# SPIN-TEST: Automatic Test Pattern Generation 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

*SPIN-TEST is a simulation-based gate-level ATPG system for Speed-Independent circuits. Its core engine is an A* search algorithm which employs an accurate fault simulator and an efficient cost function to guide a deterministic test pattern generation phase. A random test pattern generation phase is also available in order to improve run time. The key ATPG challenge in Speed-Independent circuits is the generation of patterns that are valid independently of the relative timing and the order of arrival of signals. SPIN-TEST addresses this challenge by guaranteeing fault sensitization with hazard/race-free patterns and response observation that is not affected by oscillations or non-deterministic circuit states. Experimental results on benchmark circuits demonstrate the efficiency of SPIN-TEST in terms of both high fault coverage and low test generation time.*

## 1. Introduction

The recent resuscitation of asynchronicity as a solution to several limitations encountered in submicron technology [1] has sparked new research interest in many aspects of asynchronous design automation, including test [2, 3, 4]. Advantages such as reduced power dissipation, elimination of clock distribution issues, modularity, and improved performance have enabled asynchronous circuits to carve a widening niche in many systems previously dominated by their synchronous counterparts. Asynchronous circuits do have drawbacks however. Without a global clock for synchronization, they operate in a rather independent manner, which is sensitive to race conditions and hazards, making them cumbersome to design and test. Uncontrolled feedbacks amplify this situation and make validation and simulation of asynchronous circuits even harder. Compounding this difficulty, EDA tools available for asynchronous design and test are neither as abundant nor as refined as those for synchronous.

With regards to test, controlling and observing internal nodes in asynchronous circuits is much harder than in synchronous circuits. Even worse, DFT remedies for synchronous sequential circuits, such as scan-based design, may be impractical and unacceptably expensive for asynchronous circuits, since they introduce more feedback and state holding elements. Moreover, several classes of asynchronous circuits exist and test methods for one class might not work for other classes due to different timing assumptions.

In this paper, we focus on automatic test pattern generation (ATPG) for Speed-Independent designs, one of the main classes of asynchronous circuits. First, in section 2, we discuss the challenges of ATPG for Speed-Independent circuits. Then, in section 3, we describe SPIN-TEST, the simulation-based, gate-level ATPG system that we developed for this class of circuits. Next, in section 4, we provide an example and finally, in section 5, we report experimental results.

## 2. ATPG in Speed-Independent Circuits

In this section, we introduce the class of Speed-Independent circuits, we discuss the challenges in designing an efficient ATPG algorithm for this class, and we review related research efforts.

### 2.1. Speed-Independent Circuits

Asynchronous circuits are classified into two main categories according to their design style, namely *Huffman* and *Muller* circuits. Huffman circuits [5] are designed using a traditional asynchronous state machine approach. The state is stored in combinational feedback loops. 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 feedback state variables arrive at the inputs. Violation of this assumption may result in a sequential *hazard*. 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.

In contrast, Muller circuits [5] are designed mainly based on signal transition graphs (STG, or Petri nets) as the specification form and operate under the *unbounded* gate delay model. In this model, circuits are guaranteed to work regardless of gate delays, yet 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 [5], and although the two terms are used interchangeably in the literature, we will only use the latter in the rest of the paper. A Speed-Independent circuit example is given in Figures 1-2. Figure 1 illustrates the circuit specification, which in this case is an STG. The underlined signals in the figure are primary inputs, while the remaining signals are primary outputs. A "+" indicates a rising transition while a "-" indicates a falling transition of a signal. The STG indicates the causal relationship between the signals, where a transition occurs when *all* the predecessor transitions have occurred. The two solid circles, called *initial marking*, represent the initial state of the circuit, which in this case is $ack1 = 0$, $ack2 = 0$, $req1 = 0$, $req2 = 0$, and $CSC0 = 0$. Notice that the initial state is part of the circuit specification, resembling the existence of a global reset line in a state machine representation. The gate level implementation, including C-Elements[1], is shown in Figure 2.

[1] A C-Element is a state holding element that assumes the state of its inputs after both inputs make the same transition.

0-7803-8702-3/04/$20.00 ©2004 IEEE.

