Automatic diagnosis of single fault in interconnect testing of SRAM-based FPGA

T. Nirmalraj¹ | S. Radhakrishnan¹ | S.K. Pandiyan²

¹Department of Electronics and Communication Engineering, Srinivasa Ramanujan Centre, SASTRA University, Thanjavur, Tamil Nadu, India
²Department of Electronics and Communication Engineering, SASTRA University, Thanjavur, Tamil Nadu, India

Correspondence
Nirmalraj, Department of Electronics and Communication Engineering, Srinivasa Ramanujan Centre, SASTRA University, Thanjavur, Tamil Nadu, India.
Email: nirmalraj.harsha@gmail.com

1 | INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are utilized in various critical applications because of its faster design cycle, re-configurability, low cost and so forth [1]. The general architecture of an FPGA comprises a two-dimensional array of Configurable Logic Blocks (CLBs), programmable interconnects (‘nets’) that run horizontally and vertically between CLBs and programmable Input Output Blocks (IOBs) [2]. In recent years, the number of transistors fabricated in an FPGA has increased with the current trend of deep sub-micron technology. The research results of Interuniversity Micro-electronics Center, Belgium, predicts that the average time taken for a chip fabricated on a 45 nm process will fail in less than a year [3]. Moreover, 80% of programmable resources in an FPGA are interconnects that are stacked in eight metal layers [4–6]. Hence, FPGAs are very vulnerable to defects in the interconnect networks, thereby making the testing of these interconnects extremely vital [6–8].

The interconnect test strategies are broadly classified as application-dependent and application-independent testing [6–8]. Within the application-dependent testing, only those nets that are utilized in executing an application are subjected to testing. On the other hand, testing of all interconnects in an FPGA is called as application-independent testing.

Different methodologies for application-dependent testing have been presented in literature for logic block testing or interconnect testing or both [1, 2, 4, 6–24]. These testing methodologies are able to not only detect faults within the FPGA chip but also diagnose the actual location of the faults [1, 5, 8, 10, 12, 16, 20–22]. Some of the previous methods require external hardware to perform the actual testing and to diagnose faults on the Unit Under Test (UUT), whereas other methods only require the reprogramming of the Look-Up Table (LUT) while maintaining the configuration of the interconnects in order to perform the testing [1, 2, 8, 21, 22].

The method presented in [1] is able to detect multiple faults in an application-dependent interconnect of an SRAM-based FPGA. This technique retains the original interconnect configuration and modifies the function of the LUTs using 1-Bit Sum Function. The proposed method [1] detects all possible stuck-at and bridging faults of all cardinalities using \(1 + \log_2 k\) test configurations for multiple stuck-at locations and \(2 + 2\log_2 k\) additional test configurations to locate more

Abstract
Fault detection and diagnosis of a Field-Programmable Gate Array (FPGA) in a short period is vital particularly in reducing the dead time of critical applications that are running on FPGAs. Thus, this paper proposes a new technique that is able to uniquely identify any single stuck-at fault’s location along with the type of fault. Also, the presented technique is able to locate any single pair-wise bridging fault and distinguish between the two types of common faults. The presented technique uses the Walsh Code method to significantly reduce the number of test configurations when compared with previous methods. Extensive testing of the proposed method is carried out on a series of ISCAS’89 benchmark circuits being implemented in different FPGA families. From the simulation results, the maximum number of configurations needed for interconnect fault detection and diagnosis is \(\log_2 \left( \frac{n(n-1)}{2} \right) + 3\) where \(n\) is the number of nets under test. It is noted that the proposed method is able to reduce the total number of test configurations by \(\log_2(n + 2)\) when compared with previously published methods available in the literature.
than one pair-wise bridging faults (where k is the maximum combinational depth of the FPGA circuit). In addition, the locations of multiple faults are identified using the Walking-1 test set.

The methodology presented in [8] is able to both detect and diagnose single stuck-at and single bridging fault location in FPGA chips by only reprogramming the configurations in LUT. However, the number of configurations needed to perform the fault detection is at least \( \log_2(N + 2) \) where \( N \) is the number of nets whereas diagnosis of the fault requires a different set of configurations. The number of configurations needed to diagnose single stuck-at faults is \( \log_2(2N) \) and the number of configurations required for diagnosing a single pair-wise bridging fault is \( \log_2[N(N-1)/2] \).

Unlike the approach in [8] where every net is driven by a single input, the technique presented in [24] uses the concept of activating inputs that test inputs simultaneously by driving multiple nets. Hence the testing is very efficient in terms of requiring a lower number of configurations and reduces redundancy in test configurations. [24] requires at most \( \log_2(M + 2) \) configurations (where M denotes the number of activating inputs).

Although the technique for performing fault detection and diagnosis of interconnects using multiplexer method presented in [19] is able to detect and diagnose multiple faults, this method alters the interconnect configurations in order to perform the testing method. Moreover, multiplexer outputs may be faulty thereby causing the correct output of an LUT to be propagated with a wrong value. In this case, even if no logic blocks have any fault, the reroute of such configuration may induce a false fault causing it to be hard to detect.

