J Electron Test (2007) 23:405–420
DOI 10.1007/s10836-007-5009-3

# Dynamic Fault Diagnosis of Combinational and Sequential Circuits on Reconfigurable Hardware

**Fatih Kocan · Daniel G. Saab**

Received: 15 December 2005 / Accepted: 14 May 2007 / Published online: 13 September 2007
© Springer Science + Business Media, LLC 2007

**Abstract** This article describes an emulation-based method for locating stuck-at faults in combinational and synchronous sequential circuits. The method is based on automatically designing a circuit which implements a closest-match fault location algorithm specialized for the circuit under diagnosis (CUD). This method allows designers to perform dynamic fault location of stuck-at faults in large circuits, and eliminates the need for large storage required by a software-based fault dictionary. In fact, the approach is a pure hardware solution to fault diagnosis. We demonstrate the feasibility of the method in terms of hardware resources and diagnosis time by experimenting with ISCAS85 and ISCAS89 circuits. The emulation-based diagnosis method speeds up the diagnosis process by an order of magnitude compared to the software-based fault diagnosis. This speed-up is important, especially, when the on-line diagnosis of safety–critical systems is of concern.

**Keywords** Dynamic fault diagnosis · Stuck-at faults · Circuits · Gate-level · Emulation · FPGA

*Responsible Editor*: C.-W. Wu

F. Kocan (✉)
Computer Science and Engineering,
Southern Methodist University,
Dallas, TX, USA
e-mail: kocan@engr.smu.edu

D. G. Saab
Electrical Eng. and Computer Science,
Case Western Reserve University,
Cleveland, OH, USA
e-mail: dgs3@cwru.edu

## 1 Introduction

The importance of manufacturing high quality digital circuits cannot be under-emphasized. The most practical way to assure quality is to test fabricated circuits against the specifications and to diagnose the cause and location of failure for replacement of a faulty subcircuit or improvement of the manufacturing process. Most fault diagnosis systems operate on specific circuit and fault models that allow physical entities to be represented on a computer. Circuits are modeled at behavioral, register-transfer, gate, switch, and transistor levels while faults are modeled as stuck-at, bridging, IDDq, delay, etc [5]. This article addresses the diagnosis of single stuck-at faults in circuits modeled at gate level.

Fault diagnosis techniques are classified into two main groups based on the analysis performed: (1) *cause–effect analysis* [5]; (2) *effect–cause analysis* [1, 11]. In *cause–effect analysis*, the logical behaviors of primary outputs in the presence of faults are computed using fault simulation, and stored in a table referred to as *fault dictionary* [20, 25]. Faults are located by comparing the stored behavior to the response of the failing CUD. The location of the fault that closely matches with the behavior of the CUD is reported as the suspect location. It is possible that more than one location can be a suspect. In this case, suspect locations are ranked based on their *closeness* measures to the CUD. Fault dictionaries are classified according to the type of information stored in them. (1) A pass/fail dictionary stores a pass/fail entry for each vector and fault pair. A pass (fail) entry means that the logic values of primary outputs in the fault-free circuit are equal to (different from) those of the faulty-circuit for the applied vector. On other hand, (2) a full fault dictionary stores the logic values of all outputs for each vector and fault pair. In any case, the sizes of these

dictionaries depend on the number of faults, number of outputs, and the size of test vector set. In general, fault dictionaries require large storage. Therefore, compaction and compression techniques are used to reduce their sizes [20, 22, 26]. Storage requirement may be eliminated by re-computing the information stored in the dictionary when needed. Techniques that use this method are referred to as *dynamic fault diagnosis* [22]. The computation cost of dynamic diagnosis is high which limit its usage to the case when only small number of diagnosis is performed [22]. To further reduce the overhead, a technique that mixes dynamic and dictionary based diagnosis is proposed in [20].

In *effect–cause analysis*, faults are located by deducing the location of the defect from the behaviors of the CUD and fault-free circuit. The effect–cause analysis employs an *implicit multiple stuck-at fault model*, and avoids fault enumeration or precomputing faulty responses [2]. Given the tests applied at the primary inputs (PIs), the analysis method justifies all internal line values obtained at the primary outputs (POs). This line-justification process includes forward and backward logic value implications for each vector, called horizontal implications, and also logic value implications across vectors, called vertical implications [2, 11].

Several diagnostic measures are proposed to quantify the accuracy level of locating a fault for a given test set. These metrics are *diagnostic resolution* (DR), *diagnostic power* (DP), *diagnostic power for limit K* (DP-with-K), and *diagnostic expectation* (DE) [26]. DR is the fraction of distinguished fault pairs resulting from the application of the test set, and DP is the fraction of fully distinguished faults [9]. A complete version of DR metric identifies and reports the sizes of the sets of fault equivalence classes [16]. The measure DR is controlled by the size/number of partitioned faults (faults in a partition are pair-wise indistinguishable) and the quality of the test set. DP-with-K is the fraction of all faults in classes of size $K$ or less [26]. DE reports the expected number of candidate faults after the diagnosis [26]. *Diagnostic fault simulation* calculates the values of the diagnostic metrics [12].

Emulation systems are increasingly used in the design, verification, and rapid proto-typing of digital systems [8]. They are also utilized as accelerators in certain application domains such as video and image processing and VLSI/CAD. VLSI/CAD algorithms are accelerated by running them on reconfigurable devices that provide bit-level concurrency in the computations. Stuck-at fault simulation [3, 6, 7, 10, 19], switch level fault simulation [17], emulation-based analysis of soft errors in deep sub-micron circuits [24], dynamic fault diagnosis of stuck-at faults [13, 14], automatic test pattern generation (ATPG) [15, 28], and satisfiability (SAT) [4, 18, 23, 29, 31] are among some of the CAD problems that are accelerated with reconfigurable devices.

This article introduces a reconfigurable-based approach for locating and diagnosing faults in combinational and synchronous sequential circuits. The approach is based on automatically designing a circuit which implements a closest-match-based dynamic diagnosis algorithm specialized for the CUD. The specialized circuit is mapped onto reconfigurable hardware to enable fault simulation and matching algorithms to run at the speed of the reconfigurable device. This approach (1) allows dynamic diagnosis of large circuits, (2) eliminates the need for large storage required by a full fault dictionary, (3) reduces the diagnosis time, (4) preserves the diagnostic resolution of a full dictionary, and (5) offers pure hardware-only solution to the diagnosis problem. Fast diagnosis especially is important for safety–critical systems. Also, since it is a hardware-only solution, it eliminates the usage of disk and processor in the diagnosis process.

This article is organized as follows. In Section 2, we revisit the dynamic diagnosis algorithms presented in the literature for combinational and sequential circuits that are capable of locating both modeled and un-modeled faults. Section 3 covers emulation architectures for combinational and sequential circuit diagnosis algorithms. In Section 4, we approximate the diagnosis times. In Section 5, we assess the performance of the proposed dynamic diagnosis method for ISCAS85 and ISCAS89 in terms of required reconfigurable hardware resources and diagnosis time. In Section 6, we present methods to make the approach scalable. Section 7 demonstrates the extensibility of our approach to other fault models. Finally, we conclude our article and give future directions in Section 8.

## 2 Dynamic Fault Diagnosis

In dynamic fault diagnosis, fault simulation is used repeatedly to produce the faulty circuit response when needed. To locate the cause of the failure, the logical behaviors of the simulated faulty circuit model $(C_f)$ and the CUD are compared. The responses of two combinational circuits, which are subject to the same test vector, are equal if the corresponding primary outputs of the circuits match: a match occurs if two binary values are equal. Two combinational circuits have equivalent logical behavior under a test set $T$ iff the responses of these two circuits fully match for every vector $v \in T$. Similarly, the sequential circuit diagnosis requires simulation of a faulty model of the circuit $(C_f)$ and behavioral comparison of the responses of the CUD and $C_f$. $C_f$ may produce unknown values at its outputs due to incompletely specified state variables for several clock cycles while a CUD always produces binary values at its outputs. Therefore, the primary outputs of the $C_f$ and CUD circuits may match potentially or definitely [21]. A definite match occurs when the compared responses

are equal. A potential match occurs when the response generated by $C_f$ is unknown (X) while the CUD output is binary [5].

Two diagnosis algorithms are applied. The first one is based on fault dropping and the second algorithm is based on the best-matching behavior. The first algorithm eliminates any fault from being a suspect if, for any vector in the test set, the behavior of the outputs of the faulty circuit model $(C_f)$ does not match exactly the behavior of the CUD. This approach is very fast and relies on the premise that the logical behavior of every failure is equivalent to at least one fault for the given test set [22]. In general, this is not the case. Therefore, a diagnosis algorithm based on the best-matching behavior is proposed to locate modeled and non-modeled faults. The best-matching algorithm stores the count of matches between the faulty circuit model and the CUD for each fault [20]. The faults are then ranked from the highest matching to the lowest one. Thus, the higher the match is the more suspicious the fault location. It is expected that the actual location of the failure is in the vicinity of the best matched fault(s).

Next, the diagnosis algorithm for modeled faults is presented. This algorithm is followed by the second version of the diagnosis algorithm, which handles non-modeled faults as well as modeled faults. Both of these algorithms are applicable to combinational and sequential circuits. For sequential circuits, the equality-check operators in these algorithms needs to be modified to handle unknown values and potential match cases.

### 2.1 Dynamic Location of Modeled Faults with Fault Dropping

Algorithm 1 locates the modeled faults in combinational and sequential circuits. This location technique assumes that all faults are modeled and the test set ($T$) detects the modeled faults therefore the location of the failure must *match* at least one of the faults. Since a fault represents a location in the circuit, the fault and its location are used interchangeably in different contexts. This algorithm compares the behaviors of *CUD* and the faulty circuit model ($C_f$). In case the behaviors of $C_f$ and *CUD*, which are subject to the same test vector, does not match exactly for any test vector, $f$ is eliminated from being the suspect fault (i.e., its location cannot cause the failure). This is repeated for every fault. For a total of $F$ faults, there are $F$ different faulty models of the circuit. At the end, the faults, which match behaviorally for all considered input vectors, remain in the possible set of failure locations. In this algorithm, the input patterns are restricted to the set of test vectors ($T$). The worst-case asymptotic run-time of Algo. 1 is $O(T \times F \times G)$.

