2}$, $t_2 \neq E$. The evaluation result is subject to the relative order between $\hat{t_1}$, t_1, $\hat{t_2}$, and t_2:

a. If $\hat{t_1} \| \hat{t_2}$, then

$$\begin{aligned} v_3' &= WV(F(\{v_1, E, t_1\}, \{v_2, E, t_2\})), \\ \hat{t_3'} &= ST(F\{v_1, E, t_1\}, F\{v_2, E, t_2\}, \\ t_3' &= TT(F(\{v_1, E, t_1\}, \{v_2, E, t_2\})), \\ v_3 &= \begin{cases} v_3', & \text{if } v_3' \in \mathcal{C}, \\ & \xi_{w_1,w_2} = 1, \\ v_3', & \text{if } v_3' \in \mathcal{T}, t_3' \succ \hat{t_1}, \hat{t_2}, \\ & \hat{t_3'} = E \text{ or } \hat{t_3'} \succ \hat{t_1}, \hat{t_2}, \\ & \xi_{w_1,w_2} = 1, \\ \langle X, X, Fi(v_3')\rangle, & \text{otherwise}, \end{cases} \\ t_3 &= \begin{cases} t_3' & : \ v_3 \notin \mathcal{U}, \\ E & : \ v_3 \in \mathcal{U}, \end{cases} \\ \hat{t_3} &= \begin{cases} \hat{t_3'}, & \text{if } \hat{t_3'} \neq E, v_3 \notin \mathcal{U}, \\ Late(\hat{t_1}, \hat{t_2}), & \text{if } \hat{t_3'} = E, v_3 \notin \mathcal{U}, \\ E, & \text{if } v_3 \in \mathcal{U}. \end{cases} \end{aligned}$$

b. Otherwise, if $\hat{t_1} \prec t_1 \preceq \hat{t_2} \prec t_2$, then

$$v'_3 = WV(F(\{\langle Fi(v_1), Fi(v_1), Fi(v_1)\rangle, E, E\}, \{v_2, E, t_2\})),$$
$$v_3 = \begin{cases} v'_3, & \text{if } v'_3 \notin \mathcal{U}, \\ & \xi_{w_1,w_2} = 1, \\ \langle X, X, Fi(v'_3)\rangle, & \text{otherwise}, \end{cases}$$
$$\hat{t_3} = \begin{cases} E & : \ v_3 \in \mathcal{U}, \\ \hat{t_2} & : \ \text{otherwise}, \end{cases}$$
$$t'_3 = TT(F(\{\langle Fi(v_1), Fi(v_1), Fi(v_1)\rangle, E, E\}, \{v_2, E, t_2\})),$$
$$t_3 = \begin{cases} E & : \ v_3 \in \mathcal{U}, \\ t'_3 & : \ \text{otherwise}. \end{cases}$$

c. Otherwise, if $\hat{t_1} \preceq \hat{t_2}$, then

$$w_3 = F(\{v_1, E, t_1\}, \{v_2, \hat{t_2}, t_2\}), \qquad \hat{t_2} \| t_1.$$

d. Otherwise, if $\hat{t_2} \prec t_2 \preceq \hat{t_1} \prec t_1$, then

$$v'_3 = WV(F(\{v_1, E, t_1\}, \{\langle Fi(v_2), Fi(v_2), Fi(v_2)\rangle, E, E\})),$$
$$v_3 = \begin{cases} v'_3, & \text{if } v'_3 \notin \mathcal{U}, \\ & \xi_{w_1,w_2} = 1, \\ \langle X, X, Fi(v'_3)\rangle, & \text{otherwise}, \end{cases}$$
$$t'_3 = TT(F(\{v_1, E, t_1\}, \{\langle Fi(v_2), Fi(v_2), Fi(v_2)\rangle, E, E\})),$$
$$\hat{t_3} = \begin{cases} E & : \ v_3 \in \mathcal{U}, \\ \hat{t_1} & : \ \text{otherwise}, \end{cases}$$
$$t_3 = \begin{cases} E & : \ v_3 \in \mathcal{U}, \\ t'_3 & : \ \text{otherwise}. \end{cases}$$

e. Otherwise, if $\hat{t_2} \preceq \hat{t_1}$, then

$$w_3 = F(\{v_1, \hat{t_1}, t_1\}, \{v_2, E, t_2\}), \qquad \hat{t_1} \| t_2.$$

Multiple-input gates. Evaluation of multiple-input gates can be easily performed by using the above two-input gate evaluation rules. For multiple-input gates implementing associative functions, a simple method is to compute the function input by input, using the evaluation rules for two-input gates, i.e., $F(I_1, I_2, I_3) = F(F(I_1, I_2), I_3)$. In the general case, a multiple-input gate can be replaced during evaluation by its equivalent model composed with two-input simple gates.