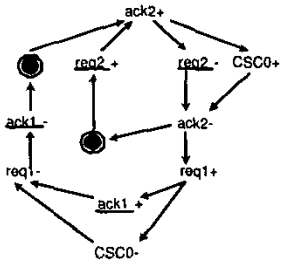


**Figure 1. Signal Transition Graph Example**

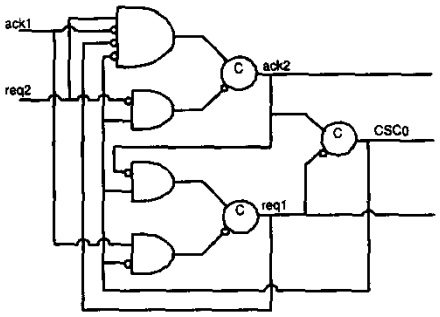


**Figure 2. Corresponding Speed-Independent Circuit**

## 2.2. ATPG Challenges

The unbounded delay model of Speed-Independent circuits implies more lax timing assumptions than those of Huffman circuits, which assume a bounded gate delay model and a fundamental mode of operation. Consequently, test generation solutions for Huffman circuits are not directly applicable to Speed-Independent circuits, which present their own set of challenges:

**Hazard/race-free pattern generation:** A key challenge in ATPG for Speed-Independent Circuits is the generation of hazard/race-free test patterns under the unbounded gate and negligible wire delay model. Test generation algorithms for synchronous circuits do not need to worry about combinational timing; thus, when these methods are used in asynchronous circuits they do not guarantee hazard/race-free patterns. Patterns with hazards or races, however, are invalid for testing asynchronous circuits. Moreover, a test generation algorithm for one class of asynchronous circuits is not necessarily appropriate for other classes of asynchronous circuits. For example, an algorithm generating test sequences that are hazard-free under the bounded gate and wire delay model, is adequate for Huffman circuits; yet it may result in test sequences that are hazardous under the unbounded gate delay and negligible wire delay model. ATPG for Speed-Independent circuits should, thus, detect and/or avoid hazards/races under the appropriate timing assumptions.

**Stable-state response observation:** As an additional challenge, an ATPG algorithm for Speed-Independent circuits has to guarantee that faults are detected in a stable state, such that they can be observed by existing Automatic Test Equipment (ATE) for synchronous circuits. Unlike synchronous circuits where all states are stable, asynchronous circuits operate in a rather independent manner and have several unstable states. While stopping the clock preserves the state of a synchronous circuit independent of what state it is in, stopping an asynchronous circuit in an unstable state is not feasible. Therefore, a test sequence which detects the fault in an unstable state is invalid for testing asynchronous circuits, simply because the fault effect cannot be observed by synchronous circuit ATE.

**Efficiency:** An ATPG algorithm for Speed-Independent circuits should be efficient not only in identifying patterns to activate the fault and propagate the fault effect, but also in guaranteeing that the generated test patterns are hazard free and the fault effect is observed in a stable state. The simplest approach would be to first generate the test patterns without considering these restrictions, then identify and discard the invalid ones through simulation and repeat the process. However, such an approach may be overly time-consuming and inefficient. Ideally, an algorithm should be able to generate patterns that *inherently* observe these restrictions. Additionally, methods for generating and efficiently fault-simulating pseudo-random patterns, identifying redundant faults, and collapsing equivalent faults should be employed to prune the fault-list and improve run-time. We note, however, that these tasks are relatively much more difficult in Speed-Independent circuits than in synchronous circuits.

## 2.3. Related Work

Several deterministic test pattern generation algorithms have been proposed for asynchronous circuits. Banerjee et al. [6] proposed a method for modelling an asynchronous circuit as a synchronous circuit for test purposes by inserting a virtual flip-flop into each feedback path. Thus, test patterns can be automatically generated by state-of-the-art synchronous ATPG techniques. The synchronous test generation model for asynchronous circuits ensures that faults are detected in stable states. However, some of these vectors may cause hazard/race conditions and are invalid for test. Therefore, fault simulation is needed to discard these vectors, reducing fault coverage. While this is not a significant problem in Huffman circuits, it becomes critical in Speed-Independent circuits. In the former, most sequential hazards are avoided by assuming fundamental mode of operation and by inserting delay elements in the feedback paths. In the latter, however, no such assumptions are made and therefore more hazards/races may take place, leading to invalidation of many test vectors generated through the above method. Moreover, as pointed in [7], the test validation method in [6] leads to optimistic results.

