Testing Redundant Asynchronous Circuits by Variable Phase Splitting

Luciano Lavagno*
Dip. di Elettronica
Politecnico di Torino
10129 Torino, Italy

Michael Kishinevsky†
Dept. of Computer Science
Technical University of Denmark
DK-2800 Lyngby, Denmark

Antonio Lioy
Dip. di Informatica
Politecnico di Torino
10129 Torino, Italy

Abstract

An approach for stuck-at-i and delay-fault testing of redundant circuits without modifying the logic *is proposed. The only requirement is the ability to control both phases of each variable* independent of each other. *The circuit becomes fully testable under very weak assumptions, equivalent to freedom from Single Cube Containment in two-level form. The main existing methods for asynchronous circuit synthesis are demonstrated to satisfy the assumptions and then are testable using the methodology. Heuristics to improve the approach include partial scan and non-scan testing.*

1 Introduction

A practical application of asynchronous circuits until now has been almost entirely limited to "niches" such as asynchronous buses. This is certainly due to the difficulty of introducing a completely new design methodology. Automated design aids can help in this respect, but another major obstacle remained until now. No digital circuit design methodology can be used in practice unless it also provides means for *testing* the circuit after manufacture, to certify that it functions according to the specification under a wide range of operating conditions.

Testing asynchronous circuits is definitely a difficult problem, due to two main reasons: (1) The absence of a global synchronization signal does not allow a direct application to asynchronous designs of the testing techniques developed for synchronous sequential circuits; (2) All known asynchronous design methodologies ensure correct *operation* (i.e., avoid spurious pulses on circuit outputs, known as *hazards*) using some level of *redundancy*, i.e., by sacrificing testability.

This difficulty remains even though at least one class of asynchronous circuits, *speed-independent* circuits, is self-checking with respect to stuck-at-i faults on the outputs of the gates [15, 1, 11]. This self-checking property is due to the fact that a stuck-at-i fault can be considered as an infinite delay, hence a circuit whose operation does not depend on that delay will halt in the presence of that fault. The class of faults for which this result holds unfortunately is limited to stuck-at-i faults on the *outputs* of logic gates. This can hardly be considered a satisfactory model of physical faults. The problem of single stuck-at-i faults on gate *inputs* was tackled in [11] by introducing *test inputs*, that may often be unacceptable in large circuits.

A design for testability technique for asynchronous circuits with bounded delays was developed in [9], based on *logic level* transformations on the circuit, that could be proved to: (1) preserve hazard-freeness, if the circuit was initially hazard-free or preserve the ability to eliminate hazards by delay padding and (2) ensure both stuck-at-i testing and path-delay-fault testing using a full-scan approach. This approach had the severe limitation that testing redundant gates required introducing additional test inputs to the circuit, with resulting increase in size, delay and complexity of the test procedure. More recently, Roncken *et al.* ([13]) developed a methodology for testing delay-insensitive circuits built from *handshake components*. The underlying implementation methodology trades off efficiency for simplicity and correctness, by using basic handshake components that are more complex than the "standard" gates used in logic design.

Our paper, on the other hand, opens a new avenue to asynchronous circuit testing that does not require a specific state encoding technique or particular logic transformations. It is based on the idea that if we partition the circuit into a set of combinational logic blocks connected by memory elements (either the native elements, or added D-latches as in the previous approaches), then *full stuck-at-i* and *delay-fault* testability can be achieved under *very weak conditions* by being able to *drive both phases of each combinational logic input independently*. This *de facto* transforms the circuit into a *unate* circuit (see Section 2.1) whose testability properties are excellent ([7]).

The significance of this result is two-fold: (1) It allows to test a circuit using both the most common (stuck-at-i) and the most stringent (hazard free robust path delay) fault models. (2) It can be applied to most (probably all) known asynchronous circuit design methodologies *without requiring any change in the logic implementation*, because the class of circuits implemented using those methodologies automatically satisfies the requirements for being tested by variable phase splitting. The only modification is the use of scan memory elements instead of standard memory ele-

*This work has been supported by MURST under 40% project "VLSI Architectures"

†This work has been supported by The Danish Technical Research Council and ACID-WG (Esprit Basic Research Working Group 7225).

ments (e.g. *S/R* or C-elements).