**Algorithm 1** Dynamic fault diagnosis algorithm for modeled faults

```
R_cud: response of the CUD
R_f : response of the C_f
for each test t ∈ T do
  for each fault f ∈ F do
    C_f ← Inject(C, f)
    R_cud(t) ← Apply(CUD, t)
    R_f(t) ← Emulate(C_f, t)
    if R_cud(t) ≠ R_f(t) then
      F ← F − {f}
    end if
  end for
end for
Return F : the set of suspect faults
```

### 2.2 Dynamic Location of Modeled and Un-modeled Faults

The second diagnosis algorithm that handles un-modeled as well as modeled faults is presented in Algo. 2. This algorithm computes the behavior of the faulty circuit model $(C_f)$ and the CUD for each fault. The primary outputs are compared to compute the number of matches (*Match* variable) and to keep track of the number of vectors with partially matched behavior in *Failed_vector* variable. For sequential circuit diagnosis, mostly, the potential matches are counted as definite matches. *Failed_vector* is incremented whenever the logical behavior of the faulty circuit model $(C_f)$ *partially matches* that of the CUD. When the faulty behavior matches exactly that of the CUD for all test vectors, *Failed_vector* remains zero. In the other case, the variable *Failed_vector* is used to prioritize faults with equal number matches (*Match*): among the two faults with equal number of matches, the one with the lower *Failed_vector* is more suspicious than the other. The routine *Rank* in Algorithm 3 updates the sorted list of $N$ suspicious faults by replacing the least suspicious fault with a more suspicious fault. The worst-case asymptotic run-time of Algo. 2 is $O(T \times F \times G + F \times N)$: $T \times F \times G$ is the fault simulation time and $F \times N$ is the time to maintain a sorted array of $N$ elements.

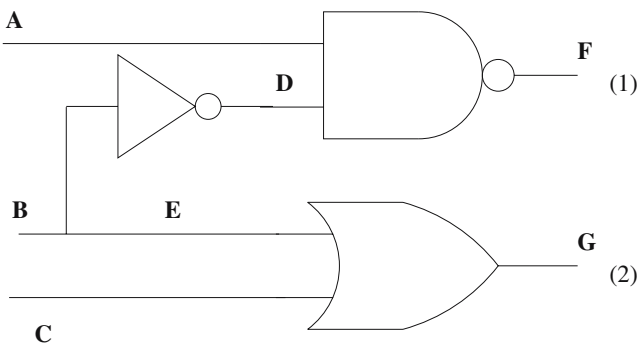


**Fig. 1** Example circuit

**Fig. 2** Faulty responses, matches and failed vectors. **a** Faulty circuits response, **b** matches and failed vectors for each fault

| | T0 | | T5 | | T6 | | | Match | Failed vector |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 1 | 2 | 1 | 2 | | | |
| Φ | 1 | 0 | 0 | 1 | 1 | 1 | Φ | - | - |
| A1 | 0 | 0 | 0 | 1 | 1 | 1 | D1 | 6 | 0 |
| B0 | 1 | 0 | 0 | 1 | 0 | 0 | B0 | 5 | 1 |
| B1 | 1 | 1 | 1 | 1 | 1 | 1 | F0 | 5 | 1 |
| C0 | 1 | 0 | 0 | 0 | 1 | 1 | E0 | 4 | 1 |
| E0 | 1 | 0 | 0 | 1 | 1 | 0 | A1 | 4 | 2 |
| D1 | 1 | 0 | 0 | 1 | 0 | 1 | C0 | 4 | 2 |
| F0 | 0 | 0 | 0 | 1 | 0 | 1 | F1 | 4 | 2 |
| F1 | 1 | 0 | 1 | 1 | 1 | 1 | G1 | 4 | 2 |
| G0 | 1 | 0 | 0 | 0 | 1 | 0 | G0 | 3 | 2 |
| G1 | 1 | 1 | 0 | 1 | 1 | 1 | B1 | 3 | 3 |

**a** **b**

An example circuit is presented in Fig. 1 to show the computation of the number of matches and failed vectors for each modeled fault. This circuit has three inputs and two outputs, stuck-at-0 faults on lines B, C, E, F and G, stuck-at-1 faults on lines A, B, D, F and G. Figure 2a shows the responses of the faulty circuits and fault-free circuit ($\Phi$) for test vectors T0 (000), T5 (101) and T6 (110), and Fig. 2b shows the number of matches and failed vectors for each fault when the circuit has stuck-at-1 fault at line D.

**Algorithm 2** Dynamic fault diagnosis based on matching

```
for each fault f ∈ F do
  Match ← 0
  Failed_vector ← 0
  C_f ← Inject(C, f)
  for each test t ∈ T do
    R_cud(t) ← Apply(CUD, t)
    R_f(t) ← Emulate(C_f)
    D ← number-of-zeros in (R_cud(t) ⊕ R_f(t))
    Match ← Match + D
    if D ≠ O then
      Failed_vector++
    end if
  end for
  Rank(f, Match, Failed_vector)
end for
```

**Algorithm 3** Ranking procedure

```
Require: f: Current fault id
Require: Match: Total number of matches for f
Require: Failed_vector: Total number of partially matched vectors for f
  min ← Match & (T − Failed_vector) // concatenation
  replace ← 0
  for i ← 1; i < N; i++ do
    if Level[i] < min then
      replace ← i
      min ← Level[i]
    end if
  end for
  if replace ≠ 0 then
    ID[replace] ← f
    Level[replace] ← Match & (T − Failed_vector)
  end if
```

In the next section, an architecture is presented for Algorithm 2.

## 3 Dynamic Diagnosis Architectures

The section introduces an architecture for the dynamic diagnosis algorithm customized for the CUD. Figure 3 illustrates the general steps involved in the customization. The mapping program transforms the original circuit $C$ into a new circuit $ALG(C)$, which executes the diagnosis algorithm specifically for $C$. The new circuit $ALG(C)$ is

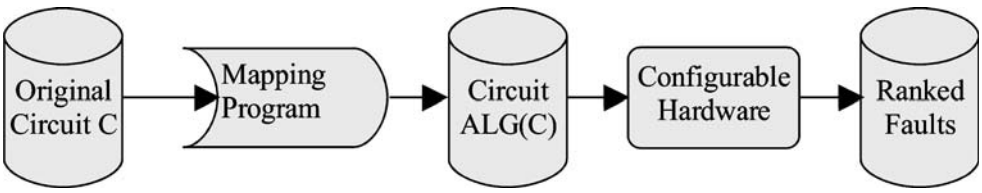


**Fig. 3** A reconfigurable method for dynamic diagnosis

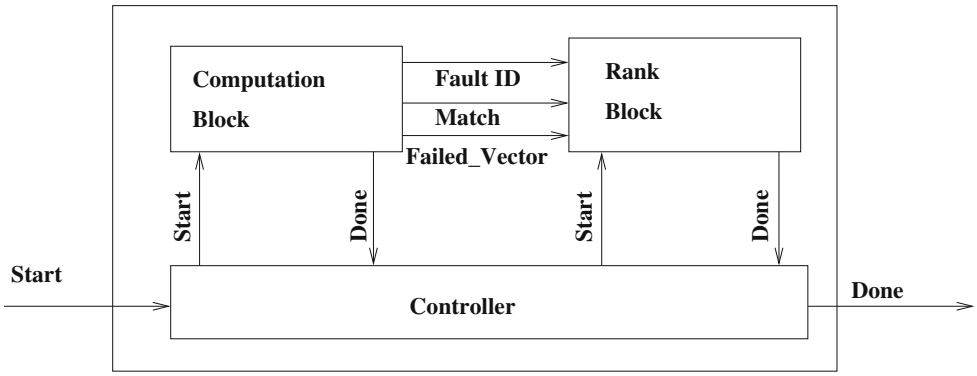


**Fig. 4** The dynamic diagnosis architecture

then mapped onto reconfigurable hardware such as FPGAs for emulation.

The architecture for $ALG(C)$ consists of a data-path and a controller. The data-path is circuit specific and the controller is circuit independent. Thus, the controller can be re-used in the emulation of other circuits. Section 3.1 covers the details of the dynamic diagnosis data-path. The generic data-path is customized for combinational and sequential circuits, respectively in Sections 3.2 and 3.3. In Section 3.4, the fault ranking architecture is presented.

### 3.1 The Data-path of Dynamic Diagnosis Architecture

The detail of the architecture for the dynamic diagnosis algorithm is given in this subsection. Fig. 4 shows the high-level block diagram of the architecture. This architecture consists of three main components: a **Computation** block that implements the fault emulation and response match computation, a **Rank** block that keeps a list of the first $N$ best matching suspicious locations, and a **Controller** block that manages the computation flow. This implementation relies on signals, which are generated in one of these blocks and managed by the controller. Moreover, the controller loads the number of test vectors into a register during initialization, and starts the execution of the diagnosis algorithm for a particular fault by activating the *Start* signal of the Computation block. When the Computation block receives the *Start* signal, it activates a fault and applies the test set to compute the number of matches (*Match*) and the number of failed vectors (*Failed_vector*) for this fault, and forwards these values along with the *ID* of the fault to the Rank block. After that, it sends a *Done* signal to the controller. Consequently, the controller activates the Rank's *Start* signal and waits for the Rank block to update the list of suspicious faults. The Rank block activates its *Done* signal when this process is completed. The controller repeats this process for every fault. Upon completion of all faults, the list of suspicious faults is traversed sequentially and made available at the output of the reconfigurable hardware.

The Computation block is depicted in Fig. 5. This block implements the fault simulation, and computes the number of matches (*Match*) and the number of failed vectors (*Failed_vector*) for each fault. It consists of **Fault activation and injection circuits**, a **Matcher**, a **Partial match tester**, a **Test Vectors RAM** and four registers (**TV**: total number of test vectors, **TVC**: the pointer to the currently applied test vector, **MATCH**: the sum of matches, and **FAILED_VECTOR**: the number of failed vectors).