Roig et al. [7] also proposed a deterministic method for generating synchronous test patterns for asynchronous circuits. In their method, the fault free circuit is first analyzed to find all input sequences without hazards and oscillations and to generate a *Confluent Stable State Graph (CSSG)*. Then, a symbolic ATPG strategy is employed on the *CSSG* to identify test sequences for each fault. This method assures that the generated test vectors are valid. However, since it generates the CSSG by analyzing the non-faulty circuit to find all hazard-free and race-free input sequences, it is computationally expensive, especially as the size of the circuit becomes larger. Moreover, a large percentage of faults is typically detected by a random ATPG phase and the deterministic method is only employed for the few remaining faults. Thus, building and maintaining the complete CSSG is often unnecessary.

Cheng et al. [8] introduced a simulation-based directed search approach for generating test vectors for synchronous and asynchronous sequential circuits. In this method, a *threshold-value* model was developed to incorporate signal controllability information, based on which a cost function is computed by the simulator to guide the selection of test vectors. This method has two advantages over conventional approaches. First, it avoids the high complexity caused by back-tracking in space and time in conventional methods. In addition, fault simulation deals with circuit delays in a very natural manner and is able to detect hazards/races and oscillations; thus, it generates tests that are guaranteed to be valid. However, the guided search algorithm in [8] is not complete since the search

may be terminated at a local cost minimum. Hence the algorithm may fail to identify test vectors for some testable faults. Furthermore, this method is unable to determine redundant faults and stop the search in an early stage; thus it often engages in vain computation, traversing the search space until it aborts.

## 3. Proposed ATPG Method

Similar to the method proposed in [8], SPIN-TEST is a *simulation-based* test pattern generation algorithm. The main advantage of simulation-based ATPG is that it can naturally detect and avoid hazards and races and, thus, generate hazard/race-free test patterns. This is very important in Speed-Independent circuits, wherein hazards and races often lead to non-deterministic behavior, which will typically invalidate the corresponding test patterns and reduce fault coverage. Therefore, efficient logic and fault simulation is instrumental for successful test pattern generation. The simulation engine of SPIN-TEST is based on SPIN-SIM [9], a novel logic and fault simulation method that was recently developed for Speed-Independent circuits.

### 3.1. Overview

As illustrated in Figure 3, SPIN-TEST is essentially an A* search algorithm [10]. Given a fault, SPIN-TEST first examines whether the fault is redundant in the corresponding combinational circuit, in which case it proceeds to the next fault. Otherwise, SPIN-TEST constructs the necessary search objectives for activating and propagating the fault. It then starts by fault-simulating the initial state of the circuit. If the fault is detected, SPIN-TEST reports the corresponding test patterns and continues with the next fault in the faultlist. Otherwise, SPIN-TEST fault-simulates one of the possible Single-Input-Change (SIC) vectors (i.e. vectors at Hamming distance one from the current test vector). If the fault is detected, SPIN-TEST reports the corresponding test patterns. Otherwise, it estimates the cost of detecting the fault by continuing with this vector, and stores it into a queue in ascending cost order. After all SIC vectors are tried, SPIN-TEST removes the cheapest vector from the queue and uses it as the current test vector. This procedure is repeated until a sequence of test vectors that detects the fault is found, or until the queue size limit is reached, in which case it aborts on the fault. A test sequence with hazards/races is naturally avoided because SPIN-SIM detects them by reporting $X$ on state/output signals, which, hence, greatly increases its cost value.

SPIN-TEST only tries test vectors which are at unit Hamming distance from the current test vector. However, as observed in [9], this does not limit fault detectability. This holds because Speed-Independent circuits operate without any restrictions on the speed or order of input changes. Therefore, if a fault in a Speed-Independent circuit is detectable by a test sequence which includes a Multiple-Input-Change (MIC) vector, then the fault is also detectable by the test sequence obtained by arbitrarily breaking down the MIC vector into a sequence of SIC vectors. Also, by allowing only SIC vectors, the number of trials in each iteration is very small (i.e. equal to the number of inputs). Moreover, as pointed out in [8], SIC test vectors are known to produce fewer hazards in asynchronous circuits, hence they are less likely to result in invalid test patterns.

SPIN-TEST is a complete search algorithm, unlike the test generation algorithm in [8]. The latter accepts only input changes that reduce the employed cost function, hence it may abandon the search due to a local minimum. In contrast, SPIN-TEST stores suboptimal trials and revisits them if the estimated optimal trial fails. Thus, if a solution exists, it is able to recover from a local minimum and detect the fault. As a result, SPIN-TEST aborts on a fault only if it is undetectable within the preset limit of allowed trials.