The paper is organized as follows. Section 2 introduces some notation and previous results that are useful in the rest of the paper. Section 3 develops the theoretical background and the practical techniques to achieve the full testability result. Section 4 demonstrates that most methodologies for asynchronous circuit synthesis satisfy the assumptions that are required to guarantee full stuck-at-i and delay-fault testability. Section 5 describes a preliminary set of experimental results performed on a set of benchmark asynchronous circuits.

2 Preliminaries

2.1 Logic Functions

An *incompletely specified single-output logic function* f of n input variables is a mapping $f : \{0,1\}^n \rightarrow \{0,1,-\}$. Each element of $\{0,1\}^n$ is a *vertex*. The sets of vertices where f evaluates to 1, 0 and - are called *on-set*, *off-set* and *dc-set* respectively. The function is *completely specified* if the dc-set is empty.

A *literal* is either a variable,x_i, or its complement, $\overline{x_i}$. A *cube* is a Boolean product of literals. A *cover* F is a Boolean sum of cubes. A cube c' *covers* another cube c'', if c'' contains all the literals of c', e.g. $a\bar{b}$ covers $a\bar{b}c$. A cube is an *implicant* of a logic function f if it does not cover any off-set vertex of f. A *cover* F of a logic function f is a set of implicants of f such that each on-set vertex of f is covered by at least one cube. Each cover F corresponds to a *unique completely specified* logic function, denoted by $B(F)$.

An implicant of f is *prime* if it is not covered by any other single implicant of f. A cover of a function f is *prime* if all its cubes are prime implicants of f. A *cube* c of a cover F of a function f is *redundant* if $F - \{c\}$ is a cover of f. Note that the testing terminology uses *redundant* in a more general sense, meaning any connection in a circuit that can be replaced with a constant without altering the *logic* function computed by it. A cover F is *minimal with respect to Single Cube Containment* (SCC) if there exists no pair of cubes $c', c'' \in F$ such that c' covers c''.

A function f is *monotone increasing (decreasing)* in a variable x_i if $f(x_i = 0, \beta) = 1 \Rightarrow f(x_i = 1, \beta) = 1$ ($f(x_i = 1, \beta) = 0 \Rightarrow f(x_i = 0, \beta) = 0$) for all $\beta \in \{0,1\}^{n-1}$. It is *unate* in x_i if it is either monotone increasing or monotone decreasing in x_i. A *cover* F is *unate* in a variable x_i if variable x_i appears in only one phase (i.e., either x_i or $\overline{x_i}$) in its cubes.

The following result (Proposition 3.3.7 of [2]) forms the basis of our testing methodology. Section 3 will generalize it to allow some optimizations in the implementation of the scan flip-flops.

Proposition 2.1 *Let F be a unate cover and P be the set of all primes of $B(F)$, the logic function specified by F. Then $P \subseteq F$. If, in addition, F is minimal with respect to* Single Cube Containment, *then $F = P$, and F is the unique minimum cover.*

2.2 Testing

In a gate-level combinational circuit any detectable stuck-at-i fault can always be detected with a single pattern, named a *test*, which belongs to the *test set* of the fault.

Testing for a delay fault cannot be accomplished by a single pattern, rather a *test sequence* S of two patterns is needed, to setup one path from inputs to one output, and to propagate a change along this path. If S is valid under arbitrary delays and is not invalidated by hazard or races, then it is named a *robust* test sequence. Circuits exist which are not robustly delay-fault testable ([4]).

When dealing with CMOS circuits a particular fault model needs attention: the transistor *stuck-open*, which affects the circuit by permanently switching one transistor off. Although this class of faults is static in nature, nonetheless it requires a sequence of patterns to be tested because it generates memory conditions which usually transform a combinational circuit into a sequential one. All the considerations about robust delay fault test apply nearly unchanged to the testing of stuck-open faults. An important relation between delay-fault and stuck-open testing is given in [7, 4]: a prime and irredundant two-level circuit implementing a unate cover is automatically robustly path-delay fault testable as well as stuck-open fault testable.