Postprocessing. After the resulting output waveform w'_o of a combinational gate F is computed according to the above evaluation rules, if it has a nontrivial start time stamp, it needs to be postprocessed in order to generate the final result. Assume that the original waveform of the gate output before evaluation was w''_o. The following rule is used to generate the final waveform w_o:

$$w_o = \begin{cases} w'_o, & \text{if } ST(w'_o) = E, \\ \{WV(w'_o), TT(w''_o), TT(w'_o)\}, & \text{if } ST(w'_o) \neq E, \\ & WV(w'_o) \in \mathcal{T}, \\ & ST(w'_o) \preceq TT(w''_o), \\ w''_o, & \text{if } ST(w'_o) \neq E, \\ & WV(w'_o) \in \mathcal{C}, \\ \{\langle X, X, Fi(w'_o)\rangle, E, E\}, & \text{otherwise}. \end{cases} \tag{2}$$

In the above equation, the newly calculated waveform w'_o is checked against the original waveform w''_o in order to ensure that no waveform is skipped between the two waveforms, in which case potential hazards may be ignored.

We also point out that (2) is valid only for evaluation of an ideal gate, i.e., when no delay is associated with the gate output. For a gate with output delay, the time stamps of w_o are derived in the following way to reflect the delay between the inputs and the output:

$$w_o = \begin{cases} w'_o, & \text{if } ST(w'_o) = E, \\ & WV(w'_o) \notin \mathcal{T}, \\ \{WV(w'_o), E, t_o\}, & \text{if } ST(w'_o) = E, \\ & WV(w'_o) \in \mathcal{T}, \\ \{WV(w'_o), TT(w''_o), t_o\}, & \text{if } ST(w'_o) \neq E, \\ & WV(w'_o) \in \mathcal{T}, \\ & ST(w'_o) \preceq TT(w''_o), \\ w''_o, & \text{if } ST(w'_o) \neq E, \\ & WV(w'_o) \in \mathcal{C}, \\ \{\langle X, X, Fi(w'_o)\rangle, E, E\}, & \text{otherwise}, \end{cases}$$
$$t_o \succ TT(w'_o).$$

4.4 Simulation Algorithm

The above rules for gate evaluation and time frame unfolding are incorporated into Eichelberger's method to enhance simulation accuracy. Signal waveforms are expressed in the extended 13-valued algebra, gates are evaluated accordingly, and time frame unfolding is checked for conservativeness. Specifically, in Procedure A, changing PIs are set to the corresponding waveforms in $\mathcal{T}$, each with a new transition time stamp. All other PIs and PPIs are set to the corresponding waveforms in $\mathcal{C}$ or $\mathcal{U}$ if unknown. Then, POs and PPOs are evaluated following the corresponding rules in Section 4.3. The main steps of the simulation method are described in Algorithm 1.

Algorithm 1. Simulation using extended 13-valued algebra

$i \Leftarrow 0$; PI^0, $PPI^0 \Leftarrow$ initial values
repeat {Procedure A}
  evaluate PO^i, PPO^i; $i \Leftarrow i + 1$
  if $WV(PPO^{i-1}) \in \mathcal{U}$ **then** {any hazard on PPO^{i-1}}
    $PPI^i \Leftarrow$ corresponding destabilizing value
  else
    $PPI^i \Leftarrow PPO^{i-1}$
  end if
until $PPI^i = PPI^{i-1}$, or $i = Max$
while $PPI^{i-1} \neq PPO^{i-1}$ **do** {Procedure B}
  $PPI^i \Leftarrow PPO^{i-1}$
  evaluate PO^i, PPO^i; $i \Leftarrow i + 1$
end while

5 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

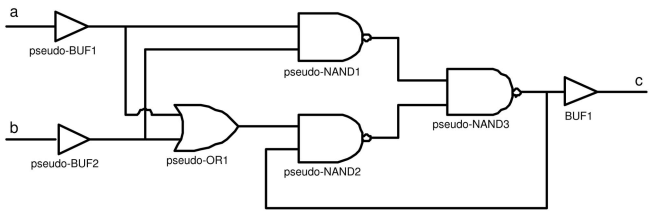


Fig. 9. C-element: pseudogate implementation.

domino structures are also frequently used in the design of Speed-Independent circuits. While it is possible to treat them as simple gates, as in [15], 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 multivalued algebras augments the memory cost significantly by increasing the base number.

