SPIN-PAC: Test Compaction 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-PAC is a static test compaction method for Speed-Independent circuits. We demonstrate how the test sets can be compacted by combining multiple consecutive test vectors within a test sequence into a vector pair of higher Hamming distance, and by eliminating or pruning independent test sequences. We discuss the exponential nature of optimally solving this problem, we propose an efficient algorithm to approximate it, and we evaluate its performance through experiments.

I. INTRODUCTION

Asynchronous circuits have recently attracted increased interest because of their potential for high performance, low power, design reusability, and elimination of clock distribution and skew problems. Widespread acceptance of asynchronous circuits, however, requires the development of efficient CAD tools to support their design and test. Yet such tools are neither as abundant nor as efficient as their counterparts for synchronous circuits. In addition, several classes of asynchronous circuits are developed based on different timing assumptions, each requiring customized design and test algorithms. In this work, we focus on test methods for the class of *Speed-Independent* circuits and, more specifically, we address the problem of test compaction.

Speed-Independent circuits [1] are a class of asynchronous circuits which are guaranteed to work under the *unbounded gate delay* model, or regardless of gate delays, assuming negligible wire delays. Speed-Independent circuit design requires explicit knowledge of the behavior protocol allowed by the environment, normally specified in the form of signal transition graphs (or Petri Nets). However, no restrictions are imposed on the order or speed that inputs, outputs, and state signals change, except that they must behave according to the protocol.

Recently, a number of efforts have been devoted to Speed-Independent circuit testing. In [2], SPIN-SIM, a fault simulator for Speed-Independent circuits was developed, which adopts a 13-valued algebra, maintains the relative order of causal signal transitions, and unfolds time frames judiciously in order to improve simulation accuracy. In [3], a random test generation algorithm for Speed-Independent circuits was developed, which reduces the probability that the circuit finds itself in non-deterministic states and helps it recover when this happens. In [4], SPIN-TEST, a simulation-based gate-level ATPG system for Speed-Independent circuits was developed, whose core engine is an A* search algorithm employing an efficient cost function to guide the deterministic test pattern generation phase.

However, as we explain in Section II, in an effort to deal with the particularities of asynchronous circuit test generation and to improve ATPG speed, these test generation methods yield test sets that are often longer than necessary. The underlying reason is that they generate a set of independent test sequences that consist of consecutive test vectors that are only allowed to be unit Hamming distance apart. To address this problem, we developed an efficient post-generation (static) test compaction method in order to reduce the number of test vectors without sacrificing fault coverage. Our method, which is described in Section III, employs two techniques: combination of multiple single-input change (SIC) vectors into a multiple-input change (MIC) vector pair, and re-ordering of independent test sequences, which results in their pruning or elimination. The proposed algorithm first employs a bidirectional search within each test sequence, which yields multiple alternative subsequences of smaller sizes, wherein SIC vectors are merged into MIC vectors. Then, the algorithm selects a set of subsequences of minimal total length, such that all faults detected by the original test sequences are still detected. This problem is formulated as an integer linear program (ILP), which is approximated through linear programming and randomized rounding. Experimental results demonstrating the efficiency of the proposed method are provided in Section IV.

II. MOTIVATION

Test patterns generated for asynchronous circuits have to be hazard-free, or the circuit may enter a nondeterministic state and the fault will not be detected. In [3, 4], random and deterministic methods were developed to generate hazard-free test patterns for Speed-Independent circuits. In both methods, SPIN-SIM [2], an accurate fault simulator is used to guarantee that the patterns are indeed hazard-free. Furthermore, in [3], the random test generation algorithm restricts tests to SIC vectors which have a significantly lower probability of invoking a hazard. In addition, it monitors the circuit state and automatically resets the circuit once it goes into a nondeterministic state, in order to avoid generating useless test patterns. Therefore, the generated test patterns are a set of independent test sequences, which can be applied in any order. Similarly, in the deterministic test generation algorithm presented in [4, 5], only SIC vectors are considered in order to not only avoid non-deterministic states, but also to reduce the search space and accelerate the ATPG process. Although this restriction does not sacrifice fault coverage, as demonstrated in [3], it often leads to longer test sequences. Furthermore, test sequences are generated for a targeted fault and the process is repeated for the next fault from the reset state. Consequently, vectors are again grouped into independent test sequences, which can be shuffled arbitrarily.