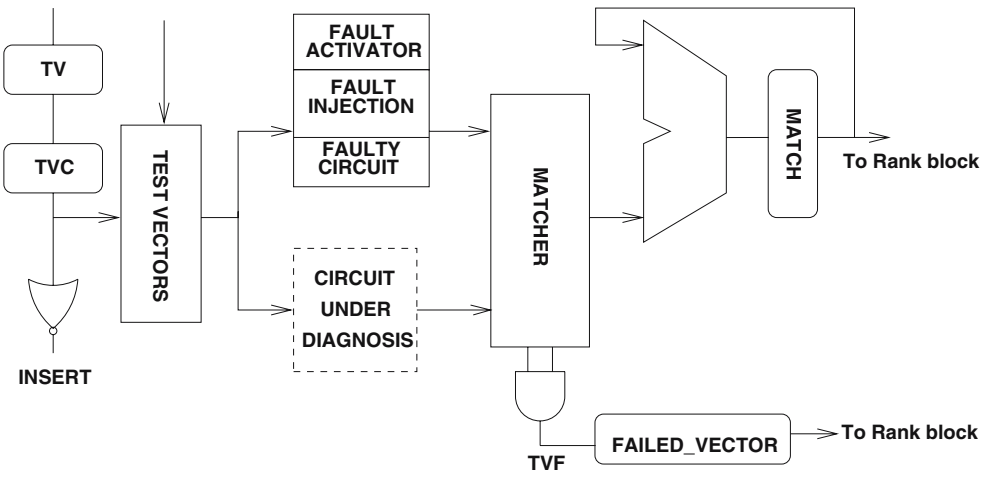


**Fig. 5** The computation block for dynamic diagnosis

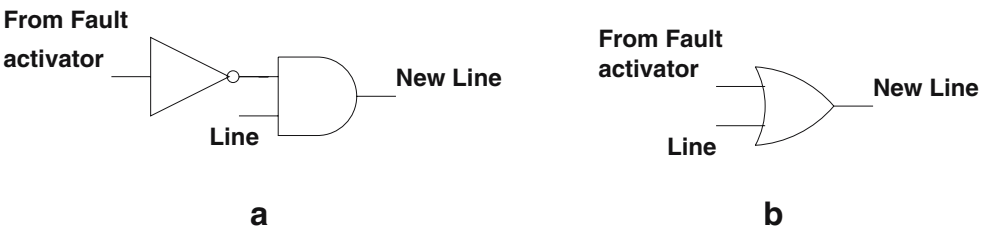


**Fig. 6** Fault injection circuits. **a** 0-Injection circuit. **b** 1-Injection circuit

The Rank and Controller blocks are common for combinational and sequential circuits. The only differences are in the Fault injection, Activation and Faulty circuit blocks. Also, the Matcher circuit needs to be designed to handle potential matches that occur during the emulation of sequential circuits.

### 3.2 Computation Block for Combinational Circuit Emulation

The computational block emulates the faulty combinational circuits and computes the number of matches. The faulty circuits are created by using fault activation and injection circuits. A fault injection circuit mimics the behavior of fault on a line when the fault is activated by a fault activator. The fault injection circuit receives an activation signal from the fault activator circuit. It is similar to the one used in [10]. The fault injection circuit for *Line* stuck-at-0 (stuck-at-1) is shown in Fig. 6a,b. For example, if the stuck-at-0 at *Line* is activated, the *From Fault Activator* signal for *Line* would be high (logic 1) and *New Line* would be always logic 0. Cascading of these two circuits serially enables injecting both stuck-at-0 and stuck-at-1 faults to the same line. Line $L$ in Fig. 7 is an example of cascading.

The Fault Activator is a shift register where each flip-flop (FF) in this register corresponds to a fault. In this representation, when a FF is set to 1 (logic high) the corresponding fault will be injected into the circuit as shown in Fig. 7. In Fig. 7, the circuit for injecting four faults at lines $L$ , $M$ and $N$ ($L$ s-a-0, $L$ s-a-1, $M$ s-a-0 and $N$ s-a-0) is shown. Each of these faults is activated by setting the corresponding FF to 1, which modifies the logic behavior of circuit by fixing the logic value of the fault location to constant 0 (1) for s-a-0 (s-a-1). The fault activation circuit minimizes the number of reconfiguration needed to compute the suspicious location and increases the size of the emulated circuit that depends on the number of faults and fault injection circuit. In addition, one FF is needed in the fault activator circuit to model the inactivation of all faults, i.e. fault-free circuit. So, for $p$ stuck-at-0 and $q$ stuck-at-1 faults, $p+q+1$ FFs and $2\times p+q$ additional gates are needed to emulate $p+q$ faulty circuits.

The Matcher circuit computes the number matches between the primary output responses of the faulty circuit model and the CUD. To do this, both circuits are subject to the same vectors, which are stored in a RAM. For every vector, the number of matches is computed by *XNOR*ing the corresponding primary outputs of CUD and the faulty circuit, and summing the number of 1s in the corresponding