A fine-grain diagnostics technique presented in [20] locates multiple interconnect faults and diagnoses the precise location of faulty interconnect. This technique requires the switch matrix to be reconfigured at least one position from the original connection position to diagnose the actual location of the interconnect faults. However, this technique is applicable only for detecting stuck-at faults but not for detecting pair-wise bridging faults where the bridging fault between nets are overlapped due to only logic ‘1’ and logic ‘0’ set for every LUT used. Moreover, this technique only considers the faults occurring within the nets under test and assumes that the unused nets are fault-free.

Another method that uses Boolean satisfiability to perform application-dependent interconnect testing of FPGAs for different stuck-at, bridging, and open faults is presented in [22]. This technique requires the same number of test configurations to test larger circuits when compared with the Walsh-based technique used in [24]. However [22] takes more time in generating test vectors. Next the method that utilizes satisfiability modulo theory (SMT) to test configurations for detecting interconnect faults like stuck-at, open and bridging faults is presented in [23]. In this method, the test generation problem is formulated as a set of constraints and an SMT solver is used to solve the optimization problem. Although this method is able to detect the interconnect faults, unlike proposed method, it is unable to locate and diagnose the interconnect faults. The method presented in [25] focuses on testing one of the elements of the interconnect known as hex PIPs of an FPGA for the possible stuck-at-0 and stuck-at-1 faults. While the faults in other elements of the interconnect and bridging faults are not covered in this method.

Differing from the above methods, this paper presents a fully automated method for application-dependent testing of interconnects, and in particular, focuses on locating and diagnosing a single fault by exploiting Walsh Code. The proposed method uses a reduced number of test configurations when compared with existing methods that are available in the literature. Any single fault can be uniquely identified and located. Moreover, this method does not require any external hardware configurations to locate the actual fault location. The proposed method requires \( \log_2(n + 2) \) less number of total configurations to perform both fault detection and fault diagnosis as compared to the method presented in [8], thereby achieving an average of 48% reduction in the number of test configurations as compared to existing methods available in the literature.

The rest of the paper is organized as followed. Section 1 reviews previous work done. Section 2 describes the proposed method for locating interconnect faults. Section 3 compares this proposed method with previous work. Finally, Section 4 concludes the paper.

## 2 | PROPOSED TECHNIQUE FOR LOCATING INTERCONNECT FAULT

The proposed technique reverses the current fault detection methods [8, 24] and is able to cover 100% of faults by exploiting the Walsh Code method to accurately identify the location of any type of single interconnect fault and its characteristics by using \( \log_2(N + 2) \) test configurations. Nevertheless, depending on the number of nets and the location of the fault occurrence, additional configurations are generated to narrow down the actual fault location. Even though the Walsh Code is being used for fault diagnosis, the principle behind single stuck-at fault and single pair-wise bridging fault is slightly different from one another. Before implementing any testing for the detection/diagnosis of faults using the proposed methodology, sequential elements in the circuit such as flip-flops, inverters, and multiplexers on the routing paths need to be removed.

### 2.1 | Single-term function

As mentioned earlier, Walsh Code method will be used for both fault detection and diagnosis. However, the main principle for generating configurations using Walsh Code method is derived from the single-term function where the logic function has only one maxterm or only one minterm value. The respective input pattern of a function for minterm or maxterm is known as an activating input. For instance, as shown in Figure 1, \( F = A + B' + C' + D \) has one minterm output
value of ‘0’ for an activating input of ABCD = 0110 while rest of the input values result in an output value of ‘1’ [24].

Figures where an undesired output result is obtained when an activating input is applied indicate that a fault has occurred. Besides that, with an activating input of ABCD = 0110 applied, certain faults are covered such as A/1 (A stuck-at-1 fault), B/0 (B stuck-at-0 fault), C/0 and D/1 for stuck-at faults, AB OR (AB bridging fault OR), AB AND (AB bridging fault AND), AC OR, AC AND, BD OR, BD AND, CD OR and CD AND for pair-wise bridging faults [24].

As mentioned in the above example, there are two types of stuck-at fault: stuck-at-1 fault and stuck-at-0 fault. When a net is stuck-at-1, it means that the net is permanently tied to logic ‘1’ no matter what input value is being applied. This situation applies to stuck-at-0 fault too. If a net is labelled A and has a stuck-at-1 fault, then the fault is denoted as A/1. On the other hand, a net labelled B and having a stuck-at-0 fault is denoted as B/0 [24]. Moreover, as demonstrated in the above example, it should be noted that there are two types of pair-wise bridging fault: wire-AND and wire-OR. If wire-AND is the dominant fault for the pair-wise bridging fault, the end line of both nets will carry the logic value of ‘0’ and the fault is denoted as AB AND. However, if wire-OR is the dominant fault for the pair-wise bridging fault, the end line of both nets will carry the logic value of ‘1’ and such a fault is denoted as AB OR [24]. Nevertheless, with the single-term function applied, any input apart from the activating input value will result in an undesired output value. Therefore, if a series of single-term functions are applied to the logic blocks, both stuck-at and pair-wise bridging faults are 100% covered for fault detection.

