Application-Dependent Diagnosis of FPGAs

Mehdi Baradaran Tahoori
Northeastern University
Boston, MA 02115
mtahoori@ece.neu.edu

Abstract

A new technique for diagnosis of faults in the interconnects and logic blocks of an arbitrary design implemented on an FPGA is presented. This work is complementary to application-dependent detection methods for FPGAs. This technique can uniquely identify any single bridging, open, or stuck-at fault in the interconnect as well as any single functional fault in the logic blocks. The number of test configurations for interconnect diagnosis is logarithmic to the size of the mapped design, whereas logic diagnosis is performed in only one test configuration.

1. Introduction

SRAM-based Field Programmable Gate Arrays (FPGAs) are two-dimensional arrays of configurable logic blocks (CLBs) and programmable switch matrices, surrounded by programmable input/output blocks on the periphery. FPGAs are widely used in many applications such as networking, storage systems, communication, and adaptive computing, due to their reprogrammability, flexibility, and reduced time-to-market.

The reprogrammability of FPGAs results in faster design and debug cycle compared to Application-Specific Integrated Circuits (ASICs). However, once the design is finalized and fixed, the programmability becomes useless and costly. This is why FPGAs are very costly for high volume fixed designs compared to ASICs. Nevertheless, the reconfigurability of FPGAs can be readily exploited for *defect tolerance* at manufacturing level as well as *fault tolerance* for user applications.

Application-dependent testing of FPGAs can be used by the manufacturer for defect tolerance in order to increase the manufacturing yield [Xilinx EasyPath] which in turn results in cost reduction. This is based on the fact that some FPGA chips that do not pass the application-independent test may be still usable for a particular design. In this case, the defects are located in some areas of the chip not used by that design. This application-specific test flow improves the manufacturing yield, and hence, reduces the costs for high-volume fixed designs.

During system operation, application-dependent test and diagnosis play a major role in online self-repair schemes for fault tolerant applications [Huang 01]. In these applications, the existence of faults in the system is first identified and faulty resources are precisely diagnosed afterwards. Then, the design is remapped to avoid faulty resources. Because test and diagnosis procedures are performed during system operation (online), the number of test vectors and configurations must be minimized. Note that the test time is dominated by loading test configurations rather than applying test vectors.

In this paper, application-dependent diagnosis techniques for logic and interconnect resources are presented. For interconnect diagnosis, the configuration of used logic blocks are modified, and the configuration of the interconnects remains unchanged. Any single fault (open, stuck-at, or bridging fault) in the interconnects can be uniquely identified in a small number of test configurations. For logic diagnosis, the configuration of used logic blocks remains unchanged while the configurations of the interconnect resources and unused logic blocks are modified. Any single functional fault, inclusive of all stuck-at faults, in logic blocks is precisely diagnosed in only one test configuration. As shown in this paper, these single-fault diagnosis techniques can be extended for multiple faults diagnosis.

Recently, the notion of reconfigurable molecular computing at nano scale has been introduced which share lots of similarities with conventional FPGAs [Collier 99][DeHon 03] [Goldstein 02]. Since these devices are more vulnerable to defects at manufacturing and during system operation, defect and fault tolerant techniques are essential parts of these systems. Since the presented techniques are suitable for very large designs, they can be used as detection and diagnosis steps for defect and fault tolerance of reconfigurable molecular computing systems.

The rest of this paper is organized as follows. In Sec.2, the previous work is presented. In Sec. 3, the diagnosis technique for interconnect faults is presented. In Sec. 4, the diagnosis method for faults in logic blocks is presented. Finally, Sec. 5 concludes the paper.

2. Previous Work

Application-independent (manufacturing) testing of FPGAs has been described in [Abramovici 00]

[Doumar 99] [Huang 98] [Michinishi 96] [Renovell 00] [Stroud 98, 96] [Sun 00] [Tahoori 03a]. These approaches target faults in the entire FPGA to ensure functionality of the device for any possible user configurations. Application-dependent testing of FPGAs has been addressed in [Das 99][Quddus 99] [Krasniewski 99, 02] [Renovell 01] [Harris 00a].