Instead, 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 mimic the original complex gate in both functionality and timing, employing both pseudogates and real gates. Fig. 9 gives an example of a pseudogate implementation of a C-element. All gates in this example are pseudogates, except for the output-stage buffer. SPIN-SIM assumes no delay when a signal propagates through a pseudogate; 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. Pseudobuffers are also inserted in the inputs to make internal fan-out invisible to the outside of the complex gate. To support SPIN-SIM, we developed a library of gate and pseudogate equivalent implementations for the most commonly used complex gates. In this way, SPIN-SIM simulates complex gates accurately and efficiently, without incurring additional memory cost.

6 Fault Simulation

Fault simulation of Speed-Independent circuits is similar to that of synchronous circuits. The targeted fault list includes all stuck-at faults on inputs or output of gates, except those inside a complex gate, and may be pruned through equivalent fault collapsing to speed up fault simulation. Each test vector is simulated on the good circuit and on each bad circuit, where a single stuck-at fault from the collapsed 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 in [22]. We refer to fault equivalence/dominance in a single gate as g-equivalence/dominance, in a combinational circuit as c-equivalence/dominance, 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 [22], 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.

TABLE 4
Fault Collapsing Results

Circuit Name	# of Gates	Complex Gates	# of Faults	Collapsed Faults	Collapsing Rate (%)
alloc-outbound	12	7	70	41	41.4
chu133	10	5	54	32	40.7
chu150	10	4	56	34	39.3
converta	9	3	54	37	31.5
dff	8	6	44	28	36.4
ebergen	14	7	74	46	37.8
half	3	3	22	15	31.8
hazard	8	5	48	33	31.2
master-read	25	19	144	86	40.3
mp-forward-pkt	11	5	60	34	43.3
mr1	27	16	152	93	38.8
nak-pa	16	12	82	48	41.5
nowick	11	5	56	28	50.0
ram-read-sbuf	16	11	90	55	38.9
rcv-setup	7	4	40	25	37.5
rpdft	12	6	62	34	45.2
sbuf-ram-write	20	9	110	69	37.3
sbuf-send-ctl	17	12	94	59	37.2
seq4	16	8	96	63	34.4
Average					38.7

Although efficient parallel fault simulation techniques have been devised for synchronous circuits [23], 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 [15], SPIN-SIM performs fault simulation serially.

7 Experimental Results

SPIN-SIM has been developed in C, based on the simulation engine of HOPE [23]. 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 standard set of Speed-Independent circuits synthesized by Petrify [24]. The name, number of gates, and number of complex gates of each benchmark are listed in the first, second, and third columns of Table 4, respectively. Complex gates like Muller C-elements are handled as described in Section 5. In each benchmark, a reset input is assumed to be connected to every memory

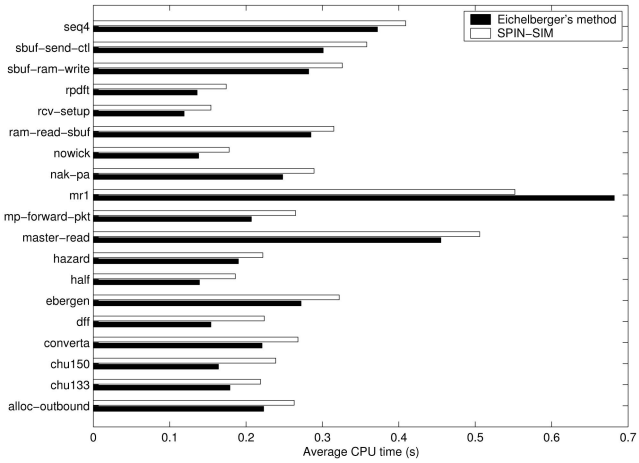


Fig. 10. Simulation CPU time.

