Research Article

A Discrete Event System Approach to Online Testing of Speed Independent Circuits

P. K. Biswal, K. Mishra, S. Biswas, and H. K. Kapoor

Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati 781 039, India

Correspondence should be addressed to S. Biswas; santoshbiswas402@yahoo.com

Received 23 November 2014; Revised 30 March 2015; Accepted 30 March 2015

Academic Editor: Marcelo Lubaszewski

Copyright © 2015 P. K. Biswal et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the increase in soft failures in deep submicron ICs, online testing is becoming an integral part of design for testability. Some techniques for online testing of asynchronous circuits are proposed in the literature, which involves development of a checker that verifies the correctness of the protocol. This checker involves Mutex blocks making its area overhead quite high. In this paper, we have adapted the Theory of Fault Detection and Diagnosis available in the literature on Discrete Event Systems to online testing of speed independent asynchronous circuits. The scheme involves development of a state based model of the circuit, under normal and various stuck-at fault conditions, and finally designing state estimators termed as detectors. The detectors monitor the circuit online and determine whether it is functioning in normal/failure mode. The main advantages are nonintrusiveness and low area overheads compared to similar schemes reported in the literature.

1. Introduction

With the advancement of VLSI technology for circuit design, there is a need to monitor the operations of circuits for detecting faults [1]. These needs have increased dramatically in recent times because, with the widespread use of deep submicron technology, there is a rise in the probability of development of faults during operation. Performing tests before operation of a circuit and assuming continued fault-free behaviour may decrease the reliability of operation. In other words, there is a need for *online testing (OLT)* of VLSI circuits, whereupon they are designed to verify, during normal operation, whether their output response conforms to the correct behaviour.

Most of the circuits used in VLSI designs are synchronous. Compared to synchronous circuits, asynchronous designs offer great advantages such as no clock skew problem, low power consumption, and average case performances rather than the worst case performances. Testing asynchronous circuits as compared to synchronous circuits is considered difficult due to the absence of the global clock [2].

OLT has been studied for the last two decades and can be broadly classified into the following main categories:

- (i) Self-checking design.
- (ii) Signature monitoring in FSMs.
- (iii) Duplication.
- (iv) Online BIST.

The approach of self-checking design consists of encoding the circuit outputs using some error detecting code and then checking some code invariant property (e.g., Parity) [3–5]. Some examples are Parity codes [6], m-out-of-n codes [7], and so forth. The area overhead for making circuits self-checkable is usually not high. These techniques, termed as “intrusive OLT methodologies,” require some special properties in the circuit structure to limit the scope of fault propagation. These properties can be achieved by resynthesis and redesign of the original circuit, which may affect the critical paths in the circuit.

Signature monitoring techniques for OLT [8, 9] work by studying the state sequences of the circuit FSM model

during its operation. These schemes detect faults that lead to illegal paths in the control flow graph, that is, paths having transitions which do not exist in the specified FSM. To make the runtime signature of the fault-free circuit FSM different from the one with fault, a *signature invariant property* is forced during FSM synthesis, making the technique intrusive. Further, the state explosion problem in FSM models makes the application of this scheme difficult for practical circuits.

Duplication based OLT technique works by simply replicating the original circuit and comparing the output responses [10]; a fault is detected if the outputs do not match. The major advantage of duplication based scheme is nonintrusivity; however, area overhead is more than double. To address this issue, partial duplication technique is applied [11, 12]. This scheme first generates a complete set of test vectors for all the faults possible, using Automatic Test Pattern Generation (ATPG) algorithms. After that, a subset of faults are selected (based on required coverage) and a subset of test vectors (based on tolerable latency) for the selected faults are taken and synthesized into a circuit which is used for OLT. It may be noted that ATPG algorithms are optimized to generate the minimum number of test vectors that detect all faults. As the scheme applies ATPG algorithms in a reverse philosophy, it becomes prohibitively complex for large circuits.

The technique of designing circuits with additional on-chip logic, which can be used to test proper operation of the circuit before it starts, is called off-line BIST. Off-line BIST resources are now being used for online testing [13–16]. This technique utilizes the idle time of various parts of the circuit to perform online BIST. Idle times are reducing in circuits because of pipelining and parallelism techniques and so online BIST scheme is of limited utility.

Motivation of the Work. From the above discussion we may state that an efficient approach for OLT should have the following metrics:

- (i) The OLT scheme should be nonintrusive. This is the most important constraint as designers meet requirements, like frequency, area, power, and so forth, of the circuit to produce an efficient design and do not want the test engineers to change them.
- (ii) The OLT technique should support well accepted fault models.
- (iii) The scheme should be computationally efficient so that it can handle reasonably large circuits.

Most of the papers cited in the above discussion are for OLT of synchronous circuits and only a few of them [4, 5, 10] are applicable to asynchronous circuits. Now, we elaborate on these three works on OLT of asynchronous circuits and derive motivation of the present work.

Traditionally, double redundancy methods were used for OLT of asynchronous designs [10]. In this scheme, two copies of the same circuit work in parallel and the online tester checks whether they generate the same output. This scheme results in more than 100% area and power overheads. Further, both being the same circuit, they are susceptible to similar nature of failures. The schemes reported in [4, 5] basically

work by checking whether the output of the asynchronous circuit maintains a predefined protocol (i.e., there is no premature or late occurrence of transitions). The checker circuit is implemented using David cells (DCs), Mutual Exclusion (Mutex) elements, C-elements, and logic gates. The checker circuit has two modes: operation-normal mode and self-test mode. In normal mode, the checker is used to detect whether there is any violation in the protocol being executed by the CUT. On the other hand, in self-test mode, the checker is used to detect faults that may occur within the checker itself. Mutex elements (component of asynchronous arbiter) were used to grant exclusive access to the shared DCs between different modes of operation. The area overhead of the Mutex blocks is high, even compared to the original circuit. So, area overhead of the online tester in this case would be much higher than that of the original circuit and even the redundancy based methods. Further, this tester only checks the protocol and so fault coverage or detection latency cannot be guaranteed.

Discrete event system (DES) model based methods are used for failure detection for a wide range of applications because of the simplicity of the model as well as the associated algorithms [17]. A DES is characterized by a discrete state space and some event driven dynamics. Finite state machine (FSM) based model is a simple example of a DES. In the state based DES model, the model is partitioned according to the failure or normal condition. The failure detection problem consists in determining, in finite time after occurrence of the failure, whether the system is traversing through normal or failure subsystem. A fault is detectable by virtue of certain transitions (in the failure states) which are called fault detecting transitions (FD-transitions). FD-transition is a transition of the faulty subsystem, for which there is no corresponding equivalent transition in the normal subsystem. Using the FD-transitions, a DES fault detector is designed, which is a kind of state estimator of the system. For OLT of circuits, the detector is synthesized as a circuit which is executed concurrently with the circuit under test (CUT). Biswas et al. in [18, 19] have developed an OLT scheme for synchronous circuits using the FSM based DES theory, which satisfies most of the metrics mentioned above for an efficient online tester design. In this paper, we aim at using the theory of failure detection of DES models for OLT of SI circuits.

Just like synchronous circuits, the basic FSM framework is also used to model asynchronous circuits with slight modification. In case of synchronous circuits, state changes in the FSM occur only at the active edge of the register clock, irrespective of the time of change of the inputs. On the other hand, in asynchronous circuits, state changes can occur immediately after transition in the inputs. FSM used to model asynchronous circuit is called AFSM [20]. An alternative to AFSM is burst-mode (BM) state machines [20]. AFSM and BM state machine are similar from the modeling perspective; however, in case of BM state machine transitions are labeled with signal changes rather than their explicit values, which is the case in AFSMs. AFSMs and BM state machines assume that first inputs change followed by outputs and finally new state is reached. Due to the strict sequence of signal changes, all asynchronous protocols cannot be

modeled using AFSMs or BM state machines. Extended BM state machines address this modeling issue by allowing some inputs in a burst to change monotonically along a sequence of bursts rather than in a particular burst. Petri net (PN) is widely accepted modeling framework for highly concurrent systems [21]. PN models a system using interface behaviors which are represented by allowed sequence of transitions or traces. The view of an asynchronous circuit as a concurrent system makes PN based models more appropriate than AFSMs and BM state machines for their analysis and synthesis. There are several variants of PNs among which signal transition graph (STG) is generally used to model asynchronous circuits. The major reason is that the STG interprets transitions as signal transitions and specifies circuit behavior by defining casual relations among these signal transitions [22].