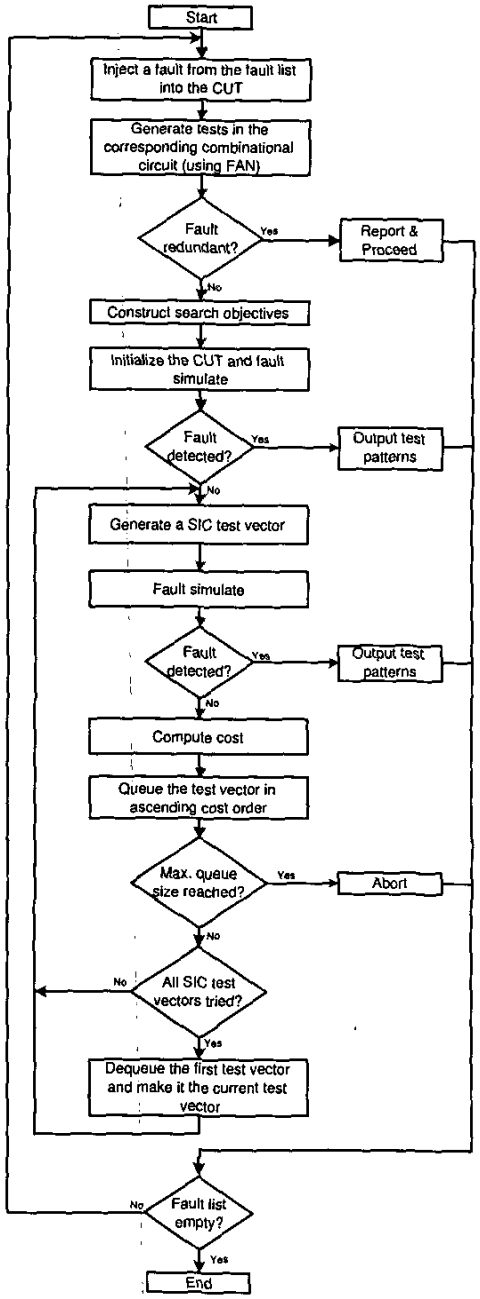


**Figure 3. Flowchart of Proposed Algorithm**

### 3.2. Redundant Fault Identification

As mentioned in [8], simulation-based ATPG is inefficient in handling redundant faults. The search algorithm essentially has no way of proving redundancy so it keeps trying all possibilities until a preset limit is exceeded, which often causes serious performance degradation. In order to overcome this limitation, SPIN-TEST employs a mechanism to detect redundant faults in an early stage. It exploits the fact that a fault in a Speed-Independent circuit cannot be detected if it is *c-redundant*, (i.e. it is redundant in the cor-

responding combinational logic derived by arbitrarily cutting each feedback path in the original asynchronous circuit into a pseudo-primary input (PPI) and a pseudo-primary output (PPO)). This happens because, when the feedback is included, such a fault cannot lead to deterministically different good machine and faulty machine responses. While the state of the faulty machine may be affected by a glitch, the timing specifications of Speed-Independent circuits place no constraints on the duration of such a glitch. Therefore, it is not guaranteed that the faulty machine state will differ from the good machine state. Thus, such a fault can at best be *potentially detected* in a Speed-Independent circuit.

Before searching for a test for a given fault, SPIN-TEST checks whether it is c-redundant, in which case it removes it from the fault list saving valuable search time. For this purpose, SPIN-TEST adopts the FAN algorithm [11]. Interestingly, even if the fault is not c-redundant, the test vectors found during this effort are useful for the rest of the process. In essence, these vectors indicate a partial solution for exciting and propagating the fault within the Speed-Independent circuit. Thus, they point to a direction for searching for a test sequence so they can be used as initial search objectives. In this way, SPIN-TEST combines redundant fault identification and establishment of appropriate search objectives for the simulation-based phase into a single step.

## 3.3. Cost Computation

SPIN-TEST is an informed search algorithm, thus its performance is strongly dependent on the heuristic cost function. Ideally, this cost function should reflect the hardness of detecting the given fault by accepting the current test vector. SPIN-TEST first tries to find a sequence of test vectors to activate the fault and then a sequence of test vectors to propagate the fault effect to a primary output. Since the objective is different in fault activation than in fault propagation, SPIN-TEST has a different cost function for each stage.