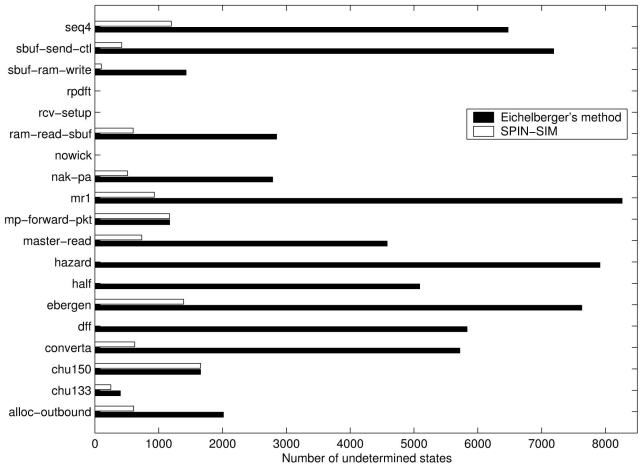


Fig. 11. Number of undetermined states.

element to appropriately initialize the circuit. Experiments were performed on a workstation with dual Xeon 1.7-GHz processors and 1 Gbyte of RAM. For each benchmark, we fault simulated 10,000 random test vectors generated through the method described in [25]. 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 [10].

First, in Fig. 10, we compare 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 percent 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 percent 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 percent more CPU time than Eichelberger's method. Yet, the accuracy of simulation is significantly improved. To demonstrate this, Fig. 11 shows the number of undetermined states for each circuit during the simulation of the random patterns by each of the two methods. Evidently, Eichelberger's method fails to resolve many hazard-free circuit states, and reports them as undetermined falsely, while the techniques employed in SPIN-SIM result in a simulation where circuit states are resolved much more accurately.

By overcoming the conservativeness of Eichelberger's method, when SPIN-SIM is used for fault simulation it also achieves a significant improvement in the reported fault coverage. Before comparing the two methods, we first perform equivalent fault collapsing as discussed in Section 6, the results of which are given in Table 4. The number of stuck-at faults in each circuit before and after fault collapsing is listed in the fourth and fifth column in Table 4, respectively, along with the fault collapsing rate in the sixth column. Although we only collapsed g-equivalent faults, the fault collapsing rate is still considerable, averaging at 38.7 percent. As illustrated in Fig. 12, across all benchmark circuits, SPIN-SIM achieves an average fault coverage of 97.1 percent, while Eichelberger's simulation method achieves an average fault coverage of only 75.6 percent using the same set of randomly generated test vectors. In Fig. 13, we also illustrate the CPU time needed by each method to fault simulate the 10,000 random test vectors on each benchmark circuit. For several circuits, such as *chu150* and *rpdft*, SPIN-SIM spends 14 percent to 70 percent 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, which 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 CPU time. The normalized savings for these experiments were 33.9 percent.

8 Other Asynchronous Circuits

Although the proposed simulation method was developed for Speed-Independent circuits, it can be extended to handle

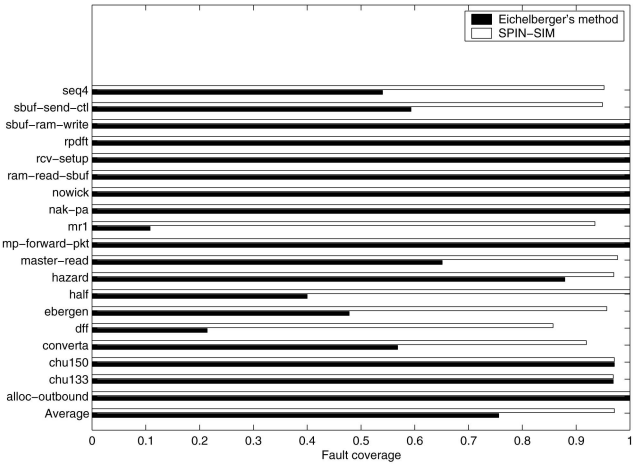


Fig. 12. Stuck-at fault coverage.

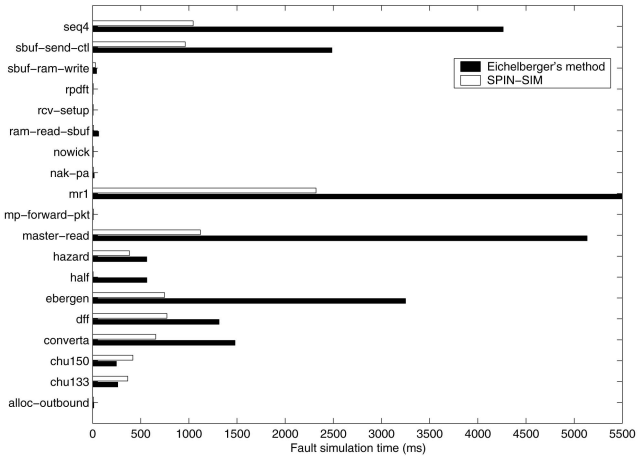


Fig. 13. Fault simulation time.