In this paper, we aim at using the theory of failure detection of DES model for OLT of SI asynchronous circuits. Several modifications are made in the DES framework used for synchronous circuits [18, 19] when applied for SI circuits. The modifications are as follows:

- (i) We first model SI circuits along with their faults as STGs and then translate them into state graphs. State graphs are similar to FSM based DES models from which FD-transitions can be determined.
- (ii) In case of synchronous circuits, the fault detector is an FSM which detects the occurrence of FD-transitions. A synchronous circuit can be synthesized in a straightforward way from the FSM specification [18, 19] that performs online testing. Why the same design cannot be synthesized as an asynchronous circuit will be discussed in this paper. As the use of synchronous circuit for OLT of asynchronous modules is not desirable, we propose a new technique for detector design which can be synthesized as a SI circuit. The detector is designed as state graph model which is live and has complete state coding (CSC); these properties ensures its synthesizability as a SI circuit.

The paper is organized as follows. In Section 2, we present some definitions and formalisms of the DES framework. Section 3 illustrates DES modeling for a speed independent circuit under normal and stuck-at faults. In Section 4, the DES detector for the SI asynchronous circuit is designed. Synthesizing the DES detector as online tester circuit is also discussed in the same section. Section 5 presents experimental results regarding area overhead and fault coverage of the DES detector based online tester. Also, comparison of area overhead of the proposed approach with other similar schemes is reported. Finally, we conclude in Section 6.

2. DES Modeling Framework: Definitions and Formalisms

A discrete event system (DES) model G is defined as $G = \langle V, X, \mathfrak{F}, X_0 \rangle$ where $V = \{v_1, v_2, \dots, v_n\}$ (In case of modeling SI circuits as DES, the state variables are values of the I/O

signals. So, in this work, we will interchangeably use the terms signal and variable.) is a finite set of discrete variables assuming values from the set $\{0, 1\}$, called the domains of the variables, X is a finite set of states, $\mathfrak{F}$ is a finite set of transitions, and $X_0 \subseteq X$ is the set of initial states. A state x is a mapping of each variable to one of the elements of the domain of the variable. A transition $\tau \in \mathfrak{F}$ from a state x to another state x^+ is an ordered pair $\langle x, x^+ \rangle$, where x is denoted by $initial(\tau)$ and x^+ is denoted as $final(\tau)$.

2.1. Failure Modeling. The failure of the system is modeled by dividing the DES model into submodels and each submodel is used to model the system under normal or failure conditions. To differentiate between the submodels, each state x is assigned a failure label by a status variable S with its domain being equal to $\{N \cup F_1 \cup F_2 \cup \dots \cup F_d\}$, where N is normal status, F_i , $1 \leq i \leq d$, is failure status, and d is the number of possible faults.

Definition 1 (normal G -state). A G -state x is normal if $x(S) = \{N\}$. The set of all normal states is denoted by X_N .

Definition 2 (F_i - G -state). A G -state x is failure state or synonymously an F_i -state, if $F_i \in x(S)$. The set of all F_i -states is denoted by X_{F_i} .

Definition 3 (normal G -transition). A G -transition $\langle x, x^+ \rangle$ is called a normal G -transition if $x, x^+ \in X_N$.

Definition 4 (F_i - G -transition). A G -transition $\langle x, x^+ \rangle$ is called an F_i - G -transition if $x, x^+ \in X_{F_i}$.

Definition 5 (equivalent states). Two states x and y are said to be equivalent, denoted by xEy , if $x|_V = y|_V$ and $x(S) \neq y(S)$.

In other words, two states are said to be equivalent if they have the same values for state variables and different value for status variable.

A transition $\langle x, x^+ \rangle$, where $x(S) \neq x^+(S)$, is called a *failure* transition indicating the first occurrence of some failure in the system. Since failures are assumed to be *permanent*, there is no transition from any state in X_{F_i} to any state in X_N or from any state in X_{F_i} to any state in X_{F_j} .

Definition 6 (equivalent transitions). Two transitions $\tau_1 = \langle x_1, x_1^+ \rangle$ and $\tau_2 = \langle x_2, x_2^+ \rangle$ are equivalent, denoted by $\tau_1 E \tau_2$, if $x_1 E x_2$, $x_1^+ E x_2^+$ and they must associate with the same signal change.

Suppose that there is a transition in failure DES model for which there is no corresponding equivalent transition in normal DES model, then that transition is called failure detecting transition (FD-transition). The failure is detected when the system traverses through the FD-transition. Thus, we can define FD-transition as follows.

Definition 7 (FD-transition). A F_i - G -transition of faulty DES model $\tau' = \langle x', x'^+ \rangle$ is an FD-transition, if there is no G -transition $\tau = \langle x, x^+ \rangle$ in the normal DES model such that $\tau' E \tau$.

The motivation of failure detection using DES model is to find out such FD-transitions and design DES detector using these transitions. In the next section, we discuss how to model SI circuits using DES.

3. DES Model of a Speed Independent Circuit: Normal and Faulty

As already discussed, the first step to design a DES based online tester is to obtain the normal and faulty state based model of the CUT. However, the traditional state based DES paradigm cannot be directly used for modeling SI circuits. So in this case we will start with signal transition graph (STG), which is a type of Petri net based DES, to specify fault-free and faulty conditions. The STGs will be converted into state graphs (similar to FSMs) using the concept presented in [22].

We first discuss fault modelling at the STG level using an example and concepts from [22]. In addition to the models (i.e., faults in transistors of the C-elements) given in [22], we have also modeled stuck-at faults on all wires (i.e., input/output of gates).

3.1. Fault Modeling. The SI asynchronous CUT example being considered to illustrate the proposed scheme is shown in Figure 1 (taken from [22]). Traditionally, synchronous circuits consist of blocks of combinational logic connected with clocked latches or registers, while, in case of SI circuit designs, we basically have logic gates as building blocks with C-elements, which act as storage elements. Transistor level diagram of C-element is shown in Figure 2; logic function of the C-element can be described by the Boolean equation $C = AB + AC' + BC'$, where C is the next state and C' is the old state value [22, 23]. The output of C-element becomes logically high (low) when both the inputs are logically high (low); otherwise it keeps its previous logic value. There are two types of C-elements used in SI circuits: static C-element and dynamic C-element. The static version of C-element promises that the information inside it can be stored for unbounded periods. However, dynamic versions of C-element provide gains in terms of area, power, and delay [23–26]. Since the circuits having high operating speed, low area, and power consumption are preferred in modern days, we have chosen SI circuits with dynamic C-elements instead of static ones.

Figure 3 shows the STG for the CUT being considered. Rising (falling) transitions on signals, indicated by + (–), are shown in the STG. The dark circles along the arcs are called tokens. The token indicates one of possibly a set of signals that enable transition to fire. If all input arcs for a signal transition have tokens then that signal transition is said to be enabled. For example, when signal R_{in} goes high (denoted by $R_{in}+$) and R_{out} goes high (denoted by $R_{out}+$), only then $A_{out}+$ transition can take place. Upon firing $A_{out}+$, a token is placed on each of its outgoing arcs, thus enabling $R_{in}-$. Note that $R_{out}-$ is enabled after $A_{out}+$ and $A_{in}+$.

In this paper, we have considered SI circuits that contain C-elements (we assumed dynamic version) and logic gates. For the logic gates, the most popular fault model is the stuck-at fault model, which is at the gate level. However,

FIGURE 1: Example of speed independent asynchronous CUT.

FIGURE 2: Transistor diagram of dynamic C-element.

for the C-elements stuck-on and stuck-off faults for each transistor are an accepted fault model [22]. So we have chosen a mixed gate/transistor level description for modeling the faults. To illustrate fault modeling at both C-elements and basic gates, we consider the circuit example from [22] which is shown in Figure 1.

3.1.1. STG Based Modeling. In this work, we model single stuck-at faults in the gates and transistors (for the C-elements) and map them to STGs of the circuit. For the analysis, the signals attached to the inputs A and B of the C-elements are also indicated in the gate level circuit diagram of Figure 1. Now, we consider some of these faults (one at a time), analyze their effects, and finally modify the STG to model the faults.

FIGURE 3: Signal transition graph of sample circuit.

FIGURE 4: STG level model of stuck-on fault $n1$ of C2 [22].