These two features, namely the independence of test sequences and the SIC restriction for consecutive vectors within each test sequence, provide a lot of opportunity for test compaction. This can be demonstrated through a simple Speed-Independent circuit example. Suppose that we are generating test patterns for the example circuit in Fig. 1, which has similar functionality to a D flip-flop. Assume that the initial state is $phi = 1$ and $D = 0$, and that all internal feedback lines are at 0. The test sequence generated by SPIN-TEST for the fault $[Q_C2]$ stuck-at one is shown in Fig. 2(a). Since the initial state of Q is zero, we have to set Q to one and then to zero through a test sequence of seven SIC vectors, including initialization, in order to detect this fault. But in fact, the fault can still be detected if vector 5 is eliminated and the two SIC vectors are replaced by a single MIC vector, as shown in Fig. 2(b). The possibility of eliminating or pruning independent test sequences can also be illustrated using this example. Suppose that we generate three test sequences to detect the three faults: Q stuck-at

0-7803-8736-8/05/$20.00 ©2005 IEEE.

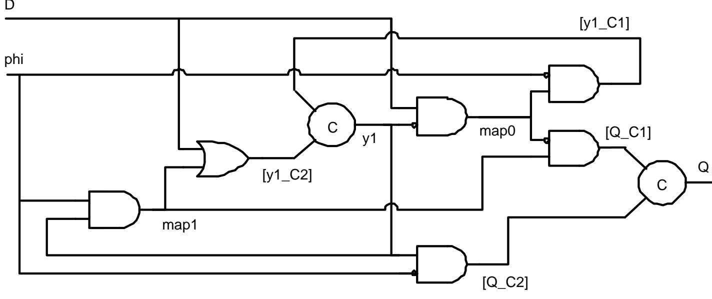

Fig. 1. Example Speed-Independent Circuit

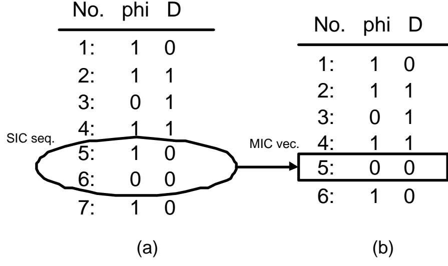

Fig. 2. Compaction of SIC Using MIC Vectors

TABLE I
TEST SEQUENCE PRUNING/ELIMINATION

	$Q/1$	$D/1$	$[y1_C1]/1$
S_1	0	0	1
S_2	0	3	1
S_3	2	6	1

1, D stuck-at 1, and $[y1_C1]$ stuck-at 1. The first sequence, S_1, is 10, the second one, S_2, is $10 \rightarrow 00 \rightarrow 10$, and the third one, S_3, is the one shown in Fig. 2(b), where the first bit represents the value of phi and the second bit represents the value of D. The faults that each sequence can detect are illustrated in Table I. A positive number represents the vector within the corresponding sequence after which the fault is detected, while a zero implies that the fault cannot be detected by the sequence. The total cost of these three sequences is 10 vectors (i.e. the sum of the maximum entries on each row). Clearly, the fault detected by sequence S_1 can also be detected by either one of the other sequences, hence S_1 can be dropped. Furthermore, fault $D/1$ can also be detected by S_2, so the last 4 vectors of S_3 can be pruned. As a result, we can keep only the sequence S_2 and the first two vectors of S_3, which reduces the total number of vectors to 5. Notice that if we kept only S_3, all faults would still be detected but by a longer test sequence, i.e. 6 vectors.