In the fault activation stage, SPIN-TEST tries to activate the fault by applying test patterns that detect the fault in the corresponding combinational circuit. Hence, given a target test pattern, the cost function is proportional to the Hamming distance between the current circuit state and the objective test pattern. However, since some state signals are easier to set than others, SPIN-TEST accounts for this difference by assigning weights to each state signal according to its input level[2]. In general, the input level reflects the difficulty of setting a state signal to a specific value, so SPIN-TEST sets its control cost to $2^{l_{in}}$, where $l_{in}$ denotes its input level. The total cost for setting the state signals to the corresponding values of the objective pattern is the sum of the control costs of the state signals that are different in the current state. In addition, since SPIN-TEST only allows SIC test vectors, the Hamming distance between the current primary inputs and those of the objective test pattern indicates the minimal number of vectors needed to reach the objective state. Yet, since primary inputs are much easier to control than internal states, the contribution of this Hamming distance to the cost function has a lower weight. Moreover, the current length of the test sequence also contributes to the cost activation function. Finally, since the objective test patterns may only activate the fault, the cost function also includes the estimated cost for propagating the fault effect. The latter is a cost estimate for driving the nearest fault effect in the objective pattern to a primary output. If the lowest output level of the state signal carrying a fault effect in the objective pattern is $l_{out}$, the estimated fault propagation cost is $2^{l_{out}}$. The final cost is equal to the minimal cost for any of the objective test patterns, since any of them may activate the fault effect. In summary, the fault activation cost function is:

$$C_A = \min_{1 \le i \le n} \{ \sum_{PI_j \in PI} I(PI_j, Obj_i) + \sum_{PPI_k \in PPI} I(PPI_k, Obj_i) \times 2^{l_{in}(PPI_k)} + C_{prop\_estimated}(Obj_i) \} + 2 \times curr\_len$$

In this equation, $Obj_i$ denotes one of the $n$ possible objective test patterns returned by FAN, $PI_j$ denotes an element of the set of primary inputs $PI$, $PPI_k$ denotes an element of the set of pseudo primary inputs in the corresponding combinational circuit $PPI$, $curr\_len$ denotes the length of the current test sequence, $C_{prop\_estimated}(Obj_i)$ returns the estimated fault propagation cost for $Obj_i$, $l_{in}(PPI_k)$ returns the input level of $PPI_k$, and $I(S, Obj_i)$ is a binary function which returns 0 if the value of signal $S$ equals that of the corresponding signal in objective pattern $Obj_i$, and 1 otherwise.

Once a fault effect appears on any state signal, SPIN-TEST enters the fault propagation stage. In this stage, the cost function should reflect the difficulty for fault propagation from the current state. SPIN-TEST takes into account the proximity of the state signal where the fault effect appears to a primary output and sets the corresponding fault propagation cost in exponential proportion to the output level of that state signal. The final cost value is the minimal cost for propagating the fault effect from any of the state signals. In addition, the final cost also includes the length of the current test sequence, since it reflects the total cost for detecting the fault. In summary, the fault propagation cost function is:

$$C_P = \min_{PPO_i \in PPO'} 2^{l_{out}(PPO_i)} + 2 \times curr\_len$$

where $PPO_i$ denotes an element in the set of pseudo primary outputs carrying fault effects $PPO'$, $l_{out}(PPO_i)$ returns the output level of $PPO_i$, and $curr\_len$ is the length of the current test sequence. Note that, in some cases, the fault effect is directly activated on a primary output instead of a state signal and the fault propagation stage is skipped.

## 3.4. Improving ATPG performance

Fault-simulation of randomly generated patterns is often performed before deterministic algorithms to improve ATPG performance in synchronous circuits. In the case of Speed-Independent circuits, random test generation remains a very efficient method, although it may need to be adapted to the particularities of Speed-Independent circuits to achieve higher performance [9]. Hence, we include an optional random test generation phase through which many faults are detected and dropped from the fault list before the deterministic algorithm of SPIN-TEST is invoked.

Fault collapsing is another efficient method for reducing the workload of test generation in synchronous sequential circuits, which can also be applied in Speed-Independent circuits. SPIN-TEST employs the fault collapsing method for Speed-Independent circuits that was proposed in [9]. More specifically, it collapses g-equivalent [12] faults (i.e. single-gate equivalent faults) before any test pattern generation is performed. Since the collapsing rate is typically quite significant, a lot of unnecessary workload is avoided, hence the overall ATPG performance is improved.

[2]Input levelization is performed in the standard method: primary inputs are assigned level 0, state signals connected to primary inputs either directly or through combinational logic are assigned level 1, state signals connected to signals in level 1 but not level 0 are assigned level 2, and so on. Output levelization is also performed in a similar way.