Consider the C-element C2 of Figure 1 and refer to transistor level circuit of Figure 2. The C-element C2 has R_{out} and R_{in} as inputs and A_{out} as output. If the transistor $n1$ has a stuck-on fault, this leads to error in the circuit that it needs to wait for only $n2$ to be enabled to generate output. When $n2$ turns on, then a path to ground via $n1$ and $n2$ gets established, which makes $p3$ on and $n3$ off, making C high. So C2 has to wait only for $n2$ (i.e., $R_{in}+$ which corresponds to input B of C2 to become 1) to turn on and change the output. In other words, it has to wait for only $R_{in}+$ (and not also for $R_{out}+$, which is the requirement under normal condition) before it can generate $A_{out}+$. Thus, the fault in $n1$ leads to premature firing of the $A_{out}+$ transition. We represent this by including a token on the arc connecting $R_{out}+$ to $A_{out}+$. Availability of this token will enable $A_{out}+$ to fire as soon as $R_{in}+$ arrives, without waiting for $R_{out}+$. This token is denoted by a “1” shown on the arc in Figure 4.

Now, consider C-element C1 producing output R_{out} , with transistor $p1$ having stuck-on fault. As $p1$ is on, the gate has to wait for $p2$ to turn on before it can change the output. When $p2$ turns on (by virtue of $B = 0$) then there is no path to ground as $n2$ is off, which makes $p3$ off and $n3$ on, making C low. Here, C1 has to wait for the input $B = 0$ to generate $R_{out}-$. Referring to Figure 1, for B to become 0, we need either A_{out} to become 1 (the same as R_{in} becoming 0) or A_{in} to become 1. Thus, as soon as we have either $A_{out}+$ or $A_{in}+$, $R_{out}-$ would fire. It may be noted that, under normal condition, for $R_{out}-$ to fire, we also require $A = 0$, which mandates both $A_{out}+$ and $A_{in}+$. This failure condition is indicated in the STG by adding a “1” to the input arcs of $R_{out}-$, which is shown in Figure 5. To elaborate, Figure 5(a) (Figure 5(b)) shows that $R_{out}-$ can be fired as soon as $A_{in}+$ ($A_{out}+$) fires and does not wait for $A_{out}+$ ($A_{in}+$).

As the third fault, let C1 have stuck-on fault at $n2$. The stuck-on fault at $n2$ enforces the circuit to wait only for $n1$ to

be enabled for generating output $R_{out}+$. As $n1$ is connected to input A, which is logical ORing of $A_{out}-$ and $A_{in}-$, transition $R_{out}+$ can fire after $A_{out}-$ or $A_{in}-$ (without requiring to wait for $R_{in}+$). This premature firing of transition $R_{out}+$ is indicated in the STG by adding a “1” to $R_{in}+$, which is shown in Figure 6.

For the gates, stuck-at-0 and stuck-at-1 faults are considered at their inputs and output nets. Let Line 2 of the AND gate from Figure 1 have a stuck-at-0 fault. This makes Line 6 stuck-at-0 fault. As Line 6 is connected to the B input of the C-element C1, we have transistor $p2$ on and transistor $n2$ off. Note that as $n2$ is always off, there is no path to the ground. So the output R_{out} can never become 1 because $p3$ can never turn on. In other words, we will never have the $R_{out}+$ transition. This is indicated by adding a “0” on the output arcs of $R_{out}+$ in Figure 7.

If Line 9 gets stuck at 1, this will lead to Lines 3 and 5 being stuck at 0, further leading to Line 6 being stuck at 0. As Line 6 is connected to the B input of the C-element, we will have the fault manifestation similar to the case of Line 2 stuck at 0. Now, we consider a stuck-at-0 fault at Line 13. As this line is connected to the B input of the C-element C2, it will lead to output A_{out} never becoming 1. The effect is shown by adding a “0” to the output arcs of $A_{out}+$ in Figure 8.

Now, we consider an example of a redundant fault; that is, no logical difference is observed in the operation of the circuit after fault. An instance of such a fault is $n1$ stuck-on fault in C1. This fault enforces the circuit to wait only for $n2$ to be enabled (i.e., B to be 1) for generating output $R_{out}+$. As $n2$ is connected to input B, which is logical ANDing of $R_{in}+$, $A_{out}-$, and $A_{in}-$, $R_{out}+$ can fire only after three transitions, namely, $R_{in}+$ and $A_{out}-$ and $A_{in}-$ fire. It may be noted that $A_{out}-$ and $A_{in}-$ also imply that input A (connected to $n1$) of C1 is 1, which in turn implies condition for $n1$. As fault and normal condition both imply $n1$ to be on, stuck-on fault at $n1$

FIGURE 5: STG level model of stuck-on fault in $p1$ of $C1$ [22]. FIGURE 6: STG level model of stuck-on fault $n2$ of $C1$ [22].

FIGURE 7: STG level model of stuck-at-0 fault in Line 2.

of $C1$ does not generate any behavioral difference. Obviously, such faults cannot be detected and under the single stuck-at fault assumption do not cause significant reliability issues.

For the fault model considered, the total number of faults in a SI circuit having dynamic C-elements is equal to 12 times the number of C-elements (each C-element consists of 6 transistors and each transistor can have stuck-on and stuck-off faults) plus twice the number of I/O lines of the gates (each line has either stuck-at-0 or stuck-at-1 fault).

So the number of faults in case of the circuit considered in Figure 1 is not too small and listing them all would make a tabular representation long. So a partial list of faults and their effects on the STG is given in Table 1.

3.1.2. State Graph Based Fault Modeling and FD-Transitions. As already discussed, the first step of DES based OLT design is to generate the normal and faulty models. For SI circuits, first the STGs under normal and faulty conditions are

FIGURE 8: STG level model of stuck-at-0 fault in Line 13.

obtained and then converted into state based models. In this subsection, we explain the concept using the example circuit of Figure 1 under normal condition and two faults, namely, stuck-on fault in $n1$ of C2 and stuck-on fault in $p1$ of C1.

The state based DES model for the normal circuit is shown in Figure 9. It may be noted that in the circuit there are 4 I/O signals, namely, R_{in} , R_{out} , A_{in} , and A_{out} . In the DES model corresponding to each signal, there is a discrete variable: $\langle v_1 = R_{in}, v_2 = R_{out}, v_3 = A_{in}, v_4 = A_{out} \rangle$ which assumes values from the set $\{0, 1\}$. The set of states are $X0$ to $X13$ and $X0$ is the initial state. State mappings and transitions are shown in Figure 9; for example, state $X0$ maps variables $\langle R_{in}, R_{out}, A_{in}, A_{out} \rangle$ to $\langle 1, 1, 0, 0 \rangle$. In states $X0$ and $X1$ mappings are $\langle 1, 1, 0, 0 \rangle$ and $\langle 1, 1, 0, 1 \rangle$, respectively. So transition from $X0$ to $X1$ changes A_{out} from 0 to 1; this is indicated by transition $A_{out}+$. Now, if we look at the STG for the normal circuit in Figure 3 we note that $A_{out}+$ can fire if $R_{in}+$ and $R_{out}+$ have a token (i.e., $R_{in} = 1$ and $R_{out} = 1$). In state $X0$ as $R_{in} = 1$ and $R_{out} = 1$, $A_{out}+$ can fire. Similarly, the whole DES can be constructed.

The state based DES model for the circuit under $n1$ stuck-on fault in C2 is shown in Figure 10. The set of states are $X'0$ to $X'13$ and $X'0$ is the initial state. The transitions and state mapping are shown in the figure. As discussed in the previous subsection, $n1$ stuck-on fault in C2 results in premature firing of $A_{out}+$ (i.e., it need not wait for $R_{out}+$ and can fire only if $R_{in}+$ holds). If we observe the failure DES model in Figure 10, we note that there are two dotted transitions, which correspond to the failure condition, that is, premature firing of $A_{out}+$. One dotted transition is between $X'11$ and $X'6$. It may be noted that in $X'11$ we have $R_{in} = 1$, $R_{out} = 0$, $A_{in} = 1$, and $A_{out} = 0$, where even though $R_{out}+$ is not enabled, because $R_{in}+$ is enabled, $A_{out}+$ fires. A similar premature firing of $A_{out}+$ occurs between $X'13$ and

TABLE 1: A partial list of faults and their effects on STG.