III. PROPOSED TEST COMPACTION METHOD

Based on the above observations, given a set of test sequences there is a large number of possible subsequences of interest. The compaction problem at hand is to select among them a set that preserves the fault coverage of the original sequences and, at the same time, minimizes the total number of vectors. The proposed test sequence compaction method for asynchronous circuits takes place in two phases. In the first phase, each test sequence is processed and a number of alternative subsequences are generated, wherein SIC vectors are combined into MIC vectors. In the second phase, a set of subsequences of minimal total length is selected from the expanded set of subsequences generated in the first phase.

A. Compacting SIC Sequences Using MIC Vectors

The process of replacing SIC subsequences with MIC vectors entails finding which SIC vectors can be skipped. Skipping one or more SIC vectors may lead to several consequences. The best case is when no hazards are invoked and all faults originally detected are still detected. In this case, the SIC vectors can be summarily skipped. The worst case is when hazards are invoked and the circuit is led to an undetermined state, hence many –if not all– faults detected originally cannot be detected any longer. In this case, the new test sequence is, typically, ineffective and should be discarded since it does not assist test compaction. In other cases, skipping some SIC vectors does not influence the final state of the circuit, but some originally detected faults may not be activated and/or observed, so they are not detected any longer. In this case, both the original and the compacted sequences are kept, since they are both candidates for being selected in the final set of subsequences of minimal length during the second phase of the algorithm.

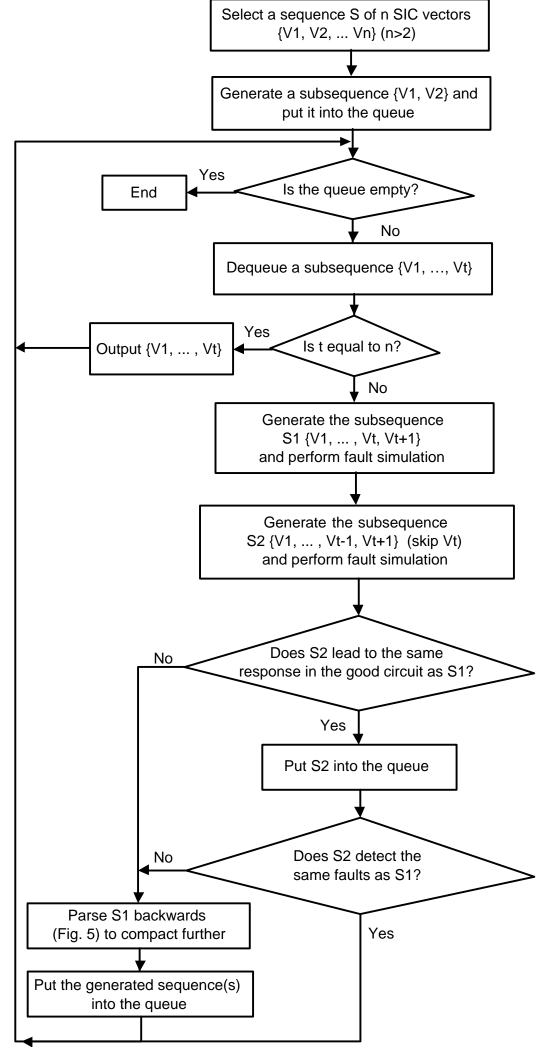

Fig. 3. Flowchart of Forward Compaction

The most straightforward method to compact a SIC sequence with MIC vectors would be to try all possible subsequences generated by skipping any possible combination of SIC vectors. Given a SIC sequence of n vectors, the first vector can typically not be skipped, since it is an initialization vector. The last vector can also not be skipped, since otherwise the sequence would be incomplete. Therefore, the total number of possible subsequences is 2^{n-2}, which is exponential in the length of the sequence. To avoid examining all possible subsequences, which would be impossible for a long test sequence, we propose an approximation algorithm which is illustrated in Fig. 3.