Testing of sequential circuits is an even harder problem, due to the added complexity of time dependence. While various attempts have been made to directly generate tests for synchronous sequential circuits, industrial practice relies on *scan techniques*.

2.3 Asynchronous Circuits

An *asynchronous circuit* is an arbitrary interconnection of logic gates $Z = \{z_1, \ldots, z_m\}$ and input nodes $X = \{x_1, \ldots, x_k\}$, with each gate input connected to strictly one gate output or one input node and with no two gate outputs tied together. A logic gate can be considered as an interconnection of a Boolean function evaluator and delay elements. Types of function evaluators available, delay models and input-output disciplines are major factors which influence both a design technique and properties of the designed asynchronous circuits.

Asynchronous circuits are by definition sensitive to all signal transitions, whether they are intentional (part of the specification) or not (then they are called *hazards*). In the framework of speed-independent design a hazard-free behavior is captured by the notion of semi-modularity. A circuit is *semi-modular* if no gate whose output is scheduled to change can be disabled during circuit operation (otherwise a spurious pulse may appear on the output of the gate).

3 Testing by variable phase splitting

This section explains the main theoretical foundations of our testing approach for asynchronous circuits. The basic idea is that *implementations of unate functions are easily testable* for most fault models. For example, any two-level circuit implementing a unate function is automatically prime and irredundant if it is free from Single Cube Containment. Hence it is testable for all stuck-at-i faults and robustly testable for all path-delay faults [7].

So, rather than modifying the circuit to make it testable, thus risking to introduce hazards during its normal operation, we will modify its *inputs* to make it *unate*. This can be done for two-level circuits by treating the positive and negative phase of each input variable as *separate entities*. That is, instead of interpreting variable $\overline{x}$ as the complement of x (i.e., driving it with a *not* gate with input x), we can consider it as another primary input to the circuit. This input is *normally* driven to be the complement of x but can, for test purposes, also assume the same value as x. The resulting circuit depends only on the positive phase of each variable, and hence is unate.

The following result is a simple corollary of Proposition 3.3.7 of [2]:

Theorem 3.1 *Let C be any cover free from Single Cube Containment. Let X be the set of Boolean variables on which C is defined. Let $\phi : \{x, \overline{x} : x \in X\} \rightarrow X \times \{', ''\}$, such that $\phi(x) = x'$ and $\phi(\overline{x}) = x''$. Let $\phi(C)$ be the cover obtained from C by applying ϕ to every literal in it.*

Then $\phi(C)$ is a prime and irredundant cover for the corresponding completely specified logic function $B(\phi(C))$.

We can show that in order to test the circuit, we will only need three of the possible four configurations of the pair of variables x' and x''. So we will be able to use a modification of *S/R* flip-flops to "split the phase" of the circuit input variables, even though such flip-flops can never produce both outputs at 1 (or at 0, depending on whether they are implemented with cross-coupled *nor* or *nand* gates).

Given a two-level cover C as above, let us define an incompletely specified logic function $B_{1,1}(\phi(C))$ as follows: (1) The dc-set of $B_{1,1}(\phi(C))$ is the set of all vertices such that there exists at least one pair of literals x', x'' both with value 1; (2) The on-set of $B_{1,1}(\phi(C))$ is the same as the on-set of $B(\phi(C))$ *minus* the vertices that already belong to the dc-set of $B_{1,1}(\phi(C))$; (3) The off-set of $B_{1,1}(\phi(C))$ is the same as the off-set of $B(\phi(C))$ *minus* the vertices that already belong to the dc-set of $B_{1,1}(\phi(C))$.

Then, the following theorem holds:

Theorem 3.2 *Let C be any cover free from Single Cube Containment. Let X be the set of Boolean variables on which C is defined. Let $\phi(C)$ be the cover obtained from C by applying ϕ (as in Theorem 3.1) to every literal in it.*

Then $\phi(C)$ is a prime and irredundant cover for the incompletely specified logic function $B_{1,1}(\phi(C))$.