### 2.2 Test configuration generation

As mentioned in above subsection, before generating any configurations based on the number of nets under test, the sequential elements that exist in the design file are removed in order to obtain 100% coverage of faults. The sequential elements are removed such that the input of the removed sequential elements is connected to its output to complete the design connection. It can be concluded that the design file without sequential elements is same as the design file that have sequential elements.

Based on the example explained in the above subsection, it is clearly shown that single-term functions are able to cover all faults using Walsh Code method. However, since pair-wise bridging faults only occur between two nets that have different input value, all ‘0’ and all ‘1’ configurations are avoided when assigning configuration values for each net. Therefore, the maximum number of configurations will be \(\log_2(n + 2)\) where \(n\) is the total number of nets under test. These number of configurations are able to detect all possible stuck-at and pair-wise bridging faults. These configuration vectors are then converted into activating inputs for each LUT under test in order to implement the single-term function method for each of the LUT under test. The algorithm used for the design manipulation and configuration generation is demonstrated in Table 1.

### 2.3 Fault diagnosis methodology

As mentioned in above sub-section, the methodology used to diagnose a single actual fault location can be done by reversing the fault detection method. This can greatly reduce the number of configurations since the configurations used for fault detection will be reused to locate the fault. The principle for locating a single stuck-at fault is slightly different from that used to locate a single pair-wise bridging fault. Both these principles are presented in the following sub-sections.

#### 2.3.1 Single stuck-at fault

A design circuit with \(n\) nets can have \(2n\) stuck-at faults consisting of both stuck-at-0 and stuck-at-1. The technique presented in this section optimizes the method used in fault detection for single stuck-at fault.

**Theorem 1** For \(\log_2(n + 2) + 1\) configurations can precisely identify any single stuck-at fault location and its characteristic.

**Proof** Upper bound of \(\log_2(n + 2) + 1 \approx \log_2(2n)\), satisfying most situations with \(n\) not being a power of two values.

Since each of the net, \(n_i\) is assigned a different set of configurations for fault detection, they are uniquely different from one to another. Therefore, the fault detection process can be reversed to diagnose and locate the actual fault location as well as the characteristic of any given single stuck-at fault. As illustrated in Table 2, there are three main steps used to locate a single stuck-at fault and its fault nature. First step is based on those configuration files that pass for fault detection and the second step is based on those configuration files that fail for fault detection. These two files are then used for result manipulation to diagnose a single stuck-at fault. However, if the first two steps are not able to locate the actual location of a single stuck-at fault, then the procedure will proceed into the third step by generating additional configurations.

To demonstrate the proposed fault location and to determine its type (hereafter referred to as diagnosis), an application
1 for each syntax line of the design do
2 extract and delete sequential elements in the design
3 connect input of the sequential element to its output to complete the connection
4 \(N\), number of nets used in the design
5 for each \(\log_2(N + 2)\) counts do
6 \(v_i\), Walsh Code value assignment of net \(n_i\)
7 for each LUT \(l\) in the design do
8 generates single-term function for LUT \(l\) based on activating input of the LUT

| Step | Description |
|------|-------------|
| 1    | Based on configurations that pass fault detection, any value (both ‘0’ and ‘1’) for each net is a true value. |
| 2    | Based on configurations that fail fault detection, both ‘0’ and ‘1’ values that occur in the failed configurations for the same net indicate that they are a true value. |
| 3    | If more than one single stuck-at fault is detected, additional configurations will be generated for further narrowing down the single fault location. After that, step 1 and step 2 will be repeated. |

Design with three LUTs and eight nets as shown in Figure 2 is considered. The active vectors applied to the primary inputs of the circuit are as shown in Table 3. Firstly, based on the configuration that passed fault detection, all values (both ‘0’ and ‘1’) set for each net in the configurations are true values. This can be used to eliminate and narrow down the fault location and type of the fault as demonstrated in the last two rows of Table 3. As stated in the table, the possible fault locations are \(n_1/1, n_2/0, n_3/1, n_4/0, n_5/1, n_6/0, n_7/1\) and \(n_8/0\). This is based on the property that if a net is stuck-at-0, the configuration will fail when the configuration for that respective net is set at value ‘1’ and vice versa, a net stuck-at-1 will fail when the configuration is set at value ‘0’. The configurations that are used for the fault detection have been used in the first step of the fault location. Therefore, not generating new configurations for the first step of the fault location.