Diagnosis of faults in FPGA logic blocks has been discussed in [Abramovici 00][Inoue 98][Mitra 98][Stroud 97][Wang 97]. Diagnosis of faults in the FPGA interconnects has been addressed in [Das 99] [Harris 00b][Huang 96] [Liu 95] [Lombardi 96] [Yinlei 98]. A survey of detection, diagnosis, and fault tolerance techniques for FPGAs has been presented in [Doumar 03].

[Renovell 01] presents a new FPGA architecture with design for testability features. In the technique presented in [Das 99], every CLB used in the mapped design is reconfigured as transparent logic followed by flip-flips in order to construct scan chains. Also, fanout branches of a net are tested in different test configurations, i.e. dependent logic cones are tested in different configuration, resulting in a few number of test configurations. Due to complexity of configuration generation algorithm, it cannot be applied to large designs.

3. Interconnect Diagnosis

The interconnect resources in FPGAs can be categorized as *inter-CLB* and *intra-CLB* resources. Inter-CLB routing resources provide interconnections among CLBs. Inter-CLB resources include programmable switch blocks and wiring channels connecting switch blocks and CLBs. Intra-CLB resources are located inside each CLB. Intra-CLB interconnects include programmable multiplexers and wires inside CLBs. Diagnosing faults in inter-CLB routing resources is addressed in this section. For inter-CLB interconnect test and diagnosis, the configuration of routing resources remains unchanged while the configuration of logic resources is modified. Test and diagnosis of intra-CLB interconnects along with logic resources are discussed in Sec. 4. For this purpose, the configuration of used logic resources (inclusive of intra-CLB interconnects) is kept unchanged whereas the configuration of inter-CLB interconnects as well as unused logic resources are changed.

The separation between inter-CLB and intra-CLB is made because in contemporary FPGAs the programmable logic resources are not limited to LUTs; other logic resources such as carry generation/ propagation logic and cascade chains are included in CLBs. For inter-CLB interconnect test and diagnosis, these logic elements, if used in the original configuration, will be bypassed. This is discussed in the following subsections in details.

3.1 Single-Term Logic Networks

A *single-term function* is a logic function which has only one *minterm* or only one *maxterm*. In other words, the value of only one term in the truth table is different from the value of all other terms. The general form of a single-term function is a logic OR or logic AND function with some inversions (not necessarily) at the inputs and/or the output. The input corresponding to this specific minterm (or maxterm) is called *the activating input*. For a single-term function, if the applied input vector is the activating input, all sensitized faults are detected. An example is shown in Fig. 1, which is an OR function with inversions at the second and fourth inputs. This function has only one maxterm. Since the activating input (0101) is applied, *A/1* (*A* stuck-at 1 fault), *B/0, C/1*, and *D/0* are detected. Moreover the bridging faults between *A* and *B* ($A_{BF}B$), $A_{BF}D$, $B_{BF}C$, and $C_{BF}D$ are also detected.

This interesting testing property holds for any network of single-term functions. Consider a network of single-term functions *N*, and the input pattern *V*. If the values appear at the inputs of each gate (single-term function) are the activating inputs of that single-term function, all activated faults are detected. In other word, for each net *n* with value V_n, *n* stuck-at V_n' is detected, and for each pair of nets, n_i and n_j, with $V_{ni} \neq V_{nj}$, the bridging fault between n_i and n_j is detected [Tahoori 03b, 03c].

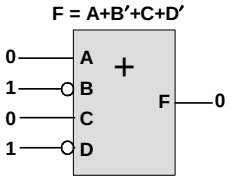

Figure 1. A single-term function with activating input pattern

For sequential networks, the extra requirement is to set the preset value of each flip-flop to the value of the activating input corresponding to its data input net. In other words, the initial state of the circuit should be the same as the combinational circuit (if each flip-flop is replaced by a wire) with the conditions described above. In this case, the required number of test clock cycles is equal to the maximum *sequential depth* of the network. The test vector must remain unchanged during all these test clock cycles. This condition guarantees that the value captured in the first flip-flop rank will be propagated and observed at the primary outputs. An example of a sequential circuit with single-term functions which satisfies all these conditions is shown in Fig. 2. The preset values of the

flip-flops are also shown. Here, two test clock cycles must be applied and the test vector (1101) must remain unchanged during these two clock cycles.