The proof of this theorem is given in [10].

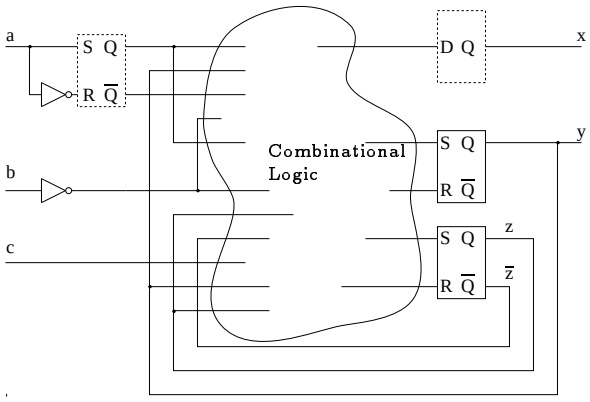

Figure 1: Testing architecture

A dual theorem can also be proved valid for the incompletely specified logic function $B_{0,0}(\phi(C))$ for *nand*-based *S/R* flip-flops, which is defined dually to $B_{1,1}(\phi(C))$.

The importance of these theorems is that the two-level prime and irredundant unate circuit obtained by variable phase splitting is fully testable even when restricting the input space available for choosing a test.

3.1 Practical considerations and optimizations

A practical application of the basic theory requires the development of asynchronous scan flip-flops and of heuristics that can reduce the need for such flip-flops.

The overall circuit architecture assumed by our design for testability methodology is described schematically in Figure 1. In the figure, a, b and c are primary inputs, x and y are primary outputs (of which y is also used as a feedback variable for the combinational logic), and z is an internal signal used only for feedback. Note that: (1) Primary inputs used in both phases (e.g., a) may need to pass through phase splitters. Those splitters are not necessary, for example: if the input comes from another sub-circuit that produces split-phase outputs on the same chip or if the input comes from outside the chip and hence is controllable for test purposes. In this case it may suffice to transform the *not* used to invert the input into a transparent inverting D-latch that can store the negative phase value while the positive phase is driven by the test machine. (2) Primary outputs that are not directly observable (i.e., going only to other sub-circuits on the same chip) may require a standard scan D-latch, held in transparent mode during normal operation.

We will assume that the selected scan flip-flops are *fully testable* for the selected class of faults. In [10] a simple example is given to demonstrate the feasibility of such a scan flip-flop. The latches never need to produce the invalid state (Theorem 3.2). Therefore, all their other states can be regarded as combinational outputs (because can be directly obtained by setting the appropriate values on the S and R inputs). It follows that the latch becomes a combinational element for testing purposes and we can thus limit ourselves to *scan only a minimum number of latches, namely those*

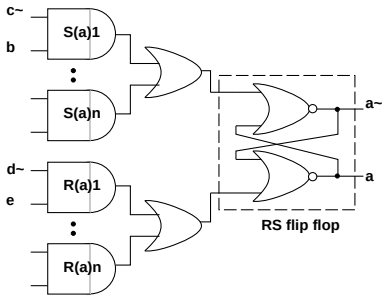

Figure 2: Standard *S/R*-implementation structure

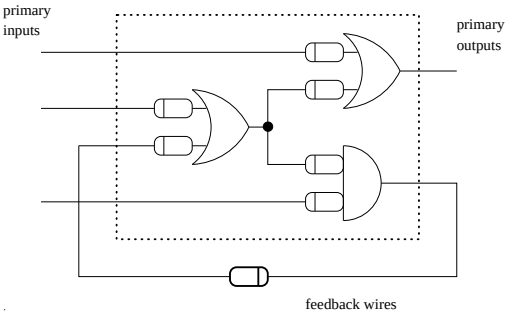

Figure 3: Example of a Huffman circuit

breaking the global feedback paths. This is conceptually similar to perform a *partial scan*, as it has already been proposed to test synchronous sequential circuits.

4 Applications

This section describes how the testing methodology described above can be applied to the main known asynchronous circuit synthesis algorithms. We basically need to show, for each case, that the logic implementation produced by the algorithm automatically satisfies the conditions of Theorem 3.1.