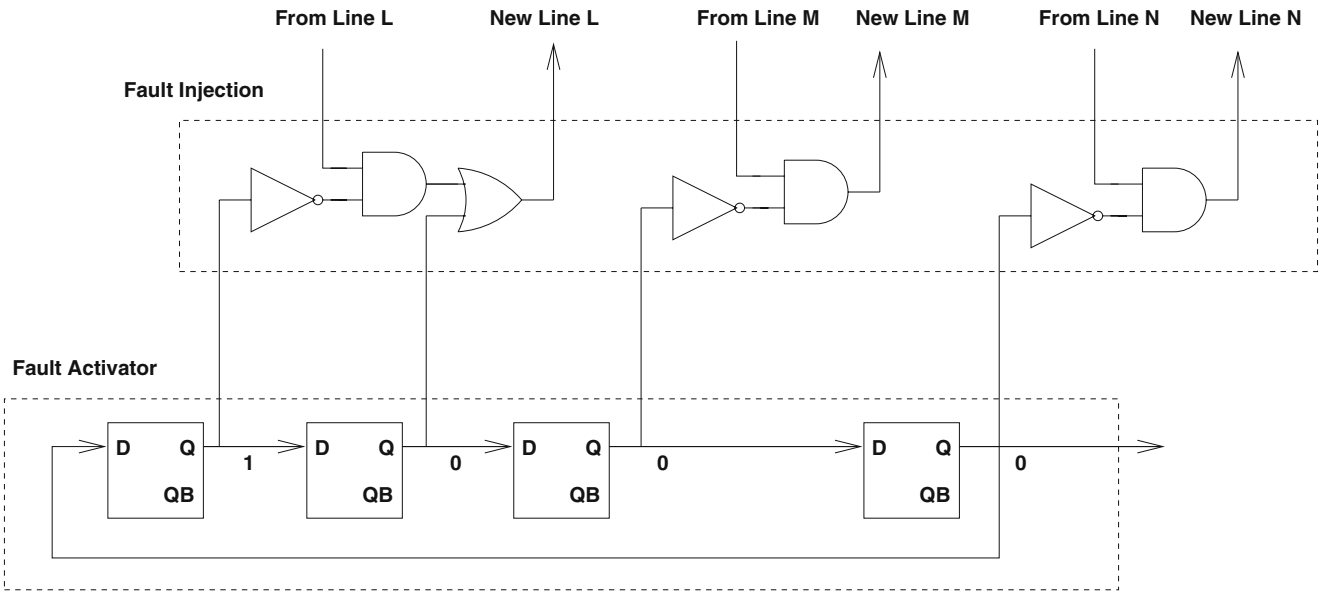


**Fig. 7** Fault activation and injection circuits for combinational circuits

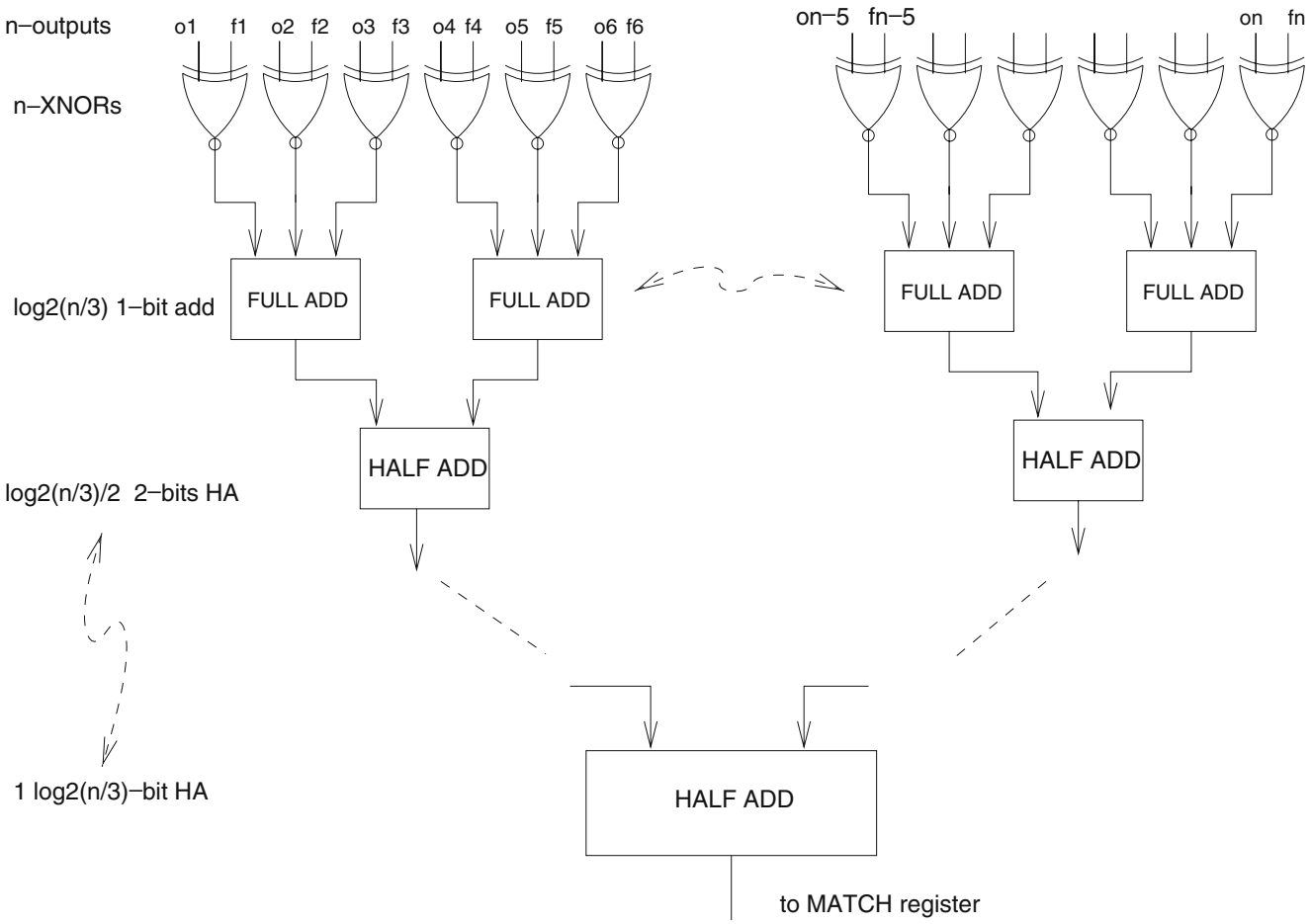


**Fig. 8** A Matcher for an n-output combinational CUD

XNOR outputs. A Matcher circuit for an n-output circuit is depicted in Fig. 8. In this figure, $o_i$ and $f_i$ are the corresponding primary outputs of the CUD and faulty circuits. The result of the addition is accumulated into register *MATCH* that at the end would contain the total number of matches for a fault. To compute the number of failed vectors, the outputs of *XNOR* gates in the Matcher block are logically NANDed to generate an increment signal for *FAILED_VECTOR* register. Register *TV* holds the number of test vectors, which is loaded at the beginning of a diagnosis session. When activating a new fault, the content of *TV* is loaded into register *TVC* that indicates at a given time which test vector is being applied to the faulty circuit model. *TVC* is decremented after the application of every test and when it becomes 0 it sends *Insert* signal to the controller to activate Ranking process.

### 3.3 Computational Block for Sequential Circuit Emulation

This section illustrates changes to the computational block to emulate and diagnose sequential circuits. Emulation of sequential circuits requires modeling and emulation of unknown values (X), in addition to logic 0 and logic 1 values, due to incompletely specified state variables. The unknown value can be either logic 1 or logic 0. Therefore, emulation uses three-valued logic (i.e., 0: logic 0, 1: logic 1, X: unknown) to compute the behavior of the sequential circuits. As a result, the faulty models of the sequential circuits need to be designed in double rail logic. Logic 0, 1 and X values are encoded in 2 bits as in Table 1 to enable emulation of unknown values: 01 for logic 0, 10 for logic 1, and 11 for X. In accordance with this encoding, the basic logic gates are remodeled as described in Table 2. For example, an AND gate in the original circuit is represented with one AND gate and one OR gate. Similarly, a primitive OR gate is represented with one OR and one AND gate in three-valued logic. In Table 3, the AND is verified, and the

**Table 1** Encoding of three-valued logic

| $Z$ | $Z_0$ | $Z_1$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| X | 1 | 1 |

**Table 2** Primitive gate modeling in double-rail logic

| Gate | Binary | Three-valued logic | |
|---|---|---|---|
| AND | $Z = X \cdot Y$ | $Z_0 = X_0 \cdot Y_0$ | $Z_1 = X_1 + Y_1$ |
| OR | $Z = X + Y$ | $Z_0 = X_0 + Y_0$ | $Z_1 = X_1 \cdot Y_1$ |
| NAND | $Z = \overline{X \cdot Y}$ | $Z_0 = X_1 + Y_1$ | $Z_1 = X_0 \cdot Y_0$ |
| NOR | $Z = \overline{X + Y}$ | $Z_0 = X_1 \cdot Y_1$ | $Z_1 = X_0 + Y_0$ |
| INV | $Z = \overline{X}$ | $Z_0 = X_1$ | $Z_1 = X_0$ |
| XOR | $Z = X \oplus Y$ | $Z_0 = X_0 \cdot Y_1 + X_1 \cdot Y_0$ | $Z_1 = (X_0 + Y_1) \cdot (X_1 + Y_0)$ |
| XNOR | $Z = \overline{X \oplus Y}$ | $Z_0 = (X_0 + Y_1) \cdot (X_1 + Y_0)$ | $Z_1 = X_0 \cdot Y_1 + X_1 \cdot Y_0$ |

other gates can be verified the same way. It should also be noted that with this coding, the inverters are eliminated from the circuits by simply switching the dual lines of the input of the inverters.

The Fault injection circuit is similar to the one proposed for sequential circuits in [10]. An injection circuit for *Line* stuck-at-0 is shown in Fig. 9a and the fault injection circuit for *Line* stuck-at-1 is shown in Fig. 9b. In this figure, $l$ is the original line and $f$ is the injection signal from the fault activator. Cascading of these two circuits serially enables injecting both stuck-at-0 and stuck-at-1 faults to the same line simultaneously. *Line C* in Fig. 10 is an example of cascading in sequential logic. The fault injection circuit receives a control signal from the fault activator circuit. The control signal is activated when the corresponding stuck-at-0 (1) is to be injected into the circuit.

The fault activator circuit is a shift register where each Flip-Flop (FF) in this register corresponds to a fault. In this representation, when a FF is set to 1 (logic high) the corresponding faults will be injected into the circuit as shown in Fig. 10. In this figure, the circuit for injecting four faults (*A* s-a-0, *B* s-a-1, C s-a-0 and *C* s-a-1) is shown. Each of these faults is activated by setting the corresponding FF to 1 which modifies the logic behavior of the circuit by fixing the logic value of the fault location to constant 0 (1) for s-a-0 (s-a-1). The necessary double rail converts are also shown in that figure. The fault activation circuit minimizes the number of reconfiguration needed to compute the suspicious location and increases the size of the emulated circuit that depends on the number of faults and fault injection circuit. In addition, one FF is needed in the fault activator circuit. So, for $p$ stuck-at-0 and $q$ stuck-at-1 faults, $p + q + 1$ FFs and $2 \times (p + q)$ additional gates are needed to emulate $p + q$ faulty circuits.

Finally, the Matcher circuit needs to be modified to calculate matches for the sequential circuits. The Matcher circuit for sequential circuits uses an extended XNOR operator ($\odot^*$) to compute the number of matches at the primary outputs of sequential circuits. This extended *XNOR* operator is a special equality operator that takes into account potential matches by handling unknown values in the comparison of the outputs of the two circuits, which are failing CUD and faulty circuit models. A potential match occurs when the CUD output has a known value whereas the respective faulty circuit $(C_f)$ output has an unknown value. Note that CUD always produces known values since all FFs in the circuits are set to 0 or 1 during startup. Table 4 tabulates the match statuses in the cells for faulty and CUD outputs: rows represent failing CUD outputs and columns represent faulty circuit model output values. In Table 4, 0, 1, and $P$ in cells indicates mismatch, match, and potential match, respectively. For example, $F =$ for responses of the faulty circuit $C_f$ and $CUD =$ for CUD's responses produce $M =$ match vector where $P$ stands for potential match ($M = F \odot^* C$). So, the number of matches for that test sequence is either 5 or 10 depending on accepting the potential matches as definite matches.

The XNOR operators in the Matcher circuit for combinational circuit presented in Table 5 are replaced by the extended XNOR operator ($\odot^*$) to compute the number of matches in sequential circuits. The hardware cost of an extended XNOR gate is seven gates if potential matches are not treated as definite matches, and three gates if potential matches are counted as definite matches.

### 3.4 Ranking in Hardware

The computation block computes the total number of matches and failed vectors for each fault and sends these values along with the fault identification to the Rank block. This block maintains an unsorted list of the first $N$ best matching faults, where $N \leq F$. The least matching fault in

**Table 3** AND gate in double rail logic

| A | B | $A_0$ | $A_1$ | $B_0$ | $B_1$ | $Z_0$ | $Z_1$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | X | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | X | 1 | 0 | 1 | 1 | 1 | 1 |
| X | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| X | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| X | X | 1 | 1 | 1 | 1 | 1 | 1 |

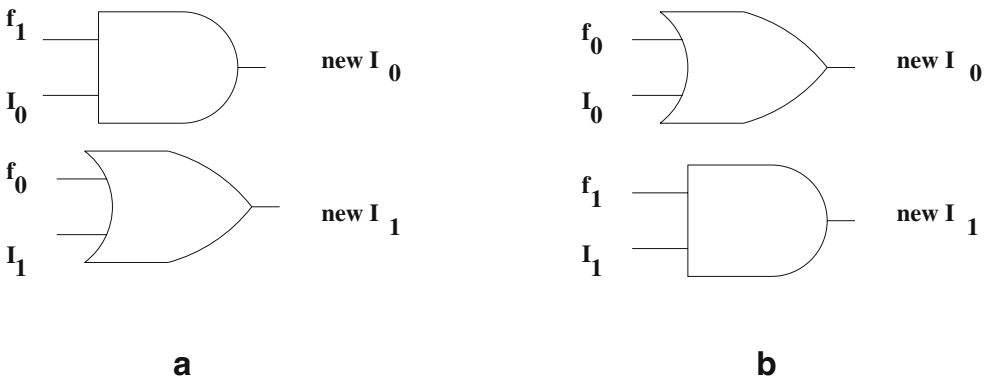


**Fig. 9** Circuits to control/set the logic value of a faulty line. **a** 0-Injection circuit. **b** 1-Injection circuit

the list is replaced by the currently emulated fault if the current fault's match is higher. Otherwise the current fault is discarded from being a suspect fault. Fig. 11 shows a Rank block for $N$=3, i.e. only three best matching faults are kept in the suspect list. In this figure, $ID_0$ is the current fault's ID from the Fault_ID register in the computation block and $D_0$ is the degree of match for that fault. $D_0$ is obtained by concatenating *Match* and the 1's complement of the number of failed vectors for this fault. Pairs of  are stored in registers. In this figure, there are three registers, three comparators and three selectors.