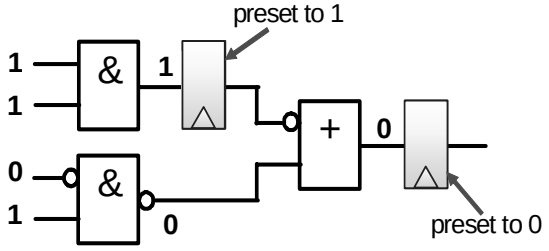

Figure 2. A sequential logic network of single-term functions

3.2. Test Configuration Generation

As explained in Sec. 3.1, single-term functions guarantee the detection of all activated faults. However, some mechanism is required to activate faults with respect to the fault list. We implement single-term functions in all used LUTs in the design. The question of which single-term function to implement in each LUT is addressed in this section, which gives us test configurations for inter-CLB interconnect testing.

Here, the idea of bus interconnect testing is used for activating the faults. In conventional board-level interconnect testing, only $\lceil \log_2(M+2) \rceil$ test vectors are used for testing all possible stuck-at, open, and pair-wise bridging faults for M interconnect bus lines [Kautz 76][Goel 82][Jarwala 89], provided that each bus is directly controllable and observable. Figure 3 shows the test vectors for 6 bus lines ($\lceil \log_2(6+2) \rceil = 3$). These vectors can be converted to the activating inputs of LUTs implementing single-term functions. Note that if the values of all input nets and the output net for an LUT are known, the single-term function to be implemented in that LUT is also known.

Since these test configurations target faults in inter-CLB interconnect, all additional logic resources in CLBs, if used, will be bypassed. Hence, CLBs are configured as LUTs followed by flip-flops (if those flip-flops are originally used in the user configuration). Here, we assume that nets extend from an LUT output to LUT input(s).

The pseudo code for the test configuration generation algorithm is shown in Fig. 4. These set of test configurations guarantee the detection of all stuck-at, open, and bridging faults (all pairs) in the interconnects. Since this technique detects all possible pair-wise bridging faults, there is no need to extract probable bridging fault list from the layout information using time-consuming inductive fault analysis methods. However, the number of test configurations can be further reduced if a particular fault list is used, since the number of test configurations is in logarithmic order of the number of faults in the fault list.

Figure 3. Complete test set for 6 buses

1. N ← number of nets in design
2. for each $\lceil \log_2(N+2) \rceil$ counting sequences **do**
3. $v_i \leftarrow$ value of net n_i in the code
4. **if** n_i is a primary input **then**
5. v_i is the corresponding bit for test vector
6. **for** each LUT l in the design **do**
7. $n^l_1,\ldots,n^l_m \leftarrow$ inputs of LUT l
8. $n^l_o \leftarrow$ output of LUT l
9. $F^l \leftarrow$ single-term function with
10. Activating inputs $v^l_1,\ldots,v^l_m$ and output v^l_o
11. **if** LUT l is driving a flip-flip f **then**
12. preset value of $f \leftarrow v^l_o$

Figure 4. Test vector and configuration generation algorithm

As an example, consider a design shown in Fig. 5 with 4 LUTs and 12 nets. Figure 6 shows the test vectors and configurations generated using the above algorithm for this design. The values of activating inputs for the LUT in each test configuration are also shown.

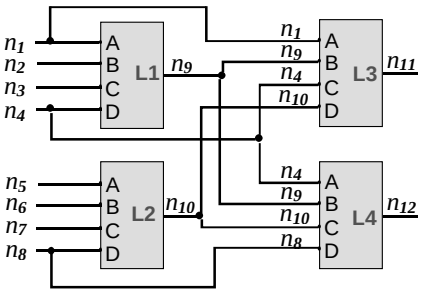

Figure 5. A design with 4 LUTs and 12 nets

3.3. Diagnosis Procedure

Assume that the outcome of each test configuration at the tester is a pass/fail result, i.e. the worst case condition is considered in which no further information regarding the failing outputs or test clock cycles is available. Note that this problem is different from conventional diagnosis approaches for bus

interconnects in which all nets are fully observable [Kautz 74][Wagner 87][Shi 95]. This special diagnosis problem in which all (internal) nets have full controllability (using single-term functions, it is possible to set any desirable value to each net) but limited observability (instead of observing the value of each net, only the pass/fail outcome can be obtained). Having considered this assumption, to precisely diagnose one single fault out of n distinct faults, at least $\lceil \mathbf{log}_2 n \rceil$ pass/fail outcomes are required.