Fault type	Effect of fault on the transitions
Transistor $n1$ stuck-on fault for C1	$R_{out}+$ can only be fired after $R_{in}+$, $A_{out}-$, and $A_{in}-$ (the same as normal condition)
Transistor $n2$ stuck-on fault for C1	$R_{out}+$ can be fired after either $A_{out}-$ or $A_{in}-$
Transistor $p1$ stuck-on fault for C1	$R_{out}-$ can be fired after either $A_{out}+$ or $A_{in}+$
Transistor $p2$ stuck-on fault for C1	$R_{out}-$ can only be fired after $A_{out}+$ and $A_{in}+$
Transistor $n1$ stuck-on fault for C2	$A_{out}+$ can be fired after $R_{in}+$
Transistor $n2$ stuck-on fault for C2	$A_{out}+$ can be fired after $R_{out}+$
Transistor $p1$ stuck-on fault for C2	$A_{out}-$ can be fired after $R_{in}-$
Transistor $p2$ stuck-on fault for C2	$A_{out}-$ can be fired after $R_{out}-$
Transistor $n1$ stuck-off fault for C1	All arcs from $R_{out}+$ are always 0
Transistor $n2$ stuck-off fault for C1	All arcs from $R_{out}+$ are always 0
Transistor $p1$ stuck-off fault for C1	All arcs from $R_{out}-$ are always 0
Transistor $p2$ stuck-off fault for C1	All arcs from $R_{out}-$ are always 0
Transistor $n1$ stuck-off fault for C2	All arcs from $A_{out}+$ are always 0
Transistor $n2$ stuck-off fault for C2	All arcs from $A_{out}+$ are always 0
Transistor $p1$ stuck-off fault for C2	All arcs from $A_{out}-$ are always 0
Transistor $p2$ stuck-off fault for C2	All arcs from $A_{out}-$ are always 0
Transistor $n3$ stuck-on fault for C1	All arcs from $R_{out}+$ are always 0
Transistor $p3$ stuck-off fault for C1	All arcs from $R_{out}+$ are always 0
Transistor $p3$ stuck-on fault for C1	All arcs from $R_{out}-$ are always 0
Transistor $n3$ stuck-off fault for C1	All arcs from $R_{out}-$ are always 0
Transistor $p3$ stuck-on fault for C2	All arcs from $A_{out}-$ are always 0
Transistor $n3$ stuck-off fault for C2	All arcs from $A_{out}-$ are always 0
Transistor $n3$ stuck-on fault for C2	All arcs from $A_{out}+$ are always 0
Transistor $p3$ stuck-off fault for C2	All arcs from $A_{out}+$ are always 0
Stuck-at-0 fault at Line 1 or Line 2 or Line 3	All arcs from $R_{out}+$ are always 0
Stuck-at-0 fault at Line 6 or Line 7	All arcs from $R_{out}+$ are always 0
Stuck-at-1 fault at Line 9 or Line 10	All arcs from $R_{out}+$ are always 0

TABLE 1: Continued.

Fault type	Effect of fault on the transitions
Stuck-at-0 fault at Line 4 and Line 5	All arcs from R_{out+} are always 0
Stuck-at-1 fault at Line 4 or Line 5	
Stuck-at-1 fault at Line 6 or Line 7	All arcs from R_{out-} are always 0
Stuck-at-0 fault at Line 9 or Line 10	
Stuck-at-0 fault at Line 8	All arcs from R_{out+} are always 0 and all arcs from A_{out+} are always 0
Stuck-at-1 fault at Line 8	All arcs from R_{out-} are always 0 and all arcs from A_{out-} are always 0
Stuck-at-0 at Line 11	All arcs from R_{out-} are always 0 and all arcs from A_{out+} are always 0
Stuck-at-1 fault at Line 11	All arcs from R_{out+} are always 0 and all arcs from A_{out-} are always 0
Stuck-at-0 fault at Line 12 or Line 13	All arcs from A_{out+} are always 0
Stuck-at-1 fault at Line 12 or Line 13	All arcs from A_{out-} are always 0

FIGURE 9: DES model for normal circuit (STG of Figure 3).

$X'8$. So, for the fault $n1$ stuck-on fault in C2, there are two FD-transitions, namely, $\langle X'11, X'6 \rangle$ and $\langle X'13, X'8 \rangle$.

The state based DES model for the circuit under $p1$ stuck-on fault in C1 is shown in Figure 11. The set of states are $X''0$ to $X''13$ and $X''0$ is the initial state. As discussed in the previous subsection, $p1$ stuck-on fault in C1 results in premature firing of R_{out-} triggered by either A_{in+} or A_{out+} . This is captured by the dotted transitions $\langle X''2, X''11 \rangle$ and $\langle X''1, X''8 \rangle$ in failure DES model in Figure 11. The transition $\langle X''2, X''11 \rangle$ represents firing of R_{out-} by A_{in+} in spite of A_{out+} not being

FIGURE 10: DES model for circuit with $n1$ stuck-on fault in C2 (STG of Figure 4). FIGURE 11: DES model for circuit with $p1$ stuck-on fault in C1 (STGs of Figure 5).

enabled and the transition $\langle X''1, X''8 \rangle$ represents firing of R_{out-} by A_{out+} in spite of A_{in+} not being enabled. Thus, for this fault there are two FD-transitions, namely, $\langle X''2, X''11 \rangle$ and $\langle X''1, X''8 \rangle$.

The set of FD-transitions is shown in Table 2.

TABLE 2: FD-transitions.

FD-transitions	Start state	Final state	Effect
1	0001	0101	Premature fire of R_{out}^+ before R_{in}^+ (by A_{in}^-)
2	0000	0100	Premature fire of R_{out}^+ before R_{in}^+ (by A_{in}^-)
3	1110	1010	Premature fire of R_{out}^- before A_{out}^+ (by A_{in}^+)
4	1010	1011	Premature fire of A_{out}^+ before R_{out}^+ (by R_{in}^+)
5	1000	1001	Premature fire of A_{out}^+ before R_{out}^+ (by R_{in}^+)
6	0111	0110	Premature fire of A_{out}^- before R_{out}^- (by R_{in}^-)
7	0101	0100	Premature fire of A_{out}^- before R_{out}^- (by R_{in}^-)
8	1011	1010	Premature fire of A_{out}^- before R_{in}^- (by R_{out}^-)
9	1001	1000	Premature fire of A_{out}^- before R_{in}^- (by R_{out}^-)
10	1101	1001	Premature fire of R_{out}^- before A_{in}^+ (by A_{out}^+)

In the next section, we will discuss the procedure for design of the DES detector from FD-transitions and its synthesis as a SI circuit.

4. DES Detector Based Online Tester

A DES detector is basically a state estimator which predicts whether the Circuit Under Test (CUT) traverses through normal or faulty states/transitions. Broadly speaking, the detector is constructed using transitions which can manifest the fault effects. In other words, such a transition is a faulty transition for which there is no corresponding equivalent normal transition. As already mentioned, we call such transitions failure detecting transitions (i.e., FD-transitions). In the circuit under consideration, comparing the normal (Figure 9) and $n1$ stuck-on fault at C2 DES models (Figure 10), we may note that there are two transitions (dotted) $\langle X'11, X'6 \rangle$ and $\langle X'13, X'8 \rangle$ which manifest the fault effect. Corresponding to these transitions, there are no equivalent transitions in the normal model. These two transitions are FD-transitions for the fault and are used in DES detector construction.

If the CUT is a synchronous circuit then obviously the online tester is also a synchronous circuit that can be designed from the FD-transitions using straightforward FSM synthesis philosophy [18, 19]. The detector FSM has three classes of states, namely, initial, intermediate, and final. The detector measures the I/O signals of the CUT (i.e., variables) to determine whether the following happens.