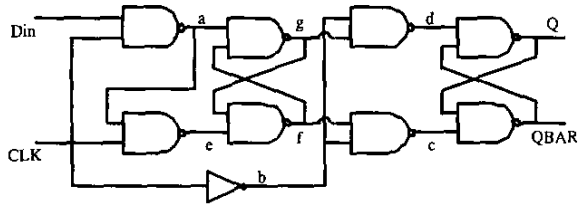


**Figure 4. Schematic of a D Flip-Flop**

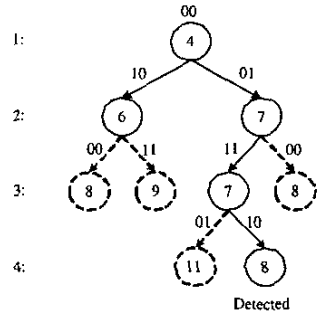


**Figure 5. An Example of Test Search**

### 3.5. Discussion

The deterministic test search algorithm of SPIN-TEST exhibits several advantages. **First,** since the cost function includes the length of the current test sequence, SPIN-TEST is guided towards finding the shortest test sequence. **Second,** unlike static algorithms which only focus on one way of fault activation or propagation before they backtrack and try another alternative, SPIN-TEST is much more dynamic. More specifically, it simultaneously considers multiple objective patterns for fault activation and multiple ways of fault propagation during the search process and dynamically selects the test sequence which is considered to be optimal at each step. Thus, SPIN-TEST has a more global view during the search process which allows it to calibrate the discrepancy between the estimated and the actual cost. **Third,** SPIN-TEST pursues fault activation and response propagation concurrently and is allowed to switch between objectives. Since the only essential difference between the two phases is the computation of the cost function, they are both handled by the same process in SPIN-TEST. This allows exploration of more global test optimization opportunities.

## 4. An Example

In order to illustrate how SPIN-TEST works, we present an example in which it generates test patterns to detect the fault $b/0$ in the circuit of the D flip-flop shown in Figure 4. Assume that initial state of the circuit is $g = 0$, $f = 1$, $Q = 0$, $QBAR = 1$, $D_{in} = 0$ and $CLK = 0$, and that SPIN-TEST generates the corresponding combinational circuit by cutting the feedback paths $g \rightarrow f$ and $Q \rightarrow QBAR$.

SPIN-TEST first runs FAN on the corresponding combinational circuit, and finds the two test patterns $(D_{in}, CLK, g', Q', Q, QBAR) = (X, 0, 1, 0, \mathbf{D}, 1)$ and $(X, 0, 0, 1, \overline{\mathbf{D}}, \mathbf{D})$, where $g'$ and $Q'$ are the pseudo primary inputs obtained by cutting the feedback paths. We also use the standard $\mathbf{D}$–notation where $\mathbf{D}$ ($\overline{\mathbf{D}}$) denotes that the value is 1 (0) in the fault-free circuit and 0 (1) in the faulty circuit. SPIN-TEST sets these two patterns as the objective patterns, and starts searching for a test sequence to detect the fault. The search process is illustrated in figure 5, wherein the number in each circle indicates the current cost and the two bits above each arrow indicate the values of $D_{in}$ and $CLK$, respectively. SPIN-TEST starts at step 1, where both $D_{in}$ and $CLK$ are zero. It first tries to keep $CLK$ at zero, since it activates the fault. However, after selecting vector 10 in step 2, SPIN-TEST finds that the cost of continuing along this path keeps becoming higher (i.e. it would become 8 or 9) due to the increasing length of the test sequence, since the fault effect cannot be propagated to any primary outputs when $CLK$ is zero. Since this cost exceeds that of the other possible SIC test pattern (01) in step 2, SPIN-TEST *switches* to it as the current test pattern. Then SPIN-TEST sets $D_{in}$ to one in step 3 (11), which preserves the cost of 7 because this vector helps in propagating the fault effect. Finally, SPIN-TEST finds a way to both activate the fault and propagate the fault effect to $Q$ by lowering $CLK$ in step 4 (10), hence the fault is detected and the test sequence $00/01/11/10$ is reported.