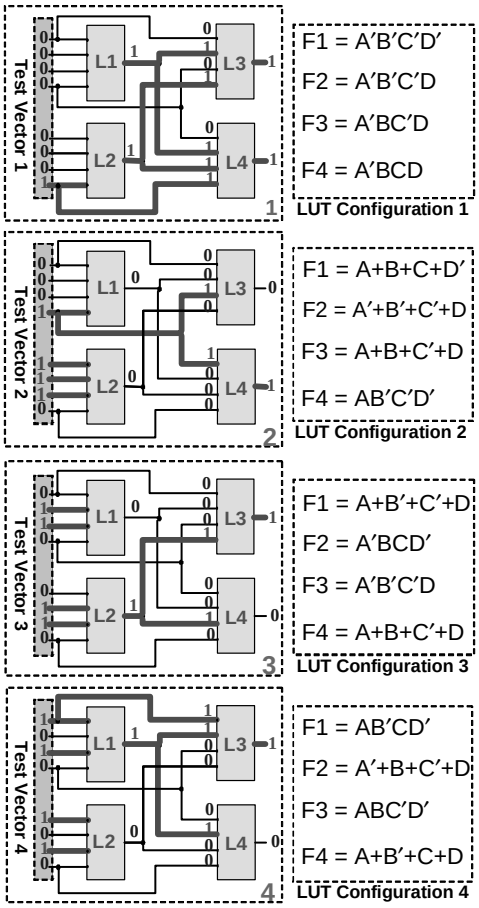

Figure 6. Test vectors and configurations for the circuit shown in Fig. 5

Diagnosis procedures are categorized into *adaptive* and *non-adaptive* approaches. In an adaptive procedure, the choice of the next step (test configuration) is based on the pass/fail outcome of the previous step. In a non-adaptive procedure, all steps are performed first (all test configurations applied and tester outcomes are gathered) and the faulty elements are determined based on the failing pattern. The *failing pattern* for m test configurations contains m bit, each bit position is 1 if and only if the corresponding configuration fails at the tester.

Non-adaptive approaches are preferable over adaptive ones since the total test application time for non-adaptive procedures are typically smaller than that for adaptive ones.

The following subsections describe the proposed diagnosis procedures based on various fault models.

3.3.1 Diagnosis of Stuck-at faults

A circuit with n nets has $2n$ stuck-at faults. Based on the above assumption, in order to uniquely identify any single stuck-at fault at least $\mathbf{log}_2 2n = 1 + \mathbf{log}_2 n$ test configurations are required.

For example, consider the circuit shown in Fig. 5 assuming that there is a stuck-at-0 fault on net n_9. Using four ($\lceil \mathbf{log}_2 12 \rceil$) test configurations presented in Fig. 6, the *failing pattern* is 1001 (first and fourth test configurations fail while second and third pass) which uniquely codes n_9 (9) in (4-bit) binary representation. Note that no two stuck-at-0 faults have the same fault pattern. However, the failing pattern for n_6 stuck-at-1 (1001) is exactly the same for n_9 stuck-at-0. This is why one extra test configuration is required to uniquely diagnose all stuck-at-0 and stuck-at-1 faults. This extra test configuration activates only stuck-at-1 faults. Based on the outcome of this test configuration, it can be determined if the failing pattern is encoding stuck-at-1 or stuck-at-0 faults. The diagnosis procedure for single stuck-at faults is as follows. Note that this procedure is non-adaptive.

1. The first configuration is all_OR in which the activating inputs for all LUTs are all-0 (all LUTs implement the OR function). This configuration activates all stuck-at-1 faults.

2. The remaining $\lceil \mathbf{log}_2 n \rceil$ configurations are the same as Sec. 3.2 (the algorithm presented in Fig. 4).

2.1. If the first configuration doesn't fail, the failing pattern for the remaining $\lceil \mathbf{log}_2 n \rceil$ configuration uniquely identifies the net with stuck-at-0 fault.

2.2. Otherwise, if the first configuration fails, the complement of the failing pattern of $\lceil \mathbf{log}_2 n \rceil$ configurations identifies the net with stuck-at-1.