On startup the detector is in its initial state and it checks whether the CUT is in the initial state of any FD-transition. For example, if we consider only two faults in the circuit under consideration, stuck-on fault in $n1$ of C2

and stuck-on fault in $p1$ of C1, then the FD-transition set is $\{\langle X'11, X'6 \rangle, \langle X'13, X'8 \rangle, \langle X''2, X''11 \rangle\}$. So in the initial state the detector checks whether the signals R_{in} , R_{out} , A_{in} , and A_{out} are 1, 0, 1, and 0 or 1, 0, 0, and 0 or 1, 1, and 0. If so, the detector moves to an intermediate state (in the next clock edge) corresponding to the value matched. For each of the FD-transitions, there is a corresponding intermediate state in the detector. For example, if R_{in} , R_{out} , A_{in} , and A_{out} are measured to be 1, 0, 1, and 0 in the initial state of the detector, the detector moves to the intermediate state corresponding to FD-transition $\langle X'11, X'6 \rangle$. However, if the signals do not match initial state of any FD-transition the detector loops in the initial state. In the intermediate state whether the values of the signals of the CUT match with the final state of the corresponding FD-transition is checked; if so, the fault is detected and the detector moves to the final state and is deadlocked there. Otherwise, it moves to the initial state. Continuing with the example, if the values of R_{in} , R_{out} , A_{in} , and A_{out} are 1, 0, 1, and 1 from the intermediate state, FD-transition $\langle X'11, X'6 \rangle$ is detected (i.e., stuck-on fault in $n1$ of C2) and detector moves to final state in the next clock edge. Otherwise, if R_{in} , R_{out} , A_{in} , and A_{out} are 1, 0, 0, and 0 then CUT is normal and the detector moves to the initial state.

The above mentioned philosophy of constructing the detector and then synthesizing it into a synchronous system is widely used in the DES theory [17] and has been applied for OLT of synchronous circuits [18, 19]. Obviously, if the CUT is an asynchronous circuit and so must be the detector circuit. However, it may be noted that the same philosophy cannot be directly used in the case of SI circuits. The reason is that the FSM of the detector designed above has liveness issue in the final state and has complete state coding (CSC) violations in the intermediate states.

Now we propose a new technique for detector design which can be synthesized as a SI circuit. The detector is designed as state graph model which is live and has complete state encoding, that ensures its synthesizability as a SI circuit. Before formalizing the algorithm for the design of the state graph of the detector, we first introduce the basic philosophy of its working using the examples from the previous section.

An FD-transition in SI circuit design paradigm can be stated as, “under failure, a signal s can change in the presence of signals $y_1, y_2, \dots, y_n$, ($1 \leq n$) which is not possible under normal condition.” For example, in the case of $n1$ of C2 stuck-on fault, $\langle X'11, X'6 \rangle$ is an FD-transition which changes signal A_{out} from 0 to 1 (i.e., A_{out}^+) and the other signals are $R_{in} = 1$, $R_{out} = 0$, and $A_{in} = 1$ (Figure 10). It may be noted that in normal condition for changing A_{out} from 0 to 1 we need $R_{in} = 1$, $R_{out} = 1$, and $A_{in} = 0$ (Figure 9). Comparing with the faulty condition we may state that, “under $n1$ of C2 stuck-on fault, signal A_{out} can change from 0 to 1 in presence of signals $R_{in} = 1$, $R_{out} = 0$, and $A_{in} = 1$ which is not possible under normal condition.” So, to detect whether FD-transition $\langle X'11, X'6 \rangle$ has occurred, the detector needs to tap lines R_{out} , A_{in} , and A_{out} (R_{in} is not required to be monitored as its value is same under normal and faulty case) of the CUT and determine whether A_{out}^+ has fired and at that time whether $R_{out} = 0$ or $A_{in} = 1$ or both; if so, a status output line is made 1. For optimization

of the detector in terms of number of states, tap lines, and so forth, without loss of fault detection capability, we may consider checking $A_{out}+$ in presence of either $R_{out} = 0$ or $A_{in} = 1$ but not both. If we consider the other FD-transition $\langle X'13, X'8 \rangle$ for the fault, it can be detected by checking whether $A_{out}+$ has fired and at that time whether $R_{out} = 0$; R_{in} and A_{in} are not required to be monitored as their values are the same under normal and faulty case. So, it may be stated that to detect the fault by FD-transitions $\langle X'11, X'6 \rangle$ and $\langle X'13, X'8 \rangle$ we need to check whether $A_{out}+$ has fired and at that time whether $R_{out} = 0$. The design and flow of the detector for these two FD-transitions are as follows:

- (1) The state encoding tuple is $\langle R_{out}, A_{out}, Status \rangle$. The initial state of the detector d_0 is encoded as 100. The first two bits represent the complement of $R_{out} = 0$ and $A_{out} = 1$, that is, complement of value of R_{out} in state $X'11$ and complement of change of A_{out} by the FD-transition. The third bit represents *Status* output of the detector which is 0 until FD-transition is detected.
- (2) The detector waits for signal R_{out} to become 0 and if so it moves to state d_1 say, which is encoded as $\langle 000 \rangle$. However, from state d_0 , if A_{out} becomes 1, FD-transition cannot be detected because this is normal situation (state $X0$ in Figure 9) where $A_{out}+$ fires when $R_{out} = 1$; detector moves to state d_5 having encoding $\langle 110 \rangle$. When A_{out} becomes 0 in state d_5 , the detector moves back to d_0 from where it again waits to detect whether the FD-transition occurs.
- (3) From state d_1 the following may happen:

- (a) If R_{out} becomes 1, then FD-transition cannot be detected and so the detector moves back to d_0 .
- (b) If A_{out} becomes 1, then FD-transition and hence fault are detected. The detector moves to state d_2 having encoding $\langle 010 \rangle$. Following that detector makes *Status* output high and moves to state d_3 with encoding $\langle 011 \rangle$.

Once *status* line is 1, that is, fault is detected online, the system should switch to an alternative backup circuit, as under the single stuck-at fault model faults are assumed to be permanent [22]. By that logic, the detector should stop or loop in d_3 indefinitely; however, it would lead to deadlock and is nonsynthesizable as a SI circuit. To avoid this deadlock, a simple modification is made in the detector state graph without affecting the fault detection performance. We wait at state d_3 for any signal to change (i.e., R_{out} from 0 to 1 or A_{out} from 1 to 0) and we move to state d_4 ; let us select R_{out} for this purpose. State encoding of d_4 is $\langle 111 \rangle$. From state d_4 we have a transition to state d_3 on change of R_{out} from 1 to 0.

Figure 12 illustrates the state graph for detecting $n1$ stuck-on fault at C2 by FD-transitions $\langle X'11, X'6 \rangle$ and $\langle X'13, X'8 \rangle$.

FIGURE 12: State graph for detecting FD-transitions $\langle X'11, X'6 \rangle$ and $\langle X'13, X'8 \rangle$.

In similar way we can design detectors for the other FD-transitions shown in Table 2. However, it may be noted that different circuits may be required for the other FD-transitions because merging all FD-transitions in a single detector state graph will lead to CSC problems. As shown in the example above, some FD-transitions can be merged into a single state graph maintaining CSC. Figures 13–16 illustrate the SGs for all the FD-transitions shown in Table 2. Also, the FD-transitions which could be merged are mentioned in the figures.

Before discussing the algorithm for generating the detector for a set of FD-transitions we introduce the notion of compatible FD-transitions.

Definition 8 (compatible FD-transitions). Two FD-transitions $\tau'1 = \langle X'1, X'1^+ \rangle$ and $\tau'2 = \langle X'2, X'2^+ \rangle$ are compatible if the following holds:

- (i) If $s1$ is the signal change by $\tau'1$ and $s2$ is the signal change by $\tau'2$, then $s1$ is the same as $s2$. In other words, signal change by both FD-transitions is the same.
- (ii) Let $V'1 \subseteq V$ (for FD-transition $\tau'1 = \langle X'1, X'1^+ \rangle$) be the set of variables whose values at $X'1$ are different compared to state(s) Xi , where Xi is any state under normal condition (normal DES) from which $s1$ is the signal change. Similarly, the set $V'2 \subseteq V$ is calculated for FD-transition $\tau'2 = \langle X'2, X'2^+ \rangle$. Then, $V'1 \cap V'2 \neq \emptyset$. In other words, there exists at least one signal (i.e., a variable) whose value is the same in initial state of both FD-transitions and that is different compared to the initial state(s) of the corresponding transition(s) under normal condition.