Given a test sequence $S = \{V_1, V_2, ..., V_n\}, n > 2$, the proposed algorithm first considers skipping V_2 by comparing the output/state of the good circuit for subsequence $S_2 = \{V_1, V3\}$ and subsequence

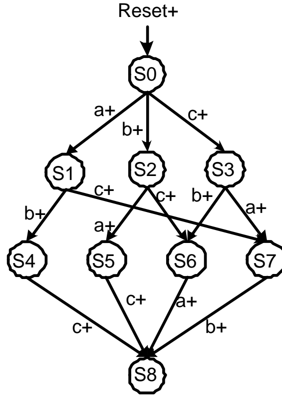

Fig. 4. An Example State Diagram

$S_1 = \{V_1, V_2, V_3\}$. If they are different, V_2 is not skipped and S_2 is discarded, otherwise, S_2 is kept. In the latter case, fault simulation is performed for S_1 and S_2 to see if S_2 can detect the same faults as S_1. If so, S_1 is discarded since S_2 is shorter and can completely replace S_1. Otherwise, S_1 is also kept. Subsequently, the algorithm extends each stored subsequence by one vector and considers skipping the second-to-last vector using the same procedure. This process is repeated until all stored subsequences include the last vector in S.

When a subsequence whose second-to-last vector is not skipped is stored into the queue, a backward parsing process takes place to further skip consecutive SIC vectors. The need for this backward parsing process is illustrated in the following example. Consider the partial state diagram of an asynchronous circuit, shown in Fig. 4. As may be observed, regardless of the relative rising order of signals a, b, and c, the circuit will eventually reach state $S8$. Assume that the circuit is implemented such that when it is at state $S0$ and it receives a MIC vector in which all of a, b, and c rise, it goes to state $S8$, therefore, the three SIC vectors can be compacted by this MIC vector. However, if the original subsequence is $a+$, $b+$, and $c+$, the first two vectors cannot be compacted during the forward parsing phase of the proposed method, since the circuit does not reach the same state with the input $a+$, $b+$ as with the input $b+$, $a+$. Therefore, after $b+$ and $c+$ are compacted and the next vector cannot be compacted, the proposed method uses the backward parsing phase to compact $a+$.

The flowchart of the backward phase is illustrated in Fig. 5. When the second-to-last vector cannot be skipped during the forward parsing procedure, and before the subsequence is stored into the queue, the backward parsing procedure is invoked. If the second-to-last vector is MIC, which means that some SIC vectors must have been skipped, the algorithm tries to further skip the third-to-last vector. If the new sequence and the original sequence generate the same output/state on the good circuit, the new sequence replaces the original one. However, if the new sequence cannot detect the same faults as the original one, the latter is also kept, since it may still be a potential contributor to the best compaction result yielded by the second phase of compaction. This procedure is repeated until no more consecutive SIC vectors can be skipped. After the backward parsing phase, all the resulting subsequences are stored into the queue, and the forward parsing procedure resumes.

B. Selecting Among Independent Test Sequences

The first phase of the algorithm yields a number of independent test sequences and the faults that each of them covers. The rest of the problem is to select among them the set of minimal length that detects all faults. This instance of test compaction was first formulated in [6], where it is shown to be NP-hard and an approximate solution is computed through Genetic Algorithms. While significant levels of compaction within reasonable time are experimentally observed, no indication of proximity to the optimal solution is provided