For example if there is n_7 stuck-at-1 fault in the circuit shown in Fig. 5, the first configuration fails (since activates any stuck-at-1 fault), and the failing pattern for the next 4 configurations is "1000", whose complement (0111) codes n_7 (7).

3.3.2 Diagnosis of Open faults

An open fault on a net can be detected by testing for both stuck-at-0 and stuck-at-1 faults on that net. In other words, an open fault behaves as both stuck-at-1 and stuck-at-0 faults on the same net [Goel 82].

As mentioned in Sec. 3.3.1, the first step of stuck-at fault diagnosis consists of the all-OR test

configuration. If the all-AND configuration (the dual of all-OR) is also applied and both of them fail, it can be concluded that there is an open fault (based on single fault assumption). In this case, by applying the remaining $\lceil \mathbf{log}_2 n \rceil$, the failing pattern uniquely identifies the net with open fault

However, if none of the all-OR and the all-AND configurations fails, it can be concluded that the fault is neither stuck-at nor open. Hence, it should be a bridging fault (since no bridging fault is activated in either the all-OR or the all-AND configurations). This situation is addressed in the next subsection in more details.

3.3.3 Diagnosis of Bridging faults

The bridging fault list for a circuit with n nets contains $n(n-1)/2$ distinct pair-wise bridging faults. Hence, at least $\mathbf{log}_2[n(n-1)/2] \approx 2\mathbf{log}_2 n - 1$ test configurations are required for single bridging fault diagnosis.

The presented technique in this section is an adaptive approach in which the determination of the next test configuration depends on the pass/fail outcome of the previous test configuration.

Theorem. For n nets, $w_0, \ldots, w_n$, $2\lceil \mathbf{log}_2 n \rceil - 1$ adaptive steps can precisely identify any single two-net bridging fault.

Proof. It is based on an induction on the number of nets. For the sake of simplicity, assume n is a power of two (the proof holds for any arbitrary value for n).

The first step sets $w_0, \ldots, w_{n/2-1}$ (subset 1) to 0 and $w_{n/2}, \ldots, w_n$ (subset 2) to 1. This activates $(n/2)(n/2) = n^2/4$ of bridging faults and the remaining $n(n-1)/2 - n^2/4 = n^2/4 - n/2$ faults are not activated.

If this step fails (i.e. there is a bridging fault between a net in subset 1 to a net in subset 2), the left sub-tree is a complete binary tree to uniquely identify one of $n^2/4$ faults in other $\mathbf{log}_2 (n^2/4) = 2\mathbf{log}_2 n - 2$ steps (using binary search).

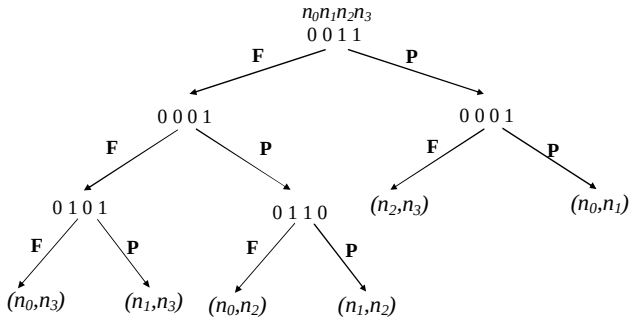

Figure 7. Adaptive steps for 4 nets: 3 steps

Otherwise, the fault is either within subset 1 $\{w_0, \ldots, w_{n/2-1}\}$ or subset 2 $\{ w_{n/2}, \ldots, w_n \}$. Diagnosing faults in subset 1 corresponds to the original problem with $n/2$ nets. Also, search within subset 1 is similar to search within subset 2 and can be performed in parallel. Hence, the right sub-tree can be constructed from the tree obtained from the solution to the problem of size $n/2$, and the duplication of the patterns (the first half corresponds to subset 1, and the second half corresponds to subset 2) at each node of that tree (*recursive construction*). Based on the step of the induction, this sub-tree has a depth of $2\mathbf{log}_2(n/2) - 1 = 2\mathbf{log}_2 n - 3$. However, since the pattern at each node is a duplication of that for $n/2$ nets, each leaf corresponds to two faults, one in subset 1 $\{w_0, \ldots, w_{n/2-1}\}$ and the other one in subset 2 $\{w_{n/2},\ldots, w_n \}$. Hence, an additional step is required to distinguish between the two faults at each leaf (by activating only one of those two).