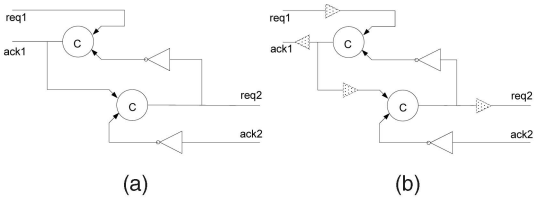


Fig. 14. (a) A Delay-Insensitive circuit and (b) its equivalent Speed-Independent circuit.

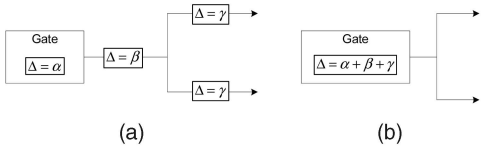


Fig. 15. (a) An isochronic fork and (b) its equivalent Speed-Independent circuit.

other classes of asynchronous circuits. For example, we can employ the method in [26] to simulate Delay-Insensitive circuits using SPIN-SIM. Delay-Insensitive circuits operate correctly under the unbounded delay model, which allows arbitrary gate and wire delays. Unlike in Speed-Independent circuits, the wires in Delay-Insensitive circuits also have unbounded delay. Therefore, we preprocess and transform the circuit into a new circuit by inserting buffers on the wires. As a result, under the Speed-Independent timing model, the transformed circuit exhibits the same timing properties as the original circuit. Therefore, the original circuit can be handled by simulating the transformed circuit in SPIN-SIM. In Fig. 14, we give an example of a Delay-Insensitive circuit (Fig. 14a) and the transformed circuit used for simulation by SPIN-SIM (Fig. 14b). The inserted buffers, which are used to mimic the timing of the original Delay-Insensitive circuit, are shown in dashed lines. We also note that it is not necessary to insert a buffer in every segment of a wire, since some of them can be combined with the delay of the gates that drive them.

Quasi-Delay-Insensitive circuits can also be simulated by SPIN-SIM by preprocessing them in a similar way. However, such circuits employ *isochronic forks*, in which the delay on all fan-out branches of a wire is assumed to be equal. These isochronic forks need to be handled differently. In general, we transform an isochronic fork into an equivalent Speed-Independent circuit form, as shown in Fig. 15. Therefore, it is not necessary to insert buffers in these isochronic forks. Of course, buffers are still inserted in other parts of the circuit, including nonisochronic forks. After the preprocessing, Quasi-Delay-Insensitive circuits can be handled by SPIN-SIM.

9 Conclusion

Accurate logic and fault simulation of asynchronous circuits requires efficient and precise detection of both combinational and sequential hazard conditions. Simulators for synchronous circuits are not concerned with this problem, while previously developed simulators for asynchronous circuits solve it only partially and for certain classes of circuits. To address this issue, we have developed SPIN-SIM, a logic and fault simulation algorithm for Speed-Independent and various other classes of asynchronous circuits by extending Eichelberger's classical hazard detection method. In addition to adopting a 13-valued algebra, SPIN-SIM maintains the relative order of causal signal transitions, unfolds time frames carefully, and handles complex gates through a pseudogate 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] A.J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The Design of an Asynchronous MIPS R3000 Microprocessor," *Proc. 17th Conf. Advanced Research in VLSI (ARVLSI '97)*, pp. 164-181, 1997.

[2] S.B. Furber, D.A. Edwards, and J.D. Garside, "AMULET3: A 100 MIPS Asynchronous Embedded Processor," *Proc. Int'l Conf. Computer Design (ICCD '00)*, pp. 329-334, 2000.

[3] A. Bink and R. York, "ARM996HS: The First Licensable, Clockless 32-Bit Processor Core," *IEEE Micro*, vol. 27, no. 2, pp. 58-68, 2007.

[4] F. te Beest, A. Peeters, M. Verra, K. van Berkel, and H. Kerkhoff, "Automatic Scan Insertion and Test Generation for Asynchronous Circuits," *Proc. Int'l Test Conf. (ITC '02)*, pp. 804-813, 2002.

[5] F. te Beest and A. Peeters, "A Multiplexer Based Test Method for Self-Timed Circuits," *Proc. Int'l Symp. Asynchronous Circuits and Systems (ASYNC '05)*, pp. 166-175, 2005.

[6] M. Roncken, E. Aarts, and W. Verhaegh, "Optimal Scan for Pipelined Testing: An Asynchronous Foundation," *Proc. Int'l Test Conf. (ITC '96)*, pp. 215-224, 1996.