After the first step is completed, the single stuck-at fault diagnoses method proceeds into the second step of diagnosis (Table 2) where result manipulation is based on the configurations that failed for fault detection in the first step. The configurations that are used for fault diagnosis are same as the ones used in the first step of the fault location. Therefore, new configurations need not be generated for the second step of the fault diagnosis. This step compares all configurations that failed for the fault detection (first step). Whenever there are both ‘0’ and ‘1’ value set for one net in those failed configurations, differing values for the same net in different configurations indicate that the respective net does not have any stuck-at fault.

For example, let us consider the net \(n_2\) was diagnosed as stuck-at-0 fault during the first step. This net was subjected to both 1 and 0 during the different failed configurations, thus indicating that \(n_2\) is not suffering from a stuck-at fault. The same diagnostic result holds good for other nets \(n_3, n_4, n_5, n_6, n_7\) and \(n_8\). The exception however is \(n_1\), because \(n_1\) was not subjected to logic 1 during all the failed configurations. Therefore, the net \(n_1\) is detected as faulty and the type of the fault is characterized as stuck-at-1. This is illustrated in Table 4.

For the second step of the fault location and detection process, the proposed method utilized only the configurations that were used in the first step for detecting the fault. Hence unlike the previous methods \([1, 8, 24]\), additional configurations are not required to locate the faults.

Next, if more than one stuck-at faults are detected after executing step 1 and step 2, additional configuration needs to be generated and both steps 1 and 2 should be repeated to narrow down and determine the actual single fault location. For instance, Table 5 illustrates a situation where step 1 and step 2 are not enough to determine the single fault that occurred in nets \(n_7/1\) and \(n_8/0\). Here the fault detection fails for only one configuration but indicates that \(n_7\) has a stuck-at-1 fault and \(n_8\) has a stuck-at-0 fault. This can be easily determined from the repetitive patterns ‘0111’ and ‘1000’ as shown in the table.

\[\text{TABLE 1 Configuration generation algorithm for single-term function}\]

\[\text{TABLE 2 Methodology for locating single stuck-at fault}\]

\[\text{FIGURE 2 Application design with three LUTs and eight nets}\]
Hence, step 3 is introduced to narrow down and determine the actual faulty net. In order to determine which actual net is faulty, an additional configuration is generated. This is generated by setting one detected faulty net to ‘1’ while the other faulty net is set to ‘0’. This additional configuration will be sufficient to determine the actual faulty location as well as the characteristic of the fault. As shown in Table 6, the additional configuration shows that the fault is at the net n8.

However, dependent on the fault location, this third step for diagnosing fault can be avoided. Table 6 demonstrates step 3 with repetitive step 1 and step 2 to determine the actual faulty net. As a result, this proposed method for diagnosing single stuck-at fault reuses the same configuration used for detecting fault. Nevertheless, a minimal of $\log_2(n + 2)$ configurations are needed to determine the actual location as shown in the above example. Additional configuration may need to be generated with a total of $\log_2(n + 2) + 1$ configurations to locate the actual faulty net after both step 1 and step 2 completed.

### 2.3.2 Single pair-wise bridging fault

A design circuit with n nets can have $n(n-1)/2$ distinct pair-wise bridging faults. Thus, $\log_2 n(n-1)/2$ configurations are needed to locate the actual faulty pair [8] in addition to $\log_2(n + 2)$ configurations for fault detection. However, based on the proposed method, the total number of configurations can be reduced by re-using the same configurations from fault detection. Hence, the proposed method requires

| Primary input | Internal signal | Primary output | Fault detection result |
|---------------|----------------|---------------|------------------------|
| $n_1$ $n_2$ $n_3$ $n_4$ $n_5$ | $n_6$ $n_7$ | $n_8$ | Pass |
| 1 0 1 0 1 | 0 1 | 0 | Pass |
| 0 1 1 0 0 | 1 1 | 0 | Fail |
| 0 0 0 1 1 | 1 1 | 0 | Fail |
| 0 0 0 0 0 | 0 0 | 1 | Fail |
| $n_1/0$ $n_2/0$ $n_3/0$ $n_4/0$ $n_5/0$ | $n_6/0$ $n_7/0$ | $n_8/0$ | Fault: $n_3/1$ |
| $n_1/1$ $n_2/1$ $n_3/1$ $n_4/1$ $n_5/1$ | $n_6/1$ | $n_7/1$ | $n_8/1$ |

| Example of second step for single stuck-at fault diagnosis |
|---------------------------------------------------------|
| Primary input | Internal signal | Primary output | Fault detection result |
|---------------|----------------|---------------|------------------------|
| $n_1$ $n_2$ $n_3$ $n_4$ $n_5$ | $n_6$ $n_7$ | $n_8$ | Pass |
| 1 0 1 0 1 | 0 1 | 0 | Pass |
| 0 1 1 0 0 | 1 1 | 0 | Fail |
| 0 0 0 1 1 | 1 1 | 0 | Fail |
| 0 0 0 0 0 | 0 0 | 1 | Fail |
| $n_1/0$ $n_2/0$ $n_3/0$ $n_4/0$ $n_5/0$ | $n_6/0$ $n_7/0$ | $n_8/0$ | Fault: $n_3/1$ |
| $n_1/1$ $n_2/1$ $n_3/1$ $n_4/1$ $n_5/1$ | $n_6/1$ | $n_7/1$ | $n_8/1$ |