The descriptions of $D$, $ID$, $M$, $l$ and $L$ in Fig. 11 are as follows, *min* is the minimum function:

- $D_0 = Match\&(T - Failed\_Vectors)$
- $ID_0 = FAULT\_ID$
- $M_i = min(M_{i-1}, D_i)$ where $M_0 = D_0$
- $l_i = 1 \Rightarrow D_i \leq M_{i-1}$
- $L_i = 1 \Rightarrow D_i = min(D_0, D_1, D_2, ..., D_n)$ where $L_i$ is the LOAD signal of register *i*.

The current fault's $D$ and $ID$ values would be stored into the $i^{th}$ register if the corresponding $L_i$ signal is set; otherwise the current fault is discarded. It takes only one clock cycle to find and replace a fault with this design.

Industrial experiments show that the location of fault is one of the three or four best matching faults [20]. Therefore, $N$ can be picked any number greater than five. As a result, the size of the Rank block would be small.

## 4 Fault Diagnosis Time Approximation

The fault diagnosis system performs the following operations:

1. Initializes a few internal registers at the beginning of the diagnosis process, e.g., the test vector (TV) count register, and fault activation register with global resets,

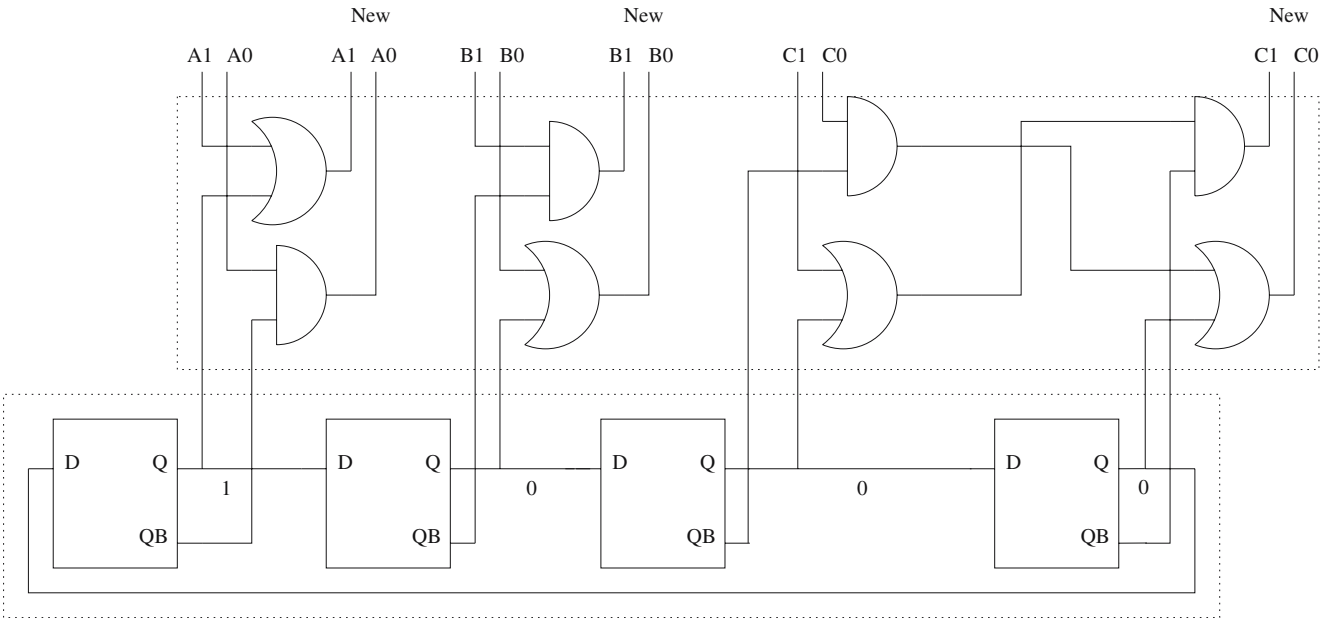


**Fig. 10** Fault activation and injection circuits for sequential circuits

**Table 4** Extended XNOR operator

| $\odot^*$ | 0 | 1 | X |
|---|---|---|---|
| 0 | 1 | 0 | P |
| 1 | 0 | 1 | P |

2. Shifts the fault activation register to activate a fault,
3. Copies TV into TVC register for every fault to read the vectors from memory using TVC as pointer,
4. Decrements TVC, and read a vector from memory,
5. Applies a vector to the faulty circuit model and updates Match and Failed_vector registers,
6. Applies a vector to the CUD, and
7. Updates the rank data structure in hardware for every fault.

Since task (1) requires two cycles and is done only once, we ignore this in our calculation. Tasks (2), (3), and (7) are done concurrently in one cycle per fault. This is possible because while we update the rank with the match results of the previous fault, at the same time we re-load TVC and shift the activation register to begin emulation of the next fault. Similarly, tasks (4), (5), and (6) are overlapped and done in one clock cycle.

Thus, the total number of clock cycles for a diagnosis process is $\simeq F \times T \times \tau_{FPGA} + N$, where $\tau$ is the FPGA speed, $F$ is the number of faults, $T$ is the number of test vectors, and $N$ is the number of faults kept in the Rank architecture.

In our speedup calculation, we excluded the preprocessing times for both emulation-based and software-based diagnosis. For the software, we did not include full fault simulation time, and fault dictionary creation time. For the emulated-based approach, we did not include FPGA map times. These two operations are performed only once per design. In this way, whenever diagnosis is needed, the precomputed models are used. Therefore, the speedup is just based on match times and is calculated according to Eq. 1.

$$Speedup = \frac{Match_{SW}}{Match_{FPGA}} \quad (1)$$

## 5 Experimental Results

In this section, the hardware and computation time requirements are investigated for the proposed dynamic diagnosis technique. Since these requirements vary from one circuit to another, the ISCAS85 and ISCAS89 benchmarks are used to estimate these requirements. The sets of locally collapsed faults and test vectors for each benchmark circuit are found with RCRIS tool [27]. The Leonardo Spectrum program from Mentor Graphics is used in the FPGA synthesis process, and the ModelSim from the same company is used for simulation. In synthesis, the circuits are implemented with the FPGA Virtex-II XCV2P70 from Xilinx [30]. This package supports up to 996 user IO pads, which is plenty for the studied benchmarks. For information, XCV2P100 package of the same family supports up to 1,164 user IO pads. XCV2P70 can operate at a maximum speed of 400 MHz. The synthesis process is run on the SUN Microcsystems Blade-100. This model is made of 500 MHz UltraSparc-IIe Processor and 768 MB memory, and is run on Solaris 9 Operating System.

The descriptions of ISCAS85 benchmark circuits are tabulated in Table 5. In this table, the name of the circuit, the number of locally collapsed faults, the number of test vectors, the number of primary inputs/outputs, the average fan-in/ the max fan-in, the average fan-out/the maximum fan-out, the number of gates in the circuit, and the sizes of full dictionaries for these circuits are tabulated in that order. The fan-ins and fan-outs of the circuits affect the number of Configuration Logic Blocks (CLBs) needed. Table 6 tabulates the reconfigurable resources needed for each

**Table 5** Description of ISCAS85 benchmarks

| Circuit | Number of faults | Number of test | Number of I/O | Avg./Max fan-in | Avg./Max fan-out | Number of gates | Size (bytes) |
|---|---|---|---|---|---|---|---|
| C17 | 22 | 8 | 5/2 | 2/2 | 2/2 | 6 | 44 |
| C432 | 524 | 50 | 36/7 | 2.10/9 | 2.65/9 | 160 | 22,925 |
| C499 | 758 | 52 | 41/32 | 2.02/5 | 4.34/12 | 202 | 15,766 |
| C880 | 942 | 31 | 60/26 | 1.90/4 | 3.50/8 | 383 | 94,906.5 |
| C1355 | 1,574 | 128 | 41/32 | 1.95/5 | 2.97/12 | 546 | 805,888 |
| C1908 | 1,879 | 153 | 33/25 | 1.70/8 | 2.58/16 | 880 | 898,396.9 |
| C2670 | 2,747 | 163 | 233/140 | 1.64/5 | 2.74/11 | 1,269 | 7,835,818 |
| C3540 | 3,428 | 211 | 50/22 | 1.76/8 | 3.15/16 | 1,669 | 1,989,097 |
| C5315 | 5,350 | 189 | 178/123 | 1.90/9 | 3.51/15 | 2,307 | 15,546,431 |
| C6288 | 7,744 | 38 | 32/32 | 1.99/2 | 2.64/16 | 2,416 | 1,177,088 |
| C7552 | 7,550 | 307 | 207/108 | 1.75/5 | 2.95/15 | 351 | 331,290,975 |