For example, consider two FD-transitions $\langle X'11, X'6 \rangle$ ($= \tau'1$, say) and $\langle X'13, X'8 \rangle$ ($= \tau'2$, say). We calculate $V'1$ for $\tau'1$ as follows. The value of variables at initial state $X'11$ is $\{R_{in} = 1, R_{out} = 0, A_{in} = 1, A_{out} = 0\}$. The signal change for $\tau'1$ is $A_{out}+$. We get two states ($X0 = \{R_{in} = 1, R_{out} = 1, A_{in} = 0, A_{out} = 0\}$ and $X2 = \{R_{in} = 1, R_{out} = 1, A_{in} = 1, A_{out} = 0\}$) in normal condition from which the signal change $A_{out}+$ occurs. Thus, $V'1 = \{R_{out} (= 0)\}$ because R_{out} is the only variable that is different in $X'11$ compared to normal states $X0$ and $X2$. Similarly, we calculate $V'2$ for $\tau'2$ as $V'2 = \{R_{out} (= 0)\}$. Since $V'1 \cap V'2 \neq \emptyset = \{R_{out} (= 0)\}$, thus, these two transitions are compatible and can be merged (as shown in state graph of Figure 12).

Algorithm 9. Algorithm for construction of detectors given the set of FD-transitions.

FIGURE 13: State graph for detecting FD-transitions-Sl. numbers 1 and 2 of Table 2.

FIGURE 14: State graph for detecting FD-transition-Sl. numbers 5 of Table 2.

FIGURE 15: State graph for detecting FD-transitions-Sl. numbers 6 and 7 of Table 2.

FIGURE 16: State graph for detecting FD-transitions-Sl. numbers 8 and 9 of Table 2.

Input. $\mathfrak{F}_{\text{FD}}$ is set of FD-transitions.

Output. Detectors for determining occurrence of FD-transitions are as follows:

- (1) Partition $\mathfrak{F}_{\text{FD}}$ into equivalence classes. Let $\mathfrak{F}_{1\text{FD}}, \mathfrak{F}_{2\text{FD}}, \dots, \mathfrak{F}_{l\text{FD}}$ be the sets generated.
- (2) For each of these sets ($\mathfrak{F}_{i\text{FD}}, (1 \leq i \leq l)$), generate a detector state graph using Step (3) to Step (10).
- (3) s is the signal changed by $\tau' \in \mathfrak{F}_{i\text{FD}}$ and y is any signal whose value is the same in initial states of all $\tau' \in \mathfrak{F}_{i\text{FD}}$. Further, signal y is different in the initial state of the corresponding normal transition which also makes the same change in s .
- (4) Let state encoding of the detector be the tuple $\langle y, s, \text{Status} \rangle$.

- (5) Create the initial state d_0 . The values of the variables in d_0 are as follows: (i) y in the tuple for d_0 is complement of the value of the variable y in $\text{initial}(\tau')$, $\tau' \in \mathfrak{F}_{i\text{FD}}$, (ii) s in the tuple for d_0 is complement of the value of the variable s after its change by $\tau' \in \mathfrak{F}_{i\text{FD}}$, and (iii) Status in d_0 is 0.
- (6) Create state d_1 , with transition from d_0 to d_1 labeled as $y+$ ($y-$) if value of y in d_0 is 0 (1). Also, create a transition from d_1 to d_0 labeled with inverse of the signal change as in transition from d_0 to d_1 . Accordingly, encode state d_1 .
- (7) Create state d_2 , with transition from d_1 to d_2 labeled as $s+$ ($s-$) if value of s in d_1 is 0 (1). Accordingly, encode state d_2 .
- (8) Create state d_3 , with transition from d_2 to d_3 labeled as $\text{Status}+$. Accordingly, encode state d_3 .
- (9) Create state d_4 , with transition from d_3 to d_4 labeled as $y+$ ($y-$) if transition from d_0 to d_1 is $y-$ ($y+$). Add a transition from d_4 to d_3 with inverse of the signal change as in transition from d_3 to d_4 . Accordingly, encode state d_4 .
- (10) Create state d_5 , with transition from d_0 to d_5 labeled as $s+$ ($s-$) if transition from d_1 to d_2 is $s+$ ($s-$). Add a transition from d_5 to d_0 with inverse of the signal change as in transition from d_0 to d_5 . Accordingly, encode state d_4 .

4.1. Circuit Synthesis for DES Detector. It is clear from the construction of the state graphs of detectors that they have complete state coding [27, 28] and are live. So they can be synthesized as SI circuits using C-elements and logic gates by applying standard asynchronous circuit synthesis procedures [29, 30].

Figure 17 illustrates some snapshots regarding the steps of synthesizing the state graph of the detector shown in Figure 14 using CAD tool Petrify [31]; Figure 17(a) is the description of the state graph that is input to Petrify, Figure 17(b) illustrates the output of Petrify showing CSC and no liveness issues, and Figure 17(c) shows the equations obtained from Petrify. The circuit schematic (of the DES detector) that is synthesized for this state is shown in Figure 18.

Now, we explain some details of the Petrify equations. *INORDER* is a keyword of Petrify to represent all I/O signals of the corresponding state graph. *OUTORDER* is the keyword to represent the output signals of the state graph. Each subsequent line (denoted by $[0, 1] \dots$) represents a gate of the circuit in terms of the function it implements. In case of the circuit of Figure 18, *INORDER* = $A_{\text{out}}, R_{\text{out}}, \text{Status}$ and *OUTORDER* = Status . The equations $[0] = A'_{\text{out}} R'_{\text{out}}$ and $[1] = 0$ represent the logic expressions of the internal Gate 4 and Gate 3, respectively. The equation $[\text{Status}] = [1]'([0] + \text{Status}) + \text{Status}[0]$ represents the output of the circuit.

In similar way, all state graphs for the FD-transitions have been synthesized into different circuits. Then, the final DES detector circuit for CUT is constructed by simply ORing

```
.model FN.g
.inputs Aout Rout
.outputs Status
.state graph
d0_110 Rout- d5_100
d5_100 Rout+ d0_110
d0_110 Aout- d1_010
d1_010 Aout+ d0_110
d1_010 Rout- d2_000
d2_000 Status+ d3_001
d3_001 Aout+ d4_101
d4_101 Aout- d3_001
.marking {d0_110}
.end
```

(a) State graph

```
pradeep@pradeep-desktop:~/Desktop/asy work/examples$ ./petrify FN.sg -gc -eqn FN.
eqn
# The STG has CSC.
# File generated by ./petrify 4.2 (compiled 15-Oct-03 at 3:06 PM)
# from <FN.sg> on 25-Apr-14 at 1:09 PM
# CPU time: 0.01 sec -- Host: pradeep-desktop
# 0.00(trav)+0.00(init)+0.00(min)+0.00(enc)+0.00(CSC)+0.00(map)+0.00(regs)+0.0
0(irred)
# The original TS had (before/after minimization) 6/6 states
# Original STG: 3 places, 8 transitions, 13 arcs ( 5 pt + 5 tp + 3 tt)
# Current STG: 5 places, 7 transitions, 20 arcs ( 10 pt + 9 tp + 1 tt)
# It is a Petri net with 2 self-loop places
.model FN
.inputs Aout Rout
.outputs Status
.graph
Rout- p2
Rout+/1 p0 p3
Aout-/2 p3 p1
Aout+/3 p0 p3
Aout-/7 p4
Status+/5 p4
Aout+/6 Aout-/7
p0 Rout+/1 Aout-/2
p1 Aout+/3 Status+/5
p2 Rout+/1 Status+/5
p3 Rout- Aout-/2 Aout+/3
p4 Aout+/6
.marking { p0 p3 }
.end
pradeep@pradeep-desktop:~/Desktop/asy work/examples$
```

(b) Output of Petrify

```
# EON file for model FN
# Generated by ./petrify 4.2 (compiled 15-Oct-03 at 3:06 PM)
# Outputs between brackets "[out]" indicate a feedback to input "out"
# Estimated area = 7.00

INORDER = Aout Rout Status;
OUTORDER = [Status];
[0] = Aout' Rout';
[1] = -0-;
[Status] = [1]' ([0] + Status) + Status [0]; # mappable onto gc
# The initial state is unstable. No reset information generated.
# Signal [1] enabled in the initial state.
```

(c) Output equations of Petrify

FIGURE 17: Screenshot showing the synthesis of DES detector from state graph using Petrify.

FIGURE 18: DES detector circuit for state graph shown in Figure 14.

the outputs of these circuits. The output of the detector circuit becomes high when output of at least one individual detector becomes high, thereby detecting the fault.

5. Experimental Evaluation

To validate the efficacy of the scheme, we analyze the area overhead ratio of the online tester circuit to that of the circuit under test. Further, we also compare the overhead with other techniques reported in the literature. In our experiments, we have considered some standard SI benchmark circuits [32]. Further, for comparison, we have also implemented our scheme on the circuits used in [5].