## 5. Experimental Results

We developed SPIN-TEST in C, based on ATALANTA [13], which is an implementation of the FAN ATPG algorithm, and SPIN-SIM [14], which is a recently developed engine for logic and fault simulation of Speed-Independent circuits. The input netlist is in ISCAS89 format and the stuck-at fault list can be either provided through a file or generated automatically. In the latter case, all single stuck-at faults on gate inputs and outputs are injected and $g$-equivalent faults are collapsed. SPIN-TEST provides an optional random test generation phase, implementing the algorithm of [9] with a user-defined termination condition (i.e. the maximal number of consecutive test vectors that detect no faults). Then, the simulation-based, deterministic test pattern generation phase described in Section 3 is performed *for each* remaining fault.

We experimented with SPIN-TEST on a set of Speed-Independent benchmark circuits synthesized by Petrify [15]. In each benchmark, a reset input port is assumed to be connected to every memory element, so that the circuit can be initialized to the initial state provided in the specification, as explained in section 2.1. The experiments were performed on a workstation with dual Xeon 1.7GHz processors and 1 gigabyte of RAM. The initial size of the search queue was set to 1K. After the algorithm terminates, the user has the option to increase this size and repeat ATPG on the aborted faults.

We first ran SPIN-TEST with the random test generation phase disabled. Table 1 illustrates the results. The name of each example circuit is listed in the first column, and the number of total faults, the number of faults after $g$-equivalent fault collapsing, and the collapsing rate for each circuit are listed in the second, third, and fourth column respectively. An average fault collapsing rate of 38.3% is achieved across these example circuits, which reaffirms that fault collapsing is still an efficient technique to improve ATPG performance. The number of total detected faults for each circuit is listed in the fifth column. The sixth column presents the number of redundant faults in the corresponding combinational circuit of each example circuit. These c-redundant faults are identified before the test vector search algorithm takes place, thus wasted computational time is avoided. The number of aborted faults during the deterministic ATPG algorithm for each circuit is listed in the seventh column. These faults were aborted because no test sequence was found for them before the maximal size of the search queue was reached. The total fault coverage for each circuit is listed in the eighth column. An average fault coverage of 97.3% is achieved across all example circuits. The ninth column presents the CPU time that SPIN-TEST spent on each circuit for all faults (detected + aborted) using only the deterministic method. The tenth column presents the CPU time

| Circuit Name | Total Faults | Faults after Collapsing | Collapsing Rate (%) | Deterministic ATPG (Random Phase Disabled) | | | | | | Random Phase Enabled | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Detected Faults | c-redundant Faults | Aborted Faults | Fault Coverage (%) | CPU Time1(s) | CPU Time2(s) | Randomly Detected Faults | CPU Time3(s) |
| alloc-outbound | 70 | 41 | 41.4 | 41 | 0 | 0 | 100 | 0.015 | 0.015 | 41 | 0.005 |
| chu133 | 54 | 32 | 40.7 | 31 | 0 | 1 | 96.9 | 0.006 | 0.005 | 31 | 0.002 |
| chu150 | 56 | 35 | 37.5 | 34 | 1 | 0 | 97.1 | 0.006 | 0.006 | 34 | 0.003 |
| converta | 54 | 38 | 29.6 | 35 | 0 | 3 | 91.9 | 0.010 | 0.010 | 33 | 0.006 |
| dff | 44 | 28 | 36.4 | 24 | 0 | 4 | 85.7 | 0.013 | 0.009 | 7 | 0.013 |
| ebergen | 74 | 46 | 37.8 | 44 | 0 | 2 | 95.7 | 0.025 | 0.021 | 44 | 0.010 |
| half | 22 | 15 | 31.8 | 15 | 0 | 0 | 100 | 0.003 | 0.003 | 15 | 0.002 |
| hazard | 48 | 33 | 31.2 | 32 | 0 | 1 | 97.0 | 0.012 | 0.011 | 32 | 0.004 |
| mp-forward-pkt | 60 | 34 | 43.3 | 34 | 0 | 0 | 100 | 0.009 | 0.009 | 34 | 0.003 |
| mr1 | 152 | 93 | 38.8 | 87 | 0 | 6 | 93.5 | 3.008 | 1.243 | 65 | 2.881 |
| nak-pa | 82 | 48 | 41.5 | 48 | 0 | 0 | 100 | 0.015 | 0.015 | 46 | 0.005 |
| nowick | 56 | 28 | 50.0 | 28 | 0 | 0 | 100 | 0.003 | 0.003 | 27 | 0.002 |
| ram-read-sbuf | 90 | 55 | 38.9 | 55 | 0 | 0 | 100 | 0.033 | 0.033 | 41 | 0.014 |
| rcv-setup | 40 | 25 | 37.5 | 25 | 0 | 0 | 100 | 0.004 | 0.004 | 24 | 0.002 |
| rpdft | 62 | 34 | 45.2 | 34 | 0 | 0 | 100 | 0.008 | 0.008 | 34 | 0.003 |
| sbuf-ram-write | 110 | 69 | 37.3 | 69 | 0 | 0 | 100 | 0.174 | 0.174 | 58 | 0.134 |
| sbuf-send-ctl | 94 | 59 | 37.2 | 56 | 0 | 3 | 94.9 | 0.101 | 0.068 | 48 | 0.070 |
| seq4 | 96 | 63 | 34.4 | 60 | 1 | 2 | 95.2 | 0.334 | 0.170 | 47 | 0.322 |
| vbe6a | 88 | 56 | 36.4 | 56 | 0 | 0 | 100 | 0.119 | 0.119 | 56 | 0.007 |
| Average | | | 38.3 | | | | 97.3 | | | | |