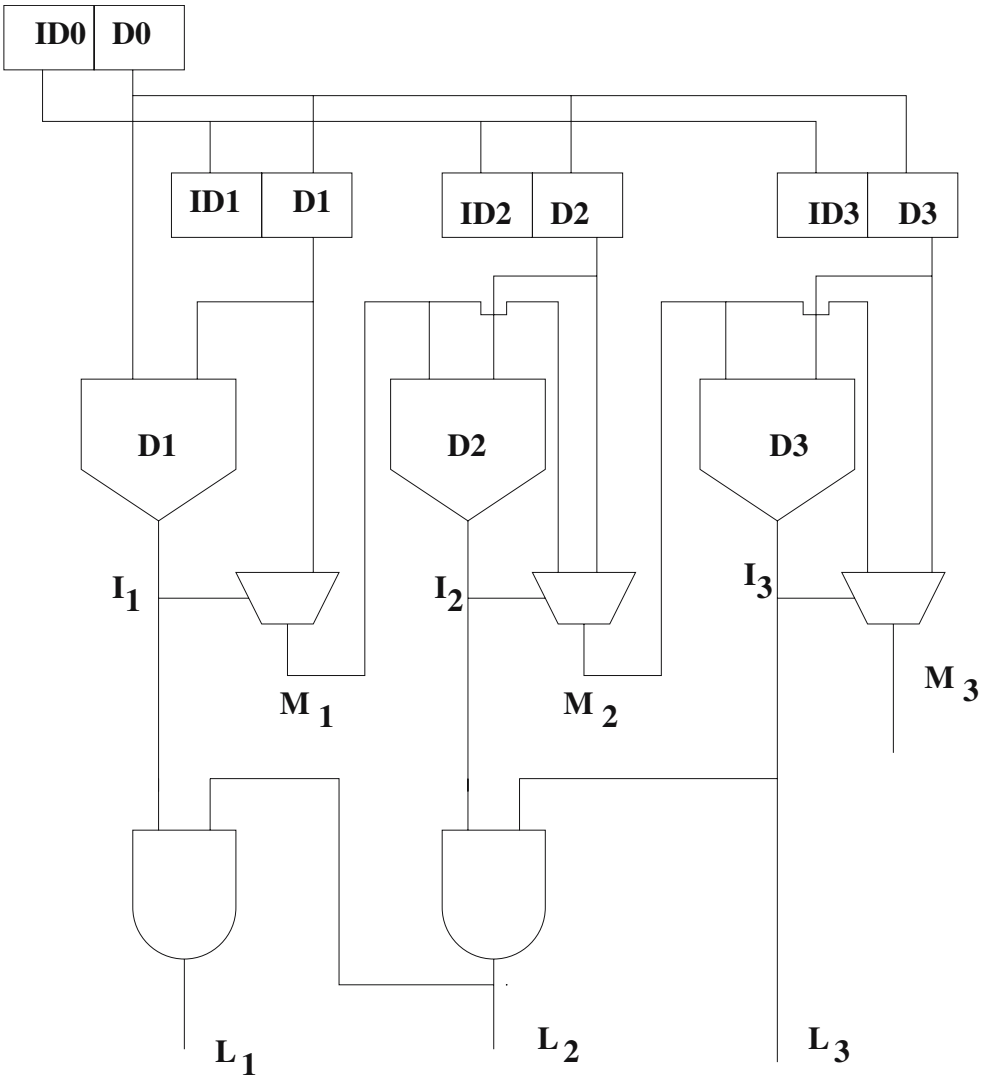


**Fig. 11** The architecture of *minCache*

circuit. We give the number of used I/O, Global Buffers, Functions Generators, CLB Slices, DFFs+Latches, Block RAMs and Block Multipliers in this order.

Additionally, Table 7 tabulates the FPGA map times in hours/minutes/seconds format, the operation speed of the FPGA in nano-seconds, dictionary built times in seconds in software, full fault simulation in seconds for the given vector test set, software match time in seconds, and the speedup. FPGA map times include preprocessing, synthesis, placement, routing and configuration bit generation times. Although map times are high, we don't remap for every diagnosis instance, and just reuse the configuration

**Table 6** The hardware resources of ISCAS85 benchmark

| Circuit | I/O | Global buffers | Function generators | CLB slices | DFFs latches | Block RAMs | Block mult |
|---|---|---|---|---|---|---|---|
| C432 | 77 | 0 | 3515 | 1758 | 2756 | 0 | 0 |
| C499 | 81 | 0 | 5330 | 2665 | 2988 | 0 | 0 |
| C880 | 100 | 0 | 5115 | 2558 | 3141 | 0 | 0 |
| C1355 | 81 | 0 | 5736 | 2868 | 3804 | 0 | 0 |
| C1908 | 73 | 0 | 5744 | 2872 | 4117 | 0 | 0 |
| C2670 | 197 | 0 | 6570 | 3285 | 4607 | 0 | 0 |
| C3540 | 90 | 0 | 6854 | 3427 | 5636 | 0 | 0 |
| C5315 | 218 | 0 | 7875 | 3938 | 6220 | 0 | 0 |
| C6288 | 72 | 0 | 10111 | 5056 | 9984 | 0 | 0 |
| C7552 | 246 | 0 | 9403 | 4702 | 8600 | 0 | 0 |

**Table 7** Speedups for ISCAS85

| Circuit | FPGA map (h/min/s) | Delay (ns) | Dictionary build (s) | Full fault sim. (s) | Software search (s) | Speedup |
|---|---|---|---|---|---|---|
| c432 | 14:18.2 | 27 | 0.19 | 23.9 | 0.17 | 240.37 |
| c499 | 21:29.6 | 97 | 0.96 | 37.3 | 0.19 | 49.69 |
| c880 | 20:57.0 | 80 | 0.62 | 21.2 | 0.26 | 111.29 |
| c1355 | 24:47.4 | 97 | 4.84 | 93.2 | 1.74 | 89.03 |
| c1908 | 26:37.2 | 78 | 5.11 | 157.7 | 2.18 | 97.21 |
| c2670 | 30:21.6 | 97 | 44.95 | 382.2 | 17.05 | 392.56 |
| c3540 | 38:35.5 | 69 | 12.39 | 747.8 | 5.01 | 100.38 |
| c5315 | 51:19.6 | 97 | 88.49 | 547.7 | 33.75 | 344.10 |
| c6288 | 1:24:50.1 | 97 | 7.00 | 443.2 | 2.72 | 95.29 |
| c7552 | 1:19:15.7 | 97 | 181.65 | 1924.5 | 62.91 | 279.80 |

bits over and over again. Full fault simulation includes simulation of all vectors for all faults. Considering only the circuits with high software match times, the emulation approach is 340 times faster on average.

We repeated the experiments for ISCAS89 sequential circuits to measure their diagnosis times using reconfigurable hardware and required reconfigurable resources. Table 8 describes each sequential benchmark. The reduced fault set and test vectors are found by the RCRIS tool. Table 9 tabulates the required hardware resources. Finally, Table 10 tabulates speed-up data for ISCAS89. In this table, average fault simulation time per fault is used. Considering only the circuits with high software match times, the emulation approach is 252 times faster on average.

## 6 Scalability Issues

Two complementary methods can be used to perform diagnosis of circuits with high number of I/Os:

1. *Buffering of the output values from the CUD*: We note that the output values from the CUD are inputs to the Matcher circuit. While the faulty circuit model computes the response of the faulty circuit, the output

**Table 8** The descriptions of ISCAS89 circuits

| Circuit | Number of faults | Number of tests | Number of I/O | Avg./max fanin | Avg./max fanout | Number of flip-flops | Number of gates | Size(KB) |
|---|---|---|---|---|---|---|---|---|
| s298 | 308 | 331 | 3/6 | 1.94/4 | 1.98/13 | 14 | 119 | 75 |
| s344 | 342 | 308 | 9/11 | 1.62/3 | 1.56/8 | 15 | 160 | 141 |
| s349 | 350 | 367 | 9/11 | 1.63/3 | 1.57/8 | 15 | 161 | 172 |
| s382 | 399 | 443 | 3/6 | 1.82/4 | 1.85/21 | 21 | 158 | 129 |
| s386 | 384 | 621 | 7/7 | 2.13/4 | 2.13/23 | 6 | 159 | 204 |
| s400 | 423 | 146 | 4/6 | 1.85/4 | 1.88/22 | 21 | 163 | 45 |
| s420 | 455 | 1945 | 18/1 | 1.70/4 | 1.59/14 | 16 | 218 | 108 |
| s444 | 474 | 437 | 3/6 | 1.84/4 | 1.87/22 | 21 | 181 | 152 |
| s526 | 555 | 1,100 | 3/6 | 2.17/4 | 2.20/13 | 21 | 193 | 447 |
| s641 | 467 | 990 | 35/24 | 1.40/4 | 1.36/12 | 19 | 379 | 1,354 |
| s713 | 581 | 918 | 35/23 | 1.48/4 | 1.43/12 | 19 | 393 | 1,497 |
| s820 | 850 | 599 | 18/19 | 2.59/4 | 2.60/49 | 5 | 289 | 1,181 |
| s832 | 870 | 721 | 18/19 | 2.65/4 | 2.66/50 | 5 | 287 | 1,455 |
| s1196 | 1,242 | 360 | 14/14 | 1.87/4 | 1.87/17 | 18 | 529 | 764 |
| s1238 | 1,355 | 380 | 14/14 | 2.01/4 | 2.01/19 | 18 | 508 | 880 |
| s1423 | 1,515 | 1,773 | 17/5 | 1.69/4 | 1.66/43 | 74 | 657 | 1,639 |
| s1488 | 1,486 | 1,023 | 8/19 | 2.11/4 | 2.15/55 | 6 | 653 | 3,526 |
| s1494 | 1,506 | 1,201 | 8/19 | 2.14/4 | 2.17/56 | 6 | 647 | 4,195 |
| s5378 | 4,603 | 3,200 | 35/49 | 1.48/4 | 1.49/10 | 179 | 2,779 | 88,104 |
| s9234 | 6,927 | 720 | 36/39 | 1.40/4 | 1.40/32 | 211 | 5,597 | 23,744 |
| s13207 | 9,815 | 635 | 62/152 | 1.37/4 | 1.38/37 | 638 | 7,951 | 115,643 |
| s15850 | 11,725 | 1,653 | 77/150 | 1.37/4 | 1.38/34 | 534 | 9,772 | 354,884 |
| s35932 | 39,094 | 951 | 35/320 | 1.68/2 | 1.68/1449 | 1,728 | 16,065 | 1,452,281 |