| Example of a situation where more than one stuck-at fault is diagnosed |
|---------------------------------------------------------------------|
| Primary input | Internal signal | Primary output | Fault detection result |
|---------------|----------------|---------------|------------------------|
| $n_1$ $n_2$ $n_3$ $n_4$ $n_5$ | $n_6$ $n_7$ | $n_8$ | Pass |
| 1 0 1 0 1 | 0 1 | 0 | Pass |
| 0 1 1 0 0 | 1 1 | 0 | Pass |
| 0 0 0 1 1 | 1 1 | 0 | Pass |
| 0 0 0 0 0 | 0 0 | 1 | Fail |
| $n_1/0$ $n_2/0$ $n_3/0$ $n_4/0$ $n_5/0$ | $n_6/0$ $n_7/0$ | $n_8/0$ | 2 faults detected |
| $n_1/1$ $n_2/1$ $n_3/1$ $n_4/1$ $n_5/1$ | $n_6/1$ | $n_7/1$ | $n_8/1$ |
log₂n(n − 1)/2 − log₂(n + 2) configurations to locate pair-wise bridging faults. For an example of eight nets, [8] uses nine configurations to determine the faulty pair while the proposed method requires only five configurations to detect and diagnose the pair-wise bridging faults.

The steps taken to locate a single pair-wise bridging fault are similar yet different compared to the method used in locating single stuck-at fault. It also uses the reversed method to diagnose and locate the fault location of a single pair-wise bridging fault. As shown in Table 7, there are three main steps used to locate the fault. First step is based on the configuration files that pass for fault detection and the second step is based on the configuration files that failed fault detection.

The proposed method for pair-wise bridging fault detection is similar to the proposed single stuck-at fault detection method where these two sets of configuration files (passed and failed) will be used for result manipulation to diagnose and locate the actual location of the pair-wise bridging fault. However, if these two main steps are not able to determine the actual location for a single pair-wise bridging fault, then this method too will proceed into a third step by generating extra configurations. In the first step, based on configurations that pass fault detection, each net are paired with different net to compare configuration value. However, if any specific pair that has same configuration value, that indicates that the pair of nets are free from pair-wise bridging fault as shown in Table 10. As can be seen from the table, n₂ and n₈ are having pair-wise bridging fault.

If more than a single pair-wise bridging fault is detected as demonstrated in Tables 11 and 12 (where the pair-wise bridging faults occur between the nets n₁n₄, n₂n₇ and n₃n₈), the proposed method generates additional configurations to narrow down the actual fault location and step 1 and step 2 are repeated. The additional configuration is shown in Table 11. With the additional configuration, the faults identified between the nets n₁n₄ and n₃n₈ are eliminated, resulting in n₂n₇ as the actual fault for pair-wise bridging. Thus, step 3 is introduced to narrow down and locate the actual fault location.

As a result, this proposed method for diagnosing the location of a single pair-wise bridging fault optimizes the same configuration used for detecting the fault. Nevertheless, a minimum of log₂(n + 2) configurations are needed to determine the actual faulty pair as shown in the above example. Additional configurations may need to be generated with a maximum of log₂ n(n − 1)/2 + 1 configurations to locate the actual faulty pair net.

### 2.3.3 Single fault diagnosis

The actual single fault can be either a single stuck-at fault or a single pair-wise bridging fault. An additional configuration to the number of fault detection configurations is generated to determine whether the actual fault is either a stuck-at fault or a pair-wise bridging fault.

For example, if the stuck-at fault diagnosis method is applied first to detect single stuck-at fault for a design before the pair-wise bridging fault diagnosis method, an additional configuration is generated to determine whether the actual detected single stuck-at fault is the true fault. On the other
hand, if the last configuration file determines that the last detected stuck-at fault is the actual fault, then the pair-wise bridging fault diagnosis method does not need to be executed since the actual fault is a stuck-at fault. However, if the last configuration file proves that the detected fault is not the actual fault, then the pair-wise bridging fault diagnosis method is executed to determine the actual fault.

This technique is applied for pair-wise fault diagnosis method as well. If the pair-wise fault diagnosis method is applied first for fault diagnosis, then an additional configuration is generated to determine whether the actual detected single pair-wise bridging fault is the true fault.