4.1 Unbounded Gate-delay Circuits

In [6] a *standard* S/R-*implementation* was suggested, based on two-level combinational logic and an *S/R* flip-flop (Figure 2), where each *and* gate implements a *region function* consisting of a single cube. Each region function $S_a(i)$ and $R_a(i)$ corresponds to one rising or falling transition of signal a in the initial specification of the circuit.

It was proved in [6] that if each region function $S_a(i)$ and $R_a(i)$ obeys the *monotonous cover requirement*, then the standard *S/R*-implementation is *semi-modular*.

For any semi-modular circuit C_1 we can always derive an *equivalent semi-modular standard* S/R-*implementation* C_2 [6]. Here by equivalence we mean equivalence of externally observable signal transition sequences.

The last statement provides a basis for a proof of the following result on testability of semi-modular circuits [10].

Theorem 4.1 *For any semi-modular circuit an equivalent semi-modular circuit can be constructed, which is fully testable for stuck-at-i faults, and fully hazard-free-robustly testable for path-delay faults and transistor stuck-open faults.*

4.2 Bounded Wire-delay Circuits

In this section we will examine three main classes of asynchronous circuits, together with the associated synthesis algorithms: (1) Huffman circuits, where a block of combinational logic implements the next state and output functions derived from a Flow Table specification, and feedback wires implement the state (see, e.g., Figure 3 and [5], [14]). (2) Burst Mode Finite State Machine (FSM) circuits [12, 16]. (3) Bounded-delay circuits synthesized from a Signal Transition Graph specification, where a block of combinational logic implements the next state function of each state and output function, and feedback wires implement the state (see, e.g., Figure 5 and [8]).

Huffman circuits. The synthesis algorithm described by Unger ([14]) implements each next state and output function using *all its prime implicants*. Prime implicants by definition do not contain each other, so a two-level implementation satisfies the conditions of Theorem 3.1. A straightforward application of our methodology would require cutting all feedback wires and splitting all input signals with scan D-latches (see, e.g., Section 3.1).

The implementation can also be optimized as a *multi-level* circuit, using the transformations that Unger showed to preserve the ability to eliminate hazards. These transformations, applying the distributive, associative and De Morgan's laws, have also been shown ([3]) to preserve both stuck-at-i and robust path delay fault testability. So the optimized circuit will also be testable (at least) for those two models using our strategy.

Burst Mode Machines. Similar considerations apply to the synthesis algorithm for self-clocked and 3-D circuits presented in [12] and [16]. In this case, the specification is a Burst Mode FSM. Burst Mode FSMs restrict the class of allowed FSMs and of *allowed environment behaviors* with respect to [14] to prove the *completeness* of the algorithm.

In a Burst Mode FSM, inputs and outputs are constrained to change in *bursts*, that is sets of signals that can change in any order. When all the signals in an input burst have changed value, the FSM changes state and produces an output burst. The environment is not allowed to change the inputs again until the output burst is completed.

The synthesis procedure uses a constrained two-level minimization framework, described more in detail in [12]. The final cover C implementing the next state and output functions must satisfy a set of conditions that allow us to prove the following result [10].

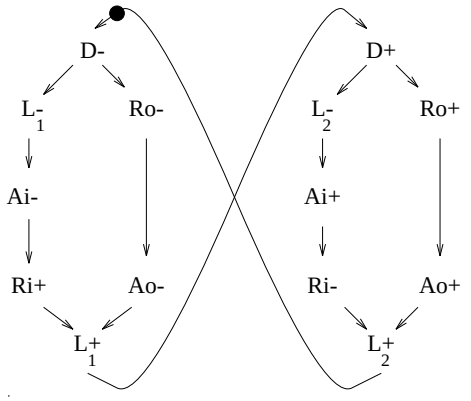

Figure 4: Example of a Signal Transition Graph

Theorem 4.2 *Any circuit constructed using the algorithms described in [16] and [12] can be made fully testable for stuck-at-i faults, and fully hazard-free-robustly testable for path-delay faults and transistor stuck-open faults.*

Bounded-delay circuits implementing Signal transition Graphs. A Signal Transition Graph (STG) is an interpreted Petri net formalizing the concept of timing diagram. See, for example, Figure 4, where x^+ and x^- represent rising and falling transitions of signal x respectively, and arcs represent causality relations between transitions.