[7] S. Banerjee, S.T. Chakradhar, and R.K. Roy, "Synchronous Test Generation Model for Asynchronous Circuits," *Proc. Int'l Conf. VLSI Design (ICVD '96),* pp. 178-185, 1996.
[8] M.A. Breuer, "The Effects of Races, Delays, and Delay Faults on Test Generation," *IEEE Trans. Computers,* vol. 23, no. 10, pp. 1078-1092, 1974.
[9] O. Roig, J. Cortadella, M.A. Peiia, and E. Pastor, "Automatic Generation of Synchronous Test Patterns for Asynchronous Circuits," *Proc. Design Automation Conf. (DAC '97),* pp. 620-625, 1997.
[10] E.B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM J. Research and Development,* vol. 9, no. 2, pp. 90-99, 1965.
[11] W.A. Clark, "Macromodular Computer Systems," *Proc. AFIPS Spring Joint Computer Conf.,* vol. 30, pp. 335-336, 1967.
[12] C.J. Myers and T. Meng, "Synthesis of Timed Asynchronous Circuits," *IEEE Trans. VLSI Systems,* vol. 1, no. 2, pp. 106-119, June 1993.
[13] D. Muller and W. Bartky, *A Theory of Asynchronous Circuits,* Annals of Computing Laboratory of Harvard Univ., pp. 204-243, 1959.
[14] C.J. Myers, *Asynchronous Circuit Design.* John Wiley & Sons, 2001.
[15] S. Sur-Kolay, M. Roncken, K. Stevens, P.P. Chaudhuri, and R. Roy, "Fsimac: A Fault Simulator for Asynchronous Sequential Circuits," *Proc. Ninth Asian Test Symp. (ATS '00),* pp. 114-119, 2000.
[16] S.H. Unger, *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, 1969.
[17] J.P. Hayes, "Digital Simulation with Multiple Logic Values," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 5, no. 2, pp. 274-283, Apr. 1986.
[18] T.J. Chakraborty, V.D. Agrawal, and M.L. Bushnell, "Delay Fault Models and Test Generation for Random Logic Sequential Circuits," *Proc. Design Automation Conf. (DAC '92),* pp. 165-172, 1992.
[19] S. Chakraborty, D. Dill, and K. Yun, "Min-Max Timing Analysis and an Application to Asynchronous Circuits," *Proc. IEEE,* vol. 87, no. 2, pp. 332-346, 1999.
[20] D.S. Kung, "Hazard-Non-Increasing Gate-Level Optimization Algorithms," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '92),* pp. 631-634, 1992.
[21] J. Brzozowski and Z. Esik, "Hazard Algebras," *Formal Methods in System Design,* vol. 23, no. 3, pp. 223-256, 2004.
[22] J.E. Chen, C.L. Lee, and W.J. Shen, "Single-Fault Fault-Collapsing Analysis in Sequential Logic Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 10, no. 12, pp. 1559-1568, Dec. 1991.
[23] H.K. Lee and D.S. Ha, "Hope: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 15, no. 9, pp. 1048-1058, Sept. 1996.
[24] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE Trans. Information and Systems,* vol. E80-D, no. 3, pp. 315-325, 1997.
[25] F. Shi and Y. Makris, "Fault Simulation and Random Test Generation for Speed-Independent Circuits," *Proc. Great Lakes Symp. VLSI (GLSVLSI '04),* pp. 127-130, 2004.
[26] D.L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* MIT Press, 1989.

Feng Shi received the BEng and the MEng degrees from Tsinghua University in 2000 and 2002, respectively, and the PhD degree from Yale University in 2007, all in electrical engineering. He is currently working with the RFIC design group in Skyworks Solutions, Inc., Irvine, California. His current research interests include asynchronous circuit design and testing, mixed-signal IC design, computer architecture, and low power design. He is a member of the IEEE.

Yiorgos Makris received the diploma of computer engineering and informatics from the University of Patras, Greece, in 1995, and the MS and PhD degrees in computer science and engineering from the University of California, San Diego, in 1997 and 2001, respectively. He is currently an associate professor of electrical engineering and computer science at Yale University, New Haven, Connecticut, where he leads the Testable and Reliable Architectures (TRELA) Research Group. His current research interests include soft-error mitigation in digital circuits, machine learning-based testing of analog/RF circuits, as well as test and reliability of asynchronous circuits. He is a senior member of the IEEE.