The algorithm discussed in previous section is used to design a CAD tool OLT-ASYN which generates detectors for

OLT given an asynchronous circuit specification. The design procedure includes the following steps.

- (i) First, the behavior description of the SI circuit is represented using STG.
- (ii) The STG representation of the SI circuit is converted into its corresponding state graph using Petrify.
- (iii) Using Petrify, the state graphs are implemented as SI circuits using generalized C-element.
- (iv) s-a-0 and s-a-1 faults are inserted in all the nets of the gates and stuck-on and stuck-off faults are inserted in the transistors of the C-elements of the circuit (one at a time) and the corresponding faulty state graphs are generated.
- (v) FD-transitions are generated for all the faults using the DES theory.
- (vi) Using Algorithm 9 state graphs of the detector(s) are generated.
- (vii) The state graphs are implemented as SI circuits using generalized C-element implementation using Petrify.

Area overhead ratio of the online tester circuit to that of the circuit under test is computed using the formula

$$\text{Area overhead ratio} = \frac{\text{Combined area of the Detector circuits}}{\text{Area of the CUT}} \quad (1)$$

Area of gates and C-elements are considered in terms of number of transistors used in their CMOS implementation.

TABLE 3: Fault coverage, area overhead ratio, and execution time for the online detector designed using the proposed approach.

CUT	Number of gates	Number of faults	Fault coverage (%)	Area overhead	Execution (sec)*
Circuit 1	3	22	95	0.8	0.12
Circuit 2	5	32	96	1.5	0.23
Circuit 3	5	54	98	1.2	0.34
2 David cells [5]	6	44	94	1.6	0.22
4 David cells [5]	12	88	93	1	0.40
chul72	9	47	96	0.24	0.36
alloc-outbound	15	130	98	0.13	0.42
sbuf-read-ctl	19	152	95	0.12	0.51
sbuf-send-ctl	18	140	94	0.087	0.47
ram-read-sbuf	23	197	93	0.025	0.56

*Executed in AMD Phenom IIX3 710 Processor with 4 GB RAM in Linux OS.

For example, a two-input NAND gate has four transistors (two PMOS and two NMOS). Fault coverage is calculated as ratio of number of faults detected by the tester to the total number of faults.

Table 3 shows the number of gates, number of faults, fault coverage, area overhead ratio, and execution time of the proposed approach for the different SI circuits being considered.

The first three circuits in the table are simple examples whose gate level designs are shown in Figures 19, 20, and 21. The fourth and fifth circuits have been used in [5]. The others are standard asynchronous benchmarks [32] which are complex in terms of area, states, and signal compared to first five circuits in the table.

Broadly speaking, it can be observed from Table 3 that area overhead decreases with the increase of the size of the circuit. In [12], Drineas and Makris have identified that the area overhead ratio for partial replication based OLT for stuck-at faults is approximately $\alpha + 1/\beta$, where α is the fraction of test patterns incorporated in detector design ($\alpha = 1$ when all FD-transitions are incorporated in detector design) and β is the number of state bits required for circuit representation (i.e., proportion to circuit size). In this work, we have taken all possible FD-transitions and the obtained area overhead ratio acts (approximately) in accordance with the fact mentioned above.

From the discussion in the last section regarding design of the detector from the FD-transitions, it may appear that a large number of such detectors may be required for complex circuits. In the worst case, the number of detectors may be equal to the number of FD-transitions. Further, in case of large circuits as the number of nets and C-elements are high the number of FD-transitions may be proportionally large. However, interestingly, the experiments illustrated reverse trends. Large circuits have larger number of possible stuck-at faults; however, many of them are mapped to similar effects and hence the same FD-transitions; this can be observed from Table 1 for the running example. Further, using the principle of compatible of FD-transitions, it was found that multiple FD-transitions fall in the same clusters thereby resulting in the fact that a single detector suffices for more than one FD-transition. To conclude, it was observed that a few detectors

FIGURE 19: Circuit 1.

FIGURE 20: Circuit 2.

are actually required to cover all the faults. To the best of our knowledge, such facts regarding OLT of asynchronous circuits were not reported in the literature.

It may be noted that percentage of fault coverage is more than 95% in average. The number of faults that could not be detected was found to be redundant.

5.1. Mutex Approach to Testing. Now, we will discuss, in brief, the Mutex approach for online testing proposed in [5] and compare its area overhead ratio with our scheme. In [5], the scheme is demonstrated on the following specification of a handshaking protocol:

$$req+ \rightarrow ack+ \rightarrow req- \rightarrow ack- . \quad (2)$$

Online testing is performed using checkers which verify that sequencing of the signals as per the protocol is maintained;

FIGURE 21: Circuit 3.

that is, there is no premature or late firing of signals, no signal is stuck-at-0/stuck-at-1 fault, and so forth. All the protocol signals are used as the input for the checker. The checker has two functionalities, namely, self-checking and online testing of the handshaking protocol. The signal “mode” decides this selection. The block diagram of the checker is given in Figure 22. A part of the checker circuit is shown in Figure 23 which will be used to present the basic working of the checker. The full circuit diagram and functionality can be found in [5]. As shown in Figure 23, there are two Mutex components; one is used for the arbitration between $req-$ and $ack-$, while the other is used to arbitrate between $mode+$, $gal1+$, and $gal1e+$; the details of the signals $gal1+$ and $gal1e+$ are given below.

The initial state is 11 (both req and ack are high). This is guaranteed by the previous state. In the previous cycle of operation, $req+$ occurs before $ack+$, which sets $gr11 = 1$, thus providing the appropriate initial condition for the given cycle. The next signal change should be $req-$ (as per the handshaking protocol). So, once $req-$ occurs, $gal11$ signal is set to 1 (left Mutex), as ack signal is still high. As a result, the checker goes to next state 01 indicating no errors. If there is an error in the protocol under test, then $ack-$ precedes $req-$, and $gal1e$ is set to 1. This moves the checker along the fault branch.

When the mode signal is set to logic 1, the three-input arbiter (right Mutex in Figure 23) is used to arbitrate $mode+$, $gal11+$, and $gal1e+$. This Mutex is used for self-testing of the checker.

The asynchronous circuit under test (which realizes the above handshaking protocol) is implemented using David cells [5]. The partial checker circuit illustrated in Figure 23 basically tests the handshake protocol between a pair of David cells and performs self-checking. Along with the partial circuit shown in Figure 23, there are four David cells (not shown), which together test the handshaking protocol (between a pair of David cells) and self-testing of the checker. Among the two Mutex blocks and four David cells, half of them are used for handshaking protocol and the other half is used for self-testing. The logic gates are shared resources for both types of testing. In [5], both the circuit under test and

FIGURE 22: Block diagram of the checker.

FIGURE 23: A part of the checker circuit [5].

the checker circuitry are implemented using David cells, using the flow of converting a Petri net model to asynchronous circuits based on David cells [33].

5.2. Comparison with the Mutex Approach. The proposed DES based approach for online testing does not involve self-testing of the detector. So, for comparison of area overhead of the proposed scheme with [5], we require only half of the resources used in [5]. Table 4 shows area overhead ratio of the checker (for online testing only) for two circuits implementing handshaking protocols involving two and four David cells, respectively. The table also reports the number of David cells, Mutex elements, and logic gates involved in the checker (for online testing). From Tables 4 and 3 (fourth and fifth circuit), we can deduce that the area overhead requirement for the Mutex method is higher compared to that of the purposed scheme. The advantages of the proposed method over the Mutex approach are as follows:

- (1) The area overhead for the online tester circuit is less as compared to that of the Mutex approach by about 50%.
- (2) There is flexibility to trade off area overhead, by reducing fault coverage depending upon the testability requirements. Such flexibility is not easy to be achieved by the Mutex approach. It may be noted that the proposed scheme verifies that there are no stuck-at faults while the Mutex approach verifies a protocol. In particular, it checks for the correct sequencing of the outputs. Fault coverage can be easily traded off with area overhead in our approach, while it is difficult to achieve something like “partial verification of protocol,” avoiding checking certain incomplete output sequences.

TABLE 4: Area ratio for the Mutex approach.