**Table 9** The hardware resources of ISCAS89 benchmark

| Circuit | I/O | Global buffers | Functions generators | CLB slices | DFF's latches | Block RAMS | Block multipliers |
|---|---|---|---|---|---|---|---|
| s298 | 44 | 0 | 4268 | 2134 | 4250 | 0 | 0 |
| s344 | 59 | 0 | 4640 | 2320 | 4278 | 0 | 0 |
| s349 | 59 | 0 | 4649 | 2325 | 4286 | 0 | 0 |
| s382 | 47 | 0 | 4448 | 2224 | 4369 | 0 | 0 |
| s386 | 55 | 0 | 4490 | 2245 | 4282 | 0 | 0 |
| s400 | 49 | 0 | 4437 | 2219 | 4393 | 0 | 0 |
| s420 | 77 | 0 | 4056 | 2197 | 4394 | 0 | 0 |
| s526n | 47 | 0 | 4652 | 2326 | 4523 | 0 | 0 |
| s526 | 47 | 0 | 4634 | 2317 | 4525 | 0 | 0 |
| s641 | 111 | 0 | 5820 | 2910 | 4412 | 0 | 0 |
| s713 | 111 | 0 | 5899 | 2950 | 4526 | 0 | 0 |
| s820 | 77 | 0 | 6115 | 3058 | 4730 | 0 | 0 |
| s832 | 77 | 0 | 6174 | 3087 | 4750 | 0 | 0 |
| s1238 | 69 | 0 | 6431 | 3216 | 5213 | 0 | 0 |
| s1423 | 75 | 0 | 6038 | 3019 | 5697 | 0 | 0 |
| s1488 | 75 | 0 | 6038 | 3019 | 5697 | 0 | 0 |
| s1494 | 57 | 0 | 7190 | 3595 | 5413 | 0 | 0 |
| s5378 | 111 | 0 | 12686 | 6343 | 9122 | 0 | 0 |
| s9234 | 113 | 0 | 11628 | 5814 | 8461 | 0 | 0 |
| s13207 | 165 | 0 | 16300 | 8150 | 11795 | 0 | 0 |
| s15850 | 195 | 0 | 6491 | 3246 | 992 | 0 | 0 |
| s35932 | 111 | 0 | 14069 | 7129 | 14257 | 0 | 0 |

values from the CUD can be serially buffered internally in the FPGAs using a few IO lines.

2. *Mapping the faulty circuit model onto multi FPGA platform*: The faulty circuit model can be decomposed into several subcircuits where each subcircuit consists of a subset of primary outputs and their dependent logic. Each partition is mapped onto an FPGA as shown in Fig. 12. Let $O_{cud}$ be the set of primary outputs of CUD. We partition $O_{cud}$ into $k$ sets to obtain $O_1, O_2 \cdots O_k$ where $\forall_{i,j,i\neq j} o_i \cap o_j = \phi$ and $\bigcup_{i=1}^{k} O_i = O_{cud}$. We map

**Table 10** Speedups for ISCAS89

| Circuit | FPGA map (h/min/s) | Delay (ns) | Dictionary build (s) | Simulation per fault (ms) | Software match (s) | Speedup |
|---|---|---|---|---|---|---|
| s298 | 38:29.3 | 23 | 0.29 | 3.02 | 0.23 | 98.09 |
| s344 | 41:10.8 | 40 | 0.54 | 1.81 | 0.28 | 66.45 |
| s349 | 41:14.6 | 41 | 0.66 | 1.81 | 0.34 | 64.56 |
| s382 | 40:05.7 | 23 | 0.58 | 2.9 | 0.41 | 100.85 |
| s386 | 41:30.2 | 22 | 0.9 | 2.9 | 0.58 | 110.56 |
| s400 | 40:26.1 | 19 | 0.21 | 3.2 | 0.14 | 119.31 |
| s420 | 38:50.2 | 9 | 1.4 | 3.2 | 1.57 | 197.12 |
| s526 | 54:48.4 | 23 | 2.01 | 6.9 | 1.43 | 101.84 |
| s641 | 50:24.3 | 83 | 4.6 | 3.5 | 1.68 | 43.78 |
| s713 | 1:08:26.0 | 28 | 5.15 | 3.6 | 1.81 | 121.2 |
| s820 | 1:30:32.3 | 67 | 4.18 | 2.4 | 1.7 | 49.83 |
| s832 | 1:57:16.4 | 60 | 5.11 | 2.3 | 2.13 | 56.59 |
| s1238 | 1:20:27.3 | 50 | 0.81 | 17.68 | 0.39 | 15.15 |
| s1423 | 1:34:11.2 | 83 | 8.01 | 109.2 | 5.67 | 25.43 |
| s1488 | 1:14:17.4 | 67 | 12.39 | 19.3 | 5.02 | 49.29 |
| s1494 | 1:14:38.6 | 67 | 26.07 | 12.4 | 7.31 | 60.32 |
| s5378 | 4:03:58.9 | 36 | 287.39 | 198.1 | 110.43 | 208.25 |
| s9234 | 5:04:38.0 | 109 | 78.55 | 80.7 | 33.06 | 60.81 |
| s13207 | 10:12:04.0 | 53 | 359.46 | 1,384.2 | 131.61 | 398.43 |
| s15850 | 10:12:12.3 | 109 | 2,042.61 | 947.3 | 391.25 | 185.2 |
| s35932 | 21:02:47.6 | 109 | 16,630.83 | 1,542.25 | 877.77 | 216.6 |

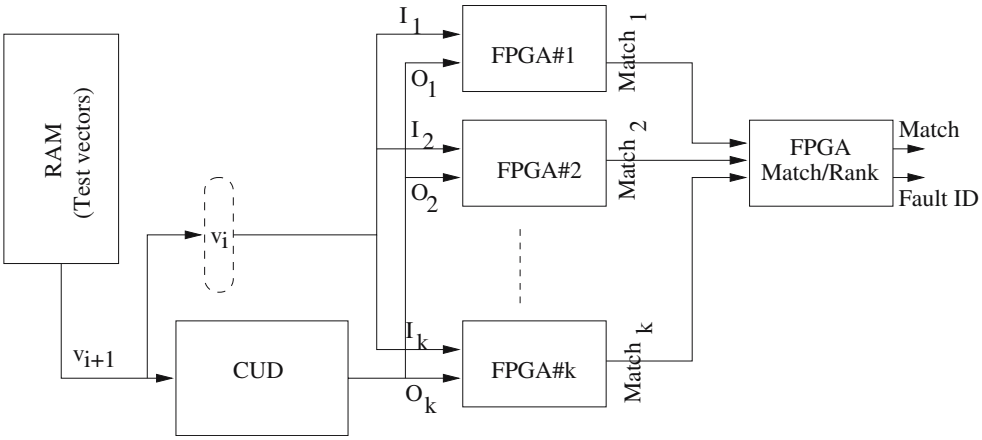


**Fig. 12** Multi-FPGA platform

the subcircuit that computes $O_i$ onto $FPGA_i$. Let $I$ be the set of inputs of CUD. Then, $I_i \in I$ and $\bigcup_{i=1}^{k} I_i = I$. However, the intersection of $I_i$ and $I_j$ may not be empty. In this Figure, $FPGA_i$ computes partial matches for $O_i$. The FPGA rank/match sums the generated partial matches and updates fault rank.

In this configuration, it is clear that the computation is performed in pipelined fashion where the first stage computes CUD response, the second stage computes the FPGA response, and the third stage computes the rank of the fault. Therefore, these three computations can be overlapped.

## 7 Extending Our Method to Other Fault Models

This section demonstrates the extensibility and generalizability of the emulation-based diagnosis approach to other fault models. Since fault activation circuitry does not change from one fault model to another, we only illustrate the fault injection models for bridging, bus stuck-at, and register stuck-at fault models. *Bridging fault* (BF) models capture the logical behavior of shorts between two or more unconnected lines or nets. The value on a short line depends on the technology. Figure 13a shows the bridging of lines $a$ and $b$, and while Fig. 14b captures functional behavior of these two lines using function $F(a, b)$, where $a$ and $b$ are the driven values on these lines. For $a = b$, $F(a, a) = a$ and for $a \neq b$, $F(a, b)$ may be determinate or inderteminate, depending on the technology. In many technologies the strong value overrides the other value. Since we know the technology of the realized circuits, based on that, we can devise a circuitry to implement $F(a, b)$ and use this as fault injector. When the bridging fault is activated, the new values on lines $a$ and $b$ would be both $F(a, b)$.

We provide two more examples to illustrate extensibility of the proposed diagnosis method. The first one, as shown in Fig. 14a, shows the fault injection circuitry for bus stuck-at-1 fault, and the second one, as shown in Fig. 14b, illustrates the fault injection circuitry for a register stuck-at-1 fault. High level fault models are generalization of single stuck-at faults and can be modeled as multiple stuck-at fault