### 2.3.4 Multiple fault detection and diagnosis

The interconnect fault detection method proposed in Table 1 configures all the LUTs with single-term functions and applies activating vectors at the primary inputs such that each of the net, $n_i$, is assigned a different set of configurations for fault detection. Therefore, they are uniquely different from one to another, thus it is possible to detect multiple interconnect faults. This is illustrated using the circuit shown in Figure 3 and the results shown in Table 3 (multiple faulty nets are $n_1/1, n_2/0, n_3/1, n_4/0, n_5/1, n_6/0, n_7/1$ and $n_8/0$). Thus, the presented method is able to detect the multiple stuck-at faults using $\log_2(n + 2)$ test configurations. In order to locate and diagnose the multiple faults, the step 3 presented in Tables 3 and 7 need to be repeated depending on the number of faults detected. Thus, the number of configurations for the fault diagnose will be increasing depending on the number of detected faults. Therefore, by repeating the diagnosing step, it is possible to detect and diagnose multiple interconnect faults using the presented method.

### 3 SIMULATION RESULTS

For the implementation of the proposed method, the configuration file manipulation of benchmark circuits is carried out automatically using a script based on PERL language. The

---

**TABLE 7** Methodology for locating single pair-wise bridging fault

| Step | Description |
|------|-------------|
| 1    | Based on configurations that pass fault detection, if a pair of nets has different configuration value, that pair is free from pair-wise bridging fault. |
| 2    | Based on configurations that failed fault detection, if a pair of nets has same configuration value, that pair is free from pair-wise bridging fault. |
| 3    | If more than one pair-wise bridging faults are detected, additional configurations will be generated for further narrowing down the single fault location. After that, step 1 and step 2 will be repeated. |

**TABLE 8** Example of first phase result for fault detection

| Primary Input | Internal Signal | Primary Output | Fault Detection Result |
|---------------|----------------|---------------|-----------------------|
| $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ | $n_9$ | $n_{10}$ |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | Pass |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | Fail |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | Pass |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Fail |

---

**TABLE 9** Example of first step for single pair-wise bridging fault diagnosis

| $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ |
|-------|-------|-------|-------|-------|-------|-------|-------|
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| No    | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |

**TABLE 10** Example of second step for single pair-wise bridging fault diagnosis

| $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ | $n_8$ |
|-------|-------|-------|-------|-------|-------|-------|-------|
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
| Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   | Yes   |
most important part required for performing manipulations are the net connections as well as the Boolean function of LUTs. By using net connections and the configurations generated, Boolean functions can be manipulated for each LUT. After every data manipulation is made, the file is ready for file conversion by using 'xdl2ncd'.

The proposed method was tested on the ISCAS’89 benchmark circuits by implementing them on the Xilinx family FPGAs. Table 13 illustrates the number of test configurations required for the respective ISCAS’89 circuit design to detect and diagnose a single stuck-at fault and pair-wise bridging fault mapped into Xilinx XC3S500 E device.

The third column of Table 13 shows the maximum nets from respective benchmark circuit while the last three columns illustrate the maximum number of configurations needed to perform fault detection and for determining the fault location using the proposed method, and previous methods [1, 8]. It can be noted that the proposed method requires a lesser number of configurations when compared with the previous methods. This decrease in the number of configurations needed is more apparent with an increase in the number of nets subjected to test. Therefore, for testing circuits with a high number of nets, the proposed method will require a lesser number of test configurations when compared to the previous

| Circuit | LUTs | Max nets | Proposed | [1] | [8] |
|---------|------|----------|----------|-----|-----|
| S27     | 3    | 15       | 5        | 7   | 4   |
| S298    | 24   | 120      | 7        | 9   | 7   |
| S344    | 35   | 175      | 7        | 10  | 8   |
| S349    | 36   | 180      | 7        | 10  | 8   |
| S382    | 48   | 240      | 7        | 10  | 8   |
| S386    | 54   | 270      | 8        | 10  | 8   |
| S400    | 54   | 270      | 8        | 10  | 8   |
| S420    | 58   | 290      | 8        | 10  | 9   |
| S444    | 54   | 270      | 8        | 10  | 8   |
| S510    | 88   | 440      | 8        | 11  | 9   |
| S526    | 40   | 200      | 7        | 10  | 8   |
| S641    | 77   | 385      | 8        | 11  | 9   |
| S713    | 68   | 340      | 8        | 11  | 9   |
| S820    | 117  | 585      | 8        | 11  | 9   |
| S953    | 172  | 860      | 9        | 12  | 10  |
| S1196   | 193  | 965      | 9        | 12  | 10  |
| S1238   | 225  | 1125     | 9        | 12  | 10  |
| S1423   | 225  | 1125     | 9        | 12  | 10  |
| S1488   | 250  | 1250     | 9        | 12  | 10  |
| S1494   | 246  | 1230     | 9        | 12  | 10  |
| S5378   | 389  | 1945     | 9        | 13  | 11  |
| S9234   | 363  | 1815     | 9        | 13  | 10  |
| S15850  | 863  | 4315     | 10       | 14  | 12  |
| S35932  | 3185 | 15,925   | 11       | 16  | 13  |
| S38417  | 2393 | 11,965   | 11       | 15  | 13  |
| S38584  | 3464 | 17,320   | 11       | 16  | 14  |
methods. Overall, the proposed method shows a 28% reduction in the total number of test configurations needed to test all the benchmarks circuits when compared against the previous methods as shown in Table 13.