CUT	Circuits for OLT	Area overhead ratio for Mutex method
2 David cells	2 David cells + 2 Mutex elements + gates	3.3
4 David cells	4 David cells + 4 Mutex elements + gates	4.9

- (3) In the detector of the proposed scheme, there is no dependency on Mutex elements. The Mutex element itself can undergo a metastable state which needs to be handled by the metastability detector, adding to area overhead.

In this work, we could not compare fault coverage of our approach with the Mutex approach. The Mutex approach verifies online whether the output of the CUT follows the specified handshaking protocol. So, the Mutex approach basically follows functional testing. The scheme proposed in our paper works on structural testing and hence fault coverage can be given, while it is not possible for functional testing.

It may be noted that the circuits considered in the Mutex-based OLT scheme [5] were simple. For comparison with our scheme, we have done manual implementation of Mutex-based tester design on those circuits.

6. Conclusion

In the present paper, a method for the online testing of SI circuits has been proposed. We start by obtaining the STG of the SI circuit under test using the tool Petrify. After that, effects of stuck-at faults were modeled in the STGs. The normal and faulty STGs were transformed into state graphs and a DES detector was designed. The detector is capable of determining, online, whether any of the modeled stuck-at faults have occurred in the circuit. Finally, the detector is synthesized as a SI circuit with C-elements which is to be placed on chip. Several circuits were considered as case study and area overhead ratio of the detector was studied for these circuits. Results illustrated that area requirement of the detector of the proposed scheme is less than that of the Mutex approach [5] by about 50% on the average. Apart from this, there are several other advantages of the proposed approach, namely, independence of circuit functionality, nonintrusiveness, liveness, and CSC of the detector to ensure synthesizability and so forth.

The present scheme is applicable only for SI circuits with dynamic implementation of C-elements. As a further direction of research, this technique can be extended for other types of asynchronous circuits like Delay Insensitive (DI) circuits. It is required to verify whether this proposed DES based OLT scheme can be directly applied for DI circuits or some modifications would be needed.

Also, in case of SI circuits, the present scheme can be extended for static C-elements. OLT of static C-elements may be comparatively more complex than that of dynamic

C-element because the dynamic C-element comprises less number of transistors than that of static C-element. Additionally, in the present work, it is found that transistor stuck-off or stuck-on faults lead to premature/nonoccurrence of a transition in the STG. Whether similar fault manifestation remains for static C-elements or there is a change needs to be verified. Clearly further research is required to solve these issues.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

- [1] M. Nicolaidis and Y. Zorian, "On-line testing for VLSI—a compendium of approaches," *Journal of Electronic Testing: Theory and Applications*, vol. 12, no. 1-2, pp. 7–20, 1998.
- [2] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: a survey," *Integration, the VLSI Journal*, vol. 19, no. 3, pp. 111–131, 1995.
- [3] M. Nicolaidis, "Design for soft-error robustness to rescue deep submicron scaling," in *Proceedings of the IEEE International Test Conference*, pp. 1140–1149, IEEE, Washington, DC, USA, October 1998.
- [4] D. Shang, A. Yakovlev, F. P. Burns, F. Xia, and A. Bystrov, "Low-cost online testing of asynchronous handshakes," in *Proceedings of the 11th IEEE European Test Symposium (ETS'06)*, pp. 225–232, IEEE, Southampton, UK, May 2006.
- [5] D. Shang, A. Bystrov, A. Yakovlev, and D. Koppad, "On-line testing of globally asynchronous circuits," in *Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS'05)*, pp. 135–140, July 2005.
- [6] K. De, C. Natarajan, D. Nair, and P. Banerjee, "RSYN: a system for automated synthesis of reliable multilevel circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 2, no. 2, pp. 186–195, 1994.
- [7] W.-F. Chang and C.-W. Wu, "Low-cost modular totally self-checking checker design for m -out-of- n code," *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 815–826, 1999.
- [8] R. Leveugle and G. Saucier, "Concurrent checking in dedicated controllers," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 124–127, October 1989.
- [9] R. Leveugle and G. Saucier, "Optimized synthesis of concurrently checked controllers," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 419–425, 1990.
- [10] T. Verdel and Y. Makris, "Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies," in *Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 345–353, IEEE, 2002.
- [11] P. Drineas and Y. Makris, "Non-intrusive design of concurrently self-testable FSMs," in *Proceedings of the 11th Asian Test Symposium (ATS'02)*, pp. 33–38, November 2002.
- [12] P. Drineas and Y. Makris, "Selective partial replication for concurrent fault-detection in FSMs," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 6, pp. 1729–1737, 2003.
- [13] Y. Balasubrahmanyam, G. L. Chowdary, and T. J. V. S. Subrahmanyam, "A novel low power pattern generation technique

- for concurrent BIST architecture,” *International Journal of Computer Technology and Applications*, vol. 3, no. 2, pp. 561–565, 2012.
- [14] P. Daniel and R. Chandel, “Dynamic self programming architecture for concurrent fault detection,” *International Journal of Research in IT, Management and Engineering*, vol. 2, no. 12, pp. 67–81, 2012.
- [15] I. Voyiatzis, A. Paschalis, D. Gizopoulos, C. Halatsis, F. S. Makri, and M. Hatzimihail, “An input vector monitoring concurrent BIST architecture based on a precomputed test set,” *IEEE Transactions on Computers*, vol. 57, no. 8, pp. 1012–1022, 2008.
- [16] V. Gherman, J. Massas, S. Evain, S. Chevobbe, and Y. Bonhomme, “Error prediction based on concurrent self-test and reduced slack time,” in *Proceedings of the 14th Design, Automation and Test in Europe Conference and Exhibition (DATE '11)*, pp. 1626–1631, March 2011.
- [17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, “Diagnosability of discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555–1575, 1995.
- [18] S. Biswas, S. Mukhopadhyay, and A. Patra, “A formal approach to on-line monitoring of digital VLSI circuits: theory, design and implementation,” *Journal of Electronic Testing*, vol. 21, no. 5, pp. 503–537, 2005.
- [19] S. Biswas, G. Paul, and S. Mukhopadhyay, “Methodology for low power design of on-line testers for digital circuits,” *International Journal of Electronics*, vol. 95, no. 8, pp. 785–797, 2008.
- [20] C. J. Myers, *Asynchronous Circuit Design*, John Wiley & Sons, New York, NY, USA, 2001.
- [21] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “Hardware and petri nets: application to asynchronous circuit design,” in *Application and Theory of Petri Nets 2000*, vol. 1825 of *Lecture Notes in Computer Science*, pp. 1–15, 2000.
- [22] D. Lu and C. Q. Tong, “High level fault modeling of asynchronous circuits,” in *Proceedings of the 13th IEEE VLSI Test Symposium*, pp. 190–195, May 1995.
- [23] M. Shams, J. C. Ebergen, and M. I. Elmasry, “Modeling and comparing CMOS implementations of the C-element,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, pp. 563–567, 1998.
- [24] M. Shams, J. C. Ebergen, and M. I. Elmasry, “A comparison of CMOS implementations of an asynchronous circuits primitive: the C-element,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '96)*, pp. 93–96, Monterey, Calif, USA, August 1996.
- [25] M. Moreira, B. Oliveira, F. Moraes, and N. Calazans, “Impact of C-elements in asynchronous circuits,” in *Proceedings of the 13th International Symposium on Quality Electronic Design (ISQED '12)*, pp. 437–443, March 2012.
- [26] M. T. Moreira, F. G. Moraes, and N. L. Calazans, “Beware the Dynamic C-Element,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 7, pp. 1644–1647, 2014.
- [27] C. N. Liu, “A state variable assignment method for asynchronous sequential switching circuits,” *Journal of the ACM*, vol. 10, no. 2, pp. 209–216, 1963.
- [28] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “A region-based theory for state assignment in speed-independent circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 8, pp. 793–812, 1997.
- [29] J. Gu and R. Puri, “Asynchronous circuit synthesis with Boolean satisfiability,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 961–973, 1995.
- [30] T. Chu, “Automatic synthesis and verification of hazard-free control circuits from asynchronous finite state machine specifications,” in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 407–413, Cambridge, Mass, USA, 1992.
- [31] 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.
- [32] Myers Research Group, *Atacs Online Demo*, Myers Research Group, 1999, <http://www.async.ece.utah.edu/>.
- [33] D. Shang, F. Xia, and A. Yakovlev, “Asynchronous circuit synthesis via direct translation,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '02)*, vol. 3, pp. 369–372, Phoenix-Scottsdale, Ariz, USA, May 2002.