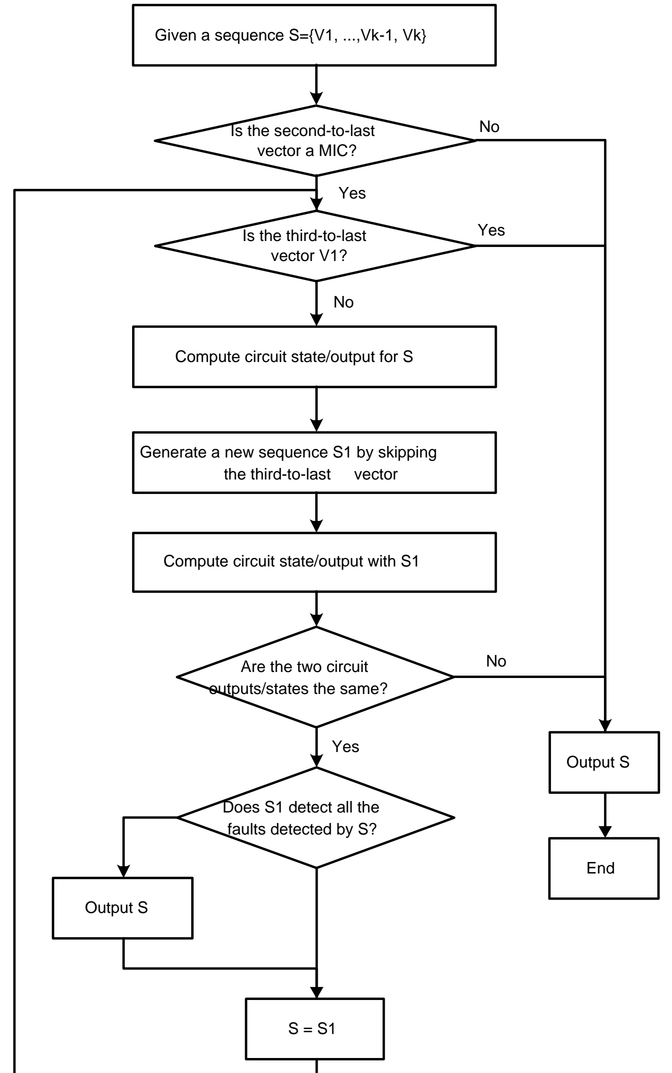

Fig. 5. Flowchart of Backward Compaction

through this method. An alternative is described in [7], wherein an *exact* method (i.e. not an approximation algorithm) that computes the optimal solution using a set of problem-size reduction rules and a branch-and-bound algorithm is proposed. Even though this algorithm works well for most benchmarks, it also lacks any provable, sub-exponential running time guarantee.

SPIN-PAC addresses these issues by employing the method proposed in [8]. The problem is modelled as an integer linear program (ILP), the polynomial time reduction rules of [7] are applied, and the remainder problem is approximated through linear program relaxation and randomized rounding [9].

IV. Experimental Results

SPIN-PAC has been developed in C, based on the fault simulation engine of SPIN-SIM [2] and on *lpsolve* [10]. 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-PAC on a set of Speed-Independent circuits synthesized by Petrify [11]. Experiments were performed on a workstation with dual Xeon 1.7GHz processors and 1 gigabyte of RAM. For each benchmark, we generated a test sequence for each fault using the deterministic test generation tool SPIN-TEST [4]. Then, SPIN-PAC was applied to compact these test sequences. The results are illustrated in Table II. The name of each benchmark circuit is listed in the first column and the number of faults in each circuit is listed in the sec-