Table 13 demonstrates the maximum number of test configurations needed when each LUT has a maximum of five nets per LUT. Considering that a given circuit has \( n \) nets, the upper bound on the number of test configurations for fault detection and locating fault of the proposed method is

\[
\text{Max Configs} = \log_2(n + 2) + 1 + \left\lfloor \log_2\left(\frac{n(n-1)}{2}\right) \right\rfloor - \log_2(n + 2) \approx \log_2\left(\frac{n(n-1)}{2}\right) + 3
\]

where \( \log_2(n + 2) + 1 \) are the total configurations needed to detect and diagnose the actual fault for stuck-at fault, \( \log_2(n + 2) + \left[ \log_2\left(\frac{n(n-1)}{2}\right) - \log_2(n + 2) \right] + 1 \) are the total configurations needed to detect and locate the actual fault for pair-wise bridging fault and 1 additional configuration is generated to determine whether the actual fault is a single stuck-at fault or a single pair-wise bridging fault. Thus, the maximum number of configurations will be \( \log_2\left(\frac{n(n-1)}{2}\right) + 3 \) for fault detection and diagnosis.

Table 14 illustrates the number of test configurations required by the proposed method and the previous works [1, 8] when the ISCAS’89 circuit design was mapped into a Virtex XCV50 device while last three designs which were mapped into an XCV200 device. It can be noted that the proposed method requires a lesser number of test configurations when compared with the previous methods. This reduction is clearly discernible when testing circuits with the large number of test configurations. It can be observed that the proposed method reduces 48% of the test configurations when compared with the previous works in testing all the benchmark circuits shown in Table 14.

| Device name | Max LUT | Max nets | Proposed [1] | Proposed [8] |
|-------------|---------|----------|-------------|-------------|
| XC2V40      | 512     | 2560     | 7           | 13          | 11          |
| XC2V80      | 1042    | 15,120   | 7           | 14          | 12          |
| XC2V250     | 3072    | 15,360   | 8           | 16          | 13          |
| XC2V500     | 6144    | 30,720   | 9           | 16          | 14          |
| XC2V1000    | 10,240  | 51,200   | 9           | 17          | 15          |
| XC2V1500    | 15,360  | 76,800   | 9           | 18          | 15          |
| XC2V2000    | 21,504  | 107,520  | 10          | 18          | 16          |
| XC2V3000    | 28,672  | 143,360  | 10          | 18          | 16          |
| XC2V4000    | 46,080  | 230,400  | 10          | 19          | 17          |
| XC2V6000    | 67,584  | 337,920  | 11          | 20          | 17          |
| XC2V8000    | 93,184  | 465,920  | 11          | 20          | 18          |

The main contribution to this reduction is from \( \log_2(n + 2) \) fault detection configurations as the proposed method reuses the same configurations from fault detection for fault diagnosis as well. It is clearly shown that the proposed testing method is scalable and applicable for all FPGA devices and families.

The fault diagnosis method presented in [19] uses multiplexers to propagate the LUT output value. This method may detect and diagnose a false fault due to the existence of possible faults that may occur within multiplexer inputs and output. Another method presented in [2] reconfigures the switch matrix that connects one functional block to another. This method will not be able to cover 100% fault as pair-wise bridging fault are overlooked. Although these methods are able to detect and diagnose multiple faults, there are flawed due to their inability to detect pair-wise bridging faults and because they may uncover false faults, thereby making these techniques unreliable.

Another method that uses Boolean satisfiability to perform application-dependent interconnect testing of FPGA for different stuck-at, bridging, and open faults is presented in [22]. This technique requires same number of test configurations to test larger circuits when compared with the Walsh-based technique. This method however takes more time in generating test vectors.

4 | CONCLUSION

In this project, a new application-dependent FPGA interconnect testing method is presented. In this method, the number of configurations needed for interconnect fault detection and diagnosis is \( \log_2\left(\frac{n(n-1)}{2}\right) + 3 \) where \( n \) is the number of nets used. By using this technique, the maximum number of configurations can be significantly reduced by an average reduction of up to 48% when compared with the method defined in [1]. The proposed method also uses \( \log_2(n + 2) \) lesser number of configurations when compared with other previously documented methods. Moreover, the location of an actual single stuck-at or pair-wise bridging fault can be uniquely identified by reusing the same configuration that was used in the fault detection (with some additional configurations). Furthermore, the nature of a single stuck-at fault can also be determined by using this proposed method.