**Table 1. Experimental Results**

SPIN-TEST spent only on the detected faults for each circuit. As may be observed, SPIN-TEST is very efficient in finding test patterns for detectable faults. Yet it is more time-consuming in handling the few aborted faults, where the search terminates only after the queue size limit is reached.

We then ran SPIN-TEST again, with the random test generation option enabled. The termination limit for the random test generation algorithm [9] was set to 16 consecutive vectors with no additional faults detected. The number of faults detected through random test generation for each circuit is listed in the eleventh column. The total CPU time of the random and deterministic methods for each circuit on all faults (detected + aborted) is listed in the twelfth column. For most circuits, the random method detects a large percentage of the total faults and in some cases such as *alloc-outbound* and *vbe6a* it detects all faults. This results in a considerable performance improvement in test generation time. Some circuits, such as *dff* and *mr1*, contain a sizeable number of random test resistant faults, therefore the performance improvement for them is smaller.

The run-time of our method is several orders of magnitude faster than the run-time reported in [7], yet the fault coverage of the two methods is almost identical. Nevertheless, such a comparison would be misleading for the following two reasons: **a)** the experiments were performed on different platforms so time comparison is infeasible, and **b)** the method in [7] assumes the ability to initialize the circuit into several states, while SPIN-TEST assumes only the single initial state provided in the specification of each circuit [16].

## 6. Conclusion

Efficient testing of Speed-Independent circuits requires test patterns that are immune to hazards and race conditions and guarantee the observation of faulty responses in stable states. Toward this end, we developed SPIN-TEST, a simulation-based gate-level ATPG system for Speed-Independent circuits. SPIN-TEST addresses the aforementioned challenges by guaranteeing fault sensitization with hazard/race-free patterns and response observation that is not affected by oscillations or non-deterministic circuit states. The deterministic test generation phase of SPIN-TEST is an A* search algorithm which employs a cost function to select appropriate patterns for fault simulation. A random test generation phase is also available in order to improve performance. Experimental results on benchmark circuits demonstrate that SPIN-TEST achieves an average fault coverage of 97.3% in very low test generation time.

## References


[1] C. Tristam, "It's time for clockless," *Technology Review*, pp. 37–41, 2001.

[2] P. J. Hazewindus, "Testing delay insensitive circuits," *Ph.D. Thesis, Department of Computer Science, California Institute of Technology*, 1992.

[3] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: A survey," *Technical Report CS-TR-94-03-06, Department of Computer Science and Engineering, University of Washington*, 1994.

[4] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin, "Partial-scan delay fault testing of asynchronous circuits," *IEEE Transactions on Computers*, vol. 17, pp. 1184–1198, 1998.

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

[6] 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.

[7] 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.

[8] K.-T. Cheng, V. D. Agrawal, and E. S. Kuh, "A simulation-based method for generating tests for sequential circuits," *IEEE Transactions on Computers*, vol. 39, no. 12, pp. 1456–1463, 1990.

[9] 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*, pp. 127–130, 2004.

[10] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 2002.

[11] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, 1983.

[12] 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.

[13] H. K. Lee and D. S. Ha, *On the Generation of Test Patterns for Combinational Circuits*, Technical Report No. 12_93, Dep't of Electrical Eng., Virginia Polytechnic Institute, 1993.

[14] F. Shi and Y. Makris, "Spin-sim: Logic and fault simulation for speed-independent circuits," in *Proceedings of International Test Conference*, 2004.

[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] Oriol Roig, "Personal communication," 2004.