TABLE II
TEST COMPACTION RESULTS

Circuit Name	No. of Faults	No. of Test Sequences	Total Length of Tests	Percentage of Sequences Compacted Using MIC vectors (%)	No. of Test Sequences After Compaction	Total Length of Tests After Compaction	Compaction Rate (%)	CPU Time (ms)
alloc-outbound	41	41	95	34.2%	3	11	88.4%	17
chu133	31	31	60	19.4%	3	8	86.7%	33
chu150	33	33	68	18.2%	2	6	91.2%	18
converta	34	34	80	11.8%	2	10	87.5%	18
dff	24	24	85	8.3%	1	7	91.8%	50
ebergen	44	44	108	9.1%	3	17	84.3%	101
half	15	15	24	0%	2	5	79.2%	18
hazard	32	32	88	34.4%	5	19	78.4%	17
mp-forward-pkt	34	34	66	8.8%	4	11	83.3%	17
mr1	87	87	256	27.6%	4	26	89.8%	1134
nak-pa	48	48	96	10.4%	4	12	87.5%	17
nowick	28	28	66	50.0%	2	8	87.9%	18
ram-read-sbuf	55	55	115	40.0%	4	13	88.7%	68
rcv-setup	25	25	54	36.0%	3	10	81.5%	17
rpdft	34	34	91	50.0%	5	14	84.6%	33
sbuf-ram-write	69	69	215	60.9%	6	21	90.2%	318
sbuf-send-ctl	56	56	139	19.6%	3	14	89.9%	101
seq4	63	63	168	14.3%	4	22	86.9%	268
vbe6a	56	56	123	5.4%	2	9	92.7%	234
Average				24.1%			86.9%	

ond column. Note that this number only includes the faults after *g*-equivalent [12] collapsing and undetectable faults are excluded. The third column presents the total number of test sequences generated by SPIN-TEST to test every fault in each circuit, and the fourth column lists the total number of test vectors in these sequences. The fifth column presents the percentage of test sequences for each circuit wherein SIC vectors can be combined into MIC vectors during the first phase of SPIN-PAC. For some benchmark circuits such as *nowick*, *rpdft*, and *sbuf-ram-write*, over half of the SIC test sequences can be compacted with MIC vectors, but for other circuits such as *half* and *vbe6a*, only very few test sequences can be compacted with MIC vectors. On average, about 24.1% of the test sequences can be compacted with MIC vectors across all benchmark circuits. The total number of test sequences and test vectors after the second phase of our method are listed in the sixth and seventh column, respectively. The eighth column presents the compaction rate for each benchmark. As can be observed, a very high compaction rate in the range of 78–93% and with an average of 86.9% is achieved across all benchmark circuits. Finally, the total time that SPIN-PAC spent on each benchmark is also reported in the ninth column of the table.

V. CONCLUSION

Existing ATPG methods for Speed-Independent circuits focus on minimizing test generation time and maximizing fault coverage. SPIN-PAC, the static test compaction method proposed in this work, complements these methods by post-processing the generated tests and minimizing their length, while maintaining the attained fault coverage. It employs a combination of two techniques that build upon the particularities of test vectors generated for Speed-Independent circuits. More specifically, it reduces the number of test vectors by combining sequences of SIC vectors into MIC pairs and by selecting among independent test sequences in order to prune or eliminate some of them. Given the exponential search space of the problem, SPIN-PAC combines efficient heuristics to approximate the optimal solution. Experimental results on benchmark circuits demonstrate that very high rates of compaction are achieved through the proposed method.

REFERENCES

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

[2] F. Shi and Y. Makris, "Spin-sim: Logic and fault simulation for speed-independent circuits," in *Proceedings of International Test Conference*, 2004, pp. 597–606.

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

[4] F. Shi and Y. Makris, "Spin-test: Automatic test pattern generation for speed-independent circuits," in *International Conference on Computer Aided Design*, 2004.

[5] 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.

[6] F. Corno, P. Prinetto, M. Rebaudegno, and M. Sonza Reorda, "New static compacion techniques of test sequences for sequential circuits," in *European Design and Test Conference*, 1997, pp. 37–43.

[7] M. Dimopoulos and P. Linardis, "Accelerating the compaction of test sequences in sequential circuits through problem size reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1443–1449, 2003.

[8] P. Drineas and Y. Makris, "Independent test sequence compaction through integer programming," in *21st International Conference on Computer Design*, 2003, pp. 380–386.

[9] P. Raghavan and C. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[10] M. Berkelaar, "Linear programming solver," Available from `http://www.cs.sunysb.edu/algorith/implement/lpsolve/implement.shtml`.

[11] 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.

[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.