ORCID
T. Nirmalraj @ https://orcid.org/0000-0001-5293-4452

REFERENCES
1. Almurib, H.A.E., Kumar, T.N., Lombardi, F.: Scalable application-dependent diagnosis of interconnects of SRAM-based FPGAs. IEEE Trans. Comput. (2014)
2. New, C.E., Kumar, T.N.: An approach for exhaustive self testing of LUTs in an FPGA using Walsh configurations. In: 33rd International Electronics Manufacturing Technology Conference, pp. 1–6 (2008)
3. Engineering and Technology: These are the general terms and conditions for E&T competitions and giveaways, pp. 27–9 (2007). www.theiet.org/engtechmag
4. Kumar, T.N., Almurib, H.A.F., Chin-Ee, N.: Fine grain faults diagnosis of FPGA interconnects. Microproc. Microsyst. 31(1), 33–40 (2012)
5. Hsu, C.L., Chen, T.H.: Built-in self-test design for fault detection and fault diagnosis in SRAM-based FPGA. IEEE Trans. Instrum. Meas. 58(7), 2300–2315 (2009)
6. Brown, S., et al.: Field-Programmable Gate Arrays. Kluwer Academic Publishers (1992)
7. Kumar, T.N., Ch'ng, S.I.: An automated approach for the diagnosis of multiple faults in FPGA interconnects. Proc. of 11th IEEE International Symposium on Quality Electronic Design – Asia (2009)
8. Tahoori, M.B.: High resolution application specific fault diagnosis of FPGAs. IEEE Trans. VLSI Syst. 19(10), 1775–1786 (2011)
9. Huang, W.K., et al.: Testing configurable LUT-based FPGA's. IEEE Trans. VLSI Syst. 6(2), 276–283 (1998)
10. Huang, W.K., et al.: An approach for detecting multiple faulty FPGA logic blocks. IEEE Trans. Comput. 49(1), 48–54 (2000)
11. Parreira, A., Teixeira, J.P., Santos, M.: A novel approach to FPGA-based hardware fault modeling and simulation
12. Abramovici, M., Stroud, C.E.: BIST-based test and diagnosis of FPGA logic blocks. IEEE Trans. VLSI Syst. 9(1), 159–172 (2001)
13. Niamat, M.Y., Jogu, R.R.: A BIST scheme for testing a multiple FPGA system. In: Proc. of IEEE 45th Midwest Symposium on Circuits and Systems MWSCAS-2002, pp. 45–48 (2002)
14. Niamat, M.Y., Sahni, A., Jamali, M.M.: A build-in self test scheme for automatic interconnect fault diagnosis in multiple and single FPGA systems. Proc. of IEEE 50th Midwest Symposium on Circuits and Systems (2007)
15. Sun, X.L., et al.: Design and implementation of a parity-based BIST scheme for FPGA global interconnects, pp. 1251–1257
16. Liu, J., Simmons, S.: BIST-diagnosis of interconnect fault locations in FPGA's, pp. 207–210 (2003)
17. Alaghi, A., Varandi, M.S., Navals, Z.: An optimum ORA BIST for multiple FPGA look-up table testing, (2006)
18. Rozkovec, M., Jeniec, J., Novak, O.: Application dependent FPGA testing method. In: 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, pp. 525–530 (2010)
19. Kumar, T.N.: Fault propagation and diagnosis of LUTs in an FPGA without fault free assumptions. In: 2010 International Conference on Electronic Devices Systems and Applications, pp. 348–351 (2010)
20. Kumar, T.N., Almurib, H.A.F., Chin-Ee, N.: Fine grain faults diagnosis of FPGA interconnects. Microproc. Microsyst. 31(1), 33–40 (2012)
21. Kumar, T.N., Almurib, A.A.F., Lombardi, F.: Locating faults in application-dependent interconnects of SRAM based FPGA. In: Proc. of 30th IEEE International Conference on Computer Design, Canada, pp. 453–459 (2012)
22. Shukla, B., Roy, S., Sen, B.: Application-dependent testing of FPGA interconnect network. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27(10), 2296–2304 (2019)
23. Shukla, B., Roy, S.: Application dependent testing of FPGA interconnect using satisfiability modulo theory. In: 2018 3rd International Conference for Convergence in Technology (I2CT), pp. 1–5 (2018)
24. Kumar, T.N., Lombardi, F.: A novel heuristic method for application-dependent testing of a SRAM-based FPGA interconnect. IEEE Trans. Comput. 62(1), 163–172 (Jan. 2013)
25. Yang, Z., et al.: Testing FPGA local interconnects based on repeatable configuration modules (abstract only). In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 280 (2016)
26. Zhang, F., et al.: Research on hex programmable interconnect points test in island-style FPGA. Electronics. 9(12), 2177 (2020). MDPI.com. https://doi.org/10.3390/electronics9122177

How to cite this article: Nirmalraj T, Radhakrishnan S, Pandiy SK. Automatic diagnosis of single fault in interconnect testing of SRAM-based FPGA. IET Comput. Digit. Tech. 2021;1–10. https://doi.org/10.1049/cdt2.12028