Various methods for STG synthesis have been described in the literature. All methods share a common underlying implementation hypothesis, that each output signal is implemented by a combinational logic block and can be fed back to implement the circuit state. Here we are mainly concerned with the method described in [8] for bounded wire-delay circuits, because strong theoretical results are known on its underlying circuit structure.

Signal Ai corresponding to one of the outputs of the box of Figure 4 can be implemented without hazards as a prime redundant two-level cover $Ai = D\overline{L} + D\overline{Ri} + L\overline{Ri}$, as shown in Figure 5.(a). The redundant cube $D\overline{Ri}$ is included in the implementation for preserving hazard-free operation. A stuck-at-1 fault on the output of the gate $D\overline{Ri}$, for example, is not testable without introducing test inputs, therefore, as shown in [9], to test this implementation two additional inputs $test1$ and $test2$ should be used.

Figure 5.(b) shows an implementation of signal Ai with signal L being split into L' and L''. According to the Theorems 3.1 and 3.2 the cover $Ai = DL'' + D\overline{Ri} + L'\overline{Ri}$ is prime and irredundant and can be tested without additional testing inputs.

We can claim the following result:

Theorem 4.3 *Any circuit constructed using the algorithms described in [8] can be made fully testable for stuck-at-i faults, and fully hazard-free-robustly testable for path-delay faults and transistor stuck-open faults.*

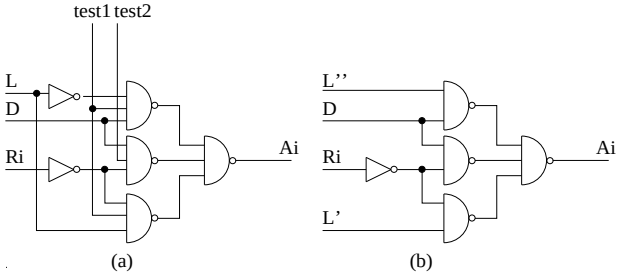

Figure 5: Testable implementation of Figure 4

5 Experimental Results

To test the effectiveness of our proposed methodology, we tried it on a standard set of asynchronous circuit specifications. The results are reported in Table 1. The column labeled "literals" contains the number of literals in a factored form representation of a circuit implementation. The column labeled "untestable paths" contains the number of paths that are not robustly path delay fault testable in a two-level hazard-free implementation of the specification (obtained using the approach described in [8]). The column labeled "split signals" gives the number of signals that need to be split to ensure full *robust path delay fault testability* using a *partial scan* approach. The column labeled "total signals" gives the number of primary inputs and primary outputs of the circuit (the latter coincide with feedback signals in the chosen implementation). Only circuits with untestable paths are included in the table. The line labeled "total 1" gives total counts for these circuits. The line labeled "total 2" gives total counts for all circuits in the benchmark which we checked, including completely path-delay testable circuits.

The *greedy* algorithm that we used for this experiment tries to ensure full testability by splitting each input signal in turn. Theorem 3.1 ensures that full testability can always be achieved in this way. The order of splitting is heuristically chosen by considering signals that are at the head of untestable paths first, and non-unate signals next. Signals that give the best reduction in the number of untestable paths are tried first within each of those two groups. The number of split signals seems reasonably low even using this simple-minded heuristic. The CPU times for synthesis, testability analysis and split variable selection are at most a few seconds on a DEC5000 machine.