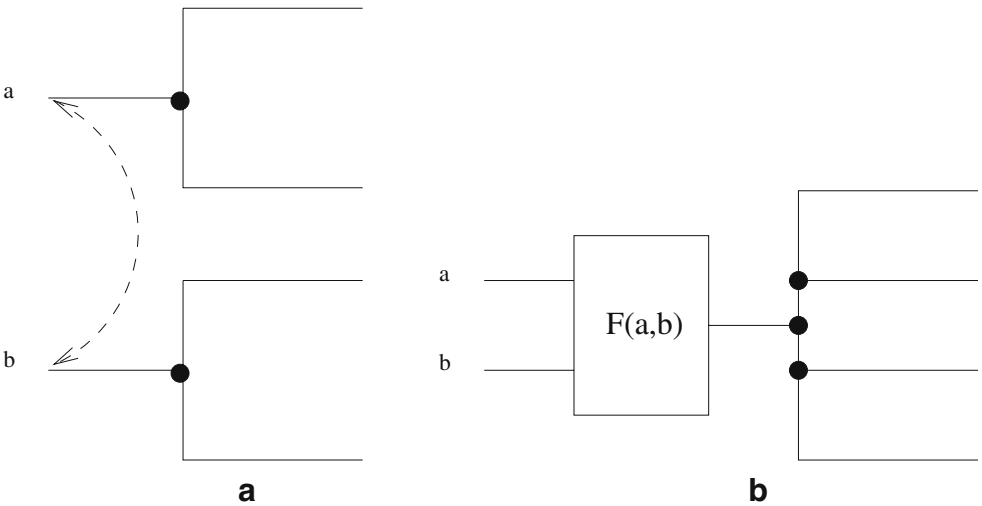


**Fig. 13** Bridge fault model
**a** BF model, **b** BF effect

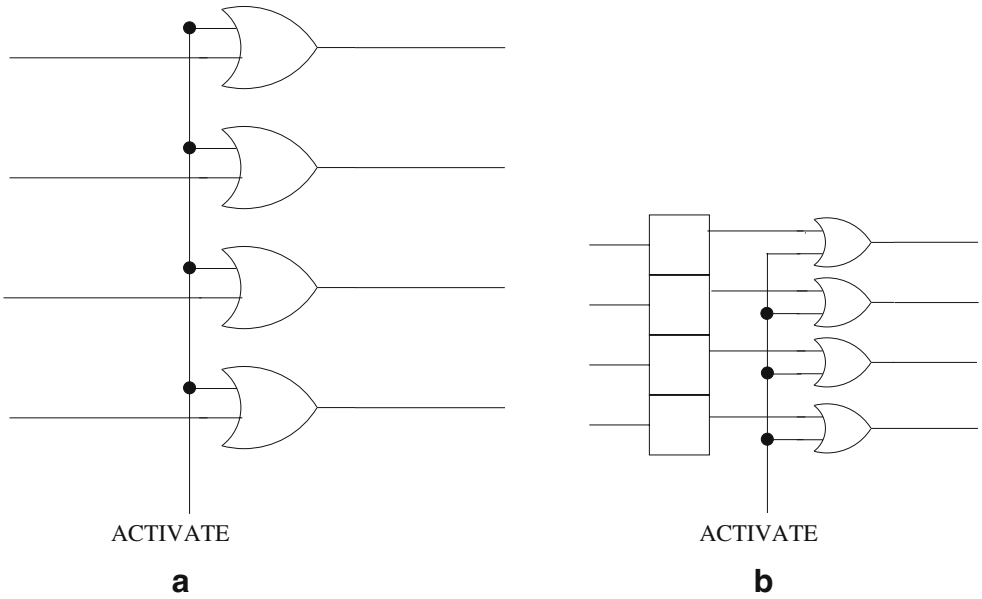


**Fig. 14** Some RTL fault models. **a** Bus stuck-at-1 fault model, **b** Register stuck-at-1 fault model

models by providing a common activation signal to all components of a composite fault model.

## 8 Conclusions

This article introduces a novel emulation-based approach to locate faults in combinational and sequential circuits. The approach tailors a dynamic diagnosis algorithm for the circuit under diagnosis (CUD) and runs the algorithm on the reconfigurable hardware to accelerate the diagnosis time. The emulation-based diagnosis method is on average 340× faster for combinational and 252× faster for sequential circuits than the software-based method. Moreover, the proposed approach preserves the diagnostic resolution of a full fault dictionary-based diagnosis and offers a hardware-only diagnosis method.

## References

1. Abramovici M (1981) A maximal resolution guided-probe testing algorithm. In: Proc. of ACM/IEEE Design Automation Conference, pp 189–195
2. Abramovici M, Breuer MA (1980) Fault diagnosis based on effect-cause analysis. In: Proc. of ACM/IEEE Design Automation Conference, pp 69–76
3. Abramovici M, Menon P (1997) Fault simulation on reconfigurable hardware. In: Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), pp 182–190, 16–18 April 1997
4. Abramovici M, Saab DG (1997) Satisfiability on reconfigurable hardware. In: Int'l. Workshop on Field-Programmable Logic and Applications, September 1997
5. Abramovici M, Breuer MA, Friedman AD (1995) Digital systems testing and testable design. IEEE Press
6. Augusto J, Almeida C, Neto H (2003) A Modular reconfigurable architecture for efficient fault simulation in digital circuits. In: Proc. of Field-Programmable Logic and Aplications, Springer, Berlin
7. Burgun L, Reblewski F, Fenelon G, Berbier J, Lepape O (1996) Serial fault emulation. In: Proceedings of ACM/IEEE Design Automation Conference, pp 801–806
8. Butts JBM, Varghese J (1992) An efficient logic emulation system. In: Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp 138–141, 11–14 October 1992
9. Camurati P, Medina D, Prinetto P, Reorda MS (1990) A diagnostic test pattern generation algorithm. In: Proc. of IEEE Int'l Test Conf., IEEE Computer Society Press, pp 52–58, Sept. 1990
10. Cheng K-T, Huang S-Y, Dai W-J (1999) Fault emulation: a new methodology for fault grading. IEEE Trans Comput-aided Des Integr Circuits Syst 18:1487–1495
11. Cox H, Rajski J (1988) A method of fault analysis for test generation and fault diagnosis. IEEE Trans on CAD of Integrated Circuits and Systems, pp 813–833
12. Hartanto I, Venkataraman S, Fuchs WK, Rudnick EM, Patel JH, Chakravarty S (2001) Diagnostic simulation of stuck-at faults in sequential circuits using compact lists. ACM Transact Des Automat Electron Syst 6:471–489
13. Kocan F, Saab DG (1998) Dynamic fault diagnosis for sequential circuits on reconfigurable hardware, in Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, 5–7, pp 214–215, October
14. Kocan F, Saab DG (1999) Dynamic fault diagnosis on reconfigurable hardware. In: Proceedings of ACM/IEEE Design Automation Conference, pp 691–696
15. Kocan F, Saab DG (2001) *ATPG* for combinational circuits on configurable hardware. IEEE Trans Very Large Scale Integr (VLSI) Syst 9:117–129
16. Kubiak K, Parkes S, Fuchs WK, Saleh R (1992) Exact evaluation of diagnostic test resolution. In: Proc. of ACM/IEEE Design Automation Conference, pp 347–352
17. Miremadi SG, Ejlali A (2003) Switch level fault emulation. In: Proc. of Field-Programmable Logic and Aplications. Springer, Berlin
18. Pagarani T, Kocan F, Saab DG, Abraham JA (2000) Parallel and scalable architecture for solving SATisfiability on reconfigurable FPGA. In: Proceedings of IEEE Custom Integrated Circuits Conference, pp 147–150
19. Parreira A, Santos M, Sousa J (2003) Fault simulation using partially reconfigurable hardware. In: Proc. of Field-Programmable Logic and Aplications. Springer, Berlin

20. Paul WKF, Ryan G, Rawatt S (1991) Two-stage fault location. In: Proc. of IEEE Int'l Test Conf., pp 963–968
21. Pomeranz I, Reddy S (1992) On the generation of small dictionaries for fault location. In: Proc. of IEEE/ACM Int'l Conf. on CAD, pp 272–279
22. Pomeranz I, Reddy SM (1997) On dictionary-based fault location in digital logic circuits. IEEE Trans Comput 46:48–59
23. Rashid A, Leonard J, Mangione-Smith WH (1998) Dynamic circuit generation for solving specific problem instances of boolean satisfiability. In: Pocek KL, Arnold J (eds) IEEE Symposium on FPGAs for Custom Computing Machines. IEEE Computer Society Press, Los Alamitos, CA, pp 196–204
24. Reorda MS, Violante M (2003) Emulation-based analysis of soft errors in deep sub-micron circuits. In: Field-Programmable Logic and Applications. Springer, Berlin
25. Richman J, Bowden KR (1985) The modern fault dictionary. In: Proc. of IEEE Int'l Test Conf, pp 696–702
26. Ryan PG, Fuchs WK, Pomeranz I (1993) Fault dictionary compression and equivalence class computation for sequential circuits. In: Proc. of IEEE/ACM Int'l Conf. on CAD, pp 508–511
27. Saab DG, Saab YG, Abraham JA (1986) Automatic test vector cultivation for sequential vlsi circuits using genetic algorithms. IEEE Trans on CAD of Integrated Circuits and Systems 15:1278–1285
29. Suyama T, Yokoo M, Sawada H (1996) Solving satisfiability problems on fpgas. In: Proc. Intn'l. Workshop on Field-Programmable Logic and Applications
28. Saab DG, Kocan F, Abraham JA (2002) Massively parallel/reconfigurable emulation model for the d-algorithm. In: Manfred Glesner PZ, Renovell M (eds) Proceedings of 12th International Conference on Field-Programmable Logic and Applications, Lecture Notes in Computer Science 2438, Springer, pp 1172–1176, 2–4 September 2002
30. XILINX, Vertex-ii pro complete data sheet, in http://www.xilinx.com
31. Zhong P, Martonosi M, Ashar P, Malik S (1998) Accelerating boolean satisfiability with configurable hardware. In: Pocek KL, Arnold J (eds) IEEE Symposium on FPGAs for Custom Computing Machines. IEEE Computer Society Press, Los Alamitos, CA, pp 186–195

**Fatih Kocan** received the B.S. degree in computer science and engineering from Hacettepe University, Ankara, Turkey, and the M.Sc. and Ph.D. degrees in databases and VLSI/CAD, respectively, from Case Western Reserve University, Cleveland, OH. He is currently an Assistant Professor at Southern Methodist University, Dallas, TX. His current research interests include algorithms, VLSI/CAD, FPGA/CAD, and embedded systems.

**Daniel G. Saab** received the B.S. degree in Computer Engineering, the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1982, 1985, 1988, respectively. In 1982 and 1983, he worked as an engineer at Tektronix, Inc. in Beaverton, Oregon. From 1983 to 1994, he was a research assistant at the Coordinated science Laboratory and an Assistant Professor of Electrical Engineering at the University of Illinois at Urbana-Champaign. In 1994, he joined Case Western Reserve University where he is an associate Professor of Electrical Engineering and Computer Science. Dr. Saab was visiting scientist at the Somerset IBM power PC group in Texas, at Bell Labs Lucent Technology in New Jersey, Level1 Communication an Intel company in California, and National Semiconductor in San-Jose California. He published over 83 research papers in the area of Computer Aided Design (CAD) focusing on design verification, evaluation of test set quality, and on the generation of high quality tests for large digital systems. Dr. Saab has lead several research projects to fruition those included: The development of the first symbolic switch-level simulator EXPRESS, the development of hierarchical simulation environment (CHAMP) which was used by Motorola to evaluate the quality of the MC68000 manufacturing tests. Dr. Saab won the 1988 Semiconductor Research Corporation Inventor award for the CHAMP techniques. He also devise the clever application of genetic-algorithms to solve the automatic generation of manufacturing test resulting in 30 times speed-up over traditional state-of-art algorithms. Dr. Saab's interests are design automation, formal design verification, low-power design and test.