Unfortunately a precise comparison with other results from the literature is almost impossible. Some approaches, as discussed in Section 1, do not guarantee full testability even for stuck-at-i faults. Estimating the overhead for approaches that use test inputs to ensure full testability ([11, 9]) is also very difficult without a detailed explanation of exactly when and how these test inputs are added. Generally speaking, we can assume that such inputs are implemented with scan-like flip-flops, like in our proposed methodology, because the overhead of adding one I/O pad for each test input would be unacceptable. A major differ-

circuit	literals	paths		signals	
		untestable	total	split	total
chu150	12	2	15	1	6
converta	26	2	30	1	6
dc	20	5	27	2	6
dlatch	6	2	7	1	4
ebergen	16	4	18	1	5
fifo	6	2	7	1	4
hazard	12	4	14	1	5
qr42	16	4	18	1	5
rpdft	10	2	10	2	6
wrdatab	42	2	48	1	11
dff	10	4	14	1	5
future	26	2	30	1	8
isend	52	2	62	1	11
total 1	254	37	300	15	82
total 2	878	37	1061	15	287

Table 1: Number of split input variables

ence with these approaches is that we can give a tight and reasonably low bound on the number of scan flip-flops: the number of primary inputs, primary outputs and feedback signals. On the other hand, we can assume that the number of test inputs required by the above mentioned approaches is approximately proportional to the number of redundant gates, which in general can be (much) larger. Finally, the approach of [13] uses an implementation methodology that is so different from our setting (being based on handshake components rather than basic gates), to make a direct comparison totally impossible.

6 Conclusions and Future Work

This paper shows that full testability can be achieved for the circuits produced using the main existing asynchronous synthesis methods. The result applies to *stuck-at-i, delay and stuck-open* fault models, and hence covers both the most widely used and the most stringent test methods.

Full testability is achieved *without modification to the logic circuit*, even if it is non-prime or redundant. The testing architecture requires only to use *special scan flip-flops* that can drive *both phases of each signal separately*. Even though the results about the testability of unate functions were already known, their application to asynchronous circuit testing is, to the best of our knowledge, new. New is also a more general result that allows to use only three out of the possible four combinations of values for the split variables, in order to reduce the cost of the scan flip-flops.

We also prove that the circuits synthesized using the best known automated design techniques for asynchronous circuits automatically satisfy the conditions required to ensure full testability using this method.

References

[1] P. A. Beerel and T. H-Y. Meng. Semi-modularity and self-diagnostic asynchronous control circuits. In *Proceedings of the Conference on Advanced Research in VLSI*, March 1991.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[3] M. J. Bryan, S. Devadas, and K. Keutzer. Testability-Preserving Circuit Transformations. In *Proceedings of the International Conference on Computer-Aided Design*, November 1990.

[4] S. Devadas and K. Keutzer. Synthesis of robust delay-fault testable circuits: Theory. *IEEE Transactions on Computer-Aided Design*, 11:87–101, January 1992.

[5] D. A. Huffman. The synthesis of sequential switching circuits. *J. Franklin Institute*, 257:161–190,275–303, March 1954.

[6] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the Design Automation Conference*, 1994.

[7] S. Kundu and S. M. Reddy. On the design of robust testable CMOS combinational logic circuits. In *Proceedings of the International Conference on Fault Tolerant Computing Systems*, pages 220–225, 1988.

[8] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proceedings of the Design Automation Conference*, June 1991.

[9] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis for testability techniques for asynchronous circuits. In *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[10] L. Lavagno, M. Kishinevsky, and A.Lioy. Testing redundant asynchronous circuits. Technical Report ID-TR: 1993-124, Technical University of Denmark, October 1993.

[11] A. J. Martin and P. J. Hazewindus. Testing delay-insensitive circuits. In *Proceedings of the Conference on Advanced Research in VLSI*, March 1991.

[12] S. M. Nowick and D. L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. In *Proceedings of the International Conference on Computer-Aided Design*, November 1992.

[13] M. Roncken and R. Saeijs. Linear test times for delay-insensitive circuits: a compilation strategy. In *proceedings of the Working Conference on Asynchronous Design Methodologies*, March 1993.

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

[15] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, L. Y. Rosenblum, A. R. Taubin, and B. S. Tzirlin. *Self-timed Control of Concurrent Processes*. Kluwer Academic Publisher, 1990. (Russian edition: 1986).

[16] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous state machines. In *Proceedings of the International Conference on Computer-Aided Design*, November 1992.