It has been shown that each sub-tree has a depth of

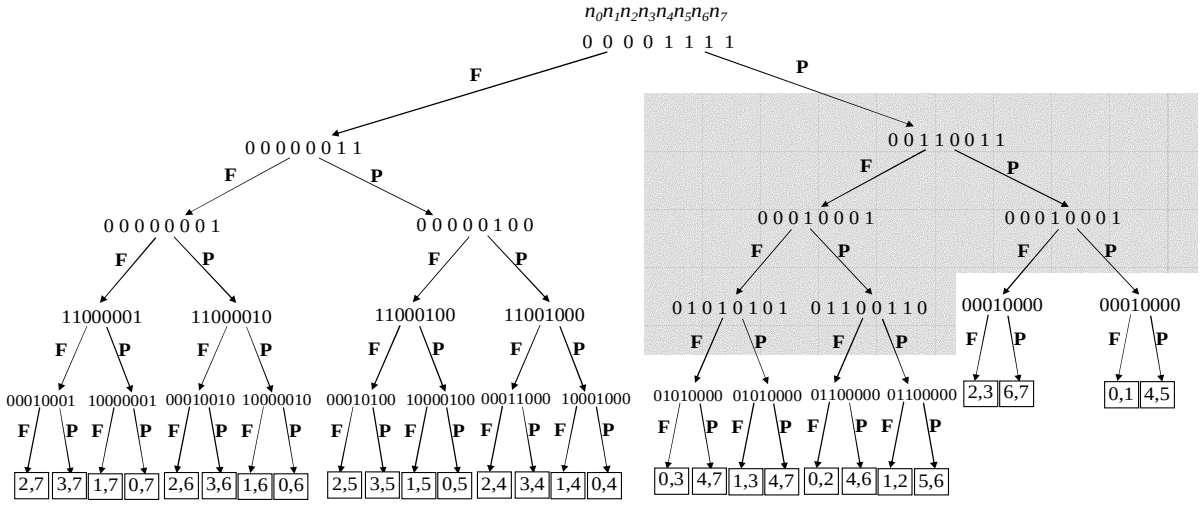

Figure 8. Adaptive steps for 8 nets: 5 steps

$2\log_2 n - 2$, so the depth of the decision tree is $2\log_2 n - 1$. Moreover, each leaf correspond to one unique bridging fault. Figure 7 shows the decision tree for 4 nets, which can be used as the basis for the induction. Hence the proof is complete. □

The example for the formation of the tree for 8 nets is shown in Fig. 8. The right sub-tree is the tree for 4 nets (Fig. 7) in which the pattern at each node is duplicated (shaded in this figure), followed by an additional step.

Note that the lower bound on the depth of a decision tree with m leaves (outcomes of a decision) is $\lceil \log_2 m \rceil$. In this problem, the number of possible outcomes is $n(n-1)/2$, hence the lower bounds on the number of steps is $\log_2 [n(n-1)/2] \approx 2\log_2 n - 1$. Since the presented approach meets the lower bound, it is the optimum algorithm in terms of the number of required steps.

The patterns in the decision tree can be converted to the activating input values for LUTs to obtain test vectors and configurations in the similar way presented in Sec. 3.2.

3.3.4 Overall Diagnosis Procedure

The pseudo code for the entire detection and diagnosis flow is shown in Fig. 9. The pass/fail outcomes of the all-OR and the all-AND test configurations determine the type of fault, namely stuck-at-0, stuck-at-1, open, and bridging faults. In the worst case, the total number of test configurations is equal to $3\lceil \log_2 n \rceil + 1$ to detect all open, stuck-at, and bridging faults, as well as diagnosis of all single faults.

1. Apply $\lceil \log_2 n \rceil$ test configurations (Sec. 3.2)
2. Obtain failing pattern $P = p_0 \ldots p_{\lceil \log_2 n \rceil}$
3. if diagnosis is required **then**
4. Apply all-OR configuration C_{OR}
5. Apply all-AND configuration C_{AND}
6. **if** $P_{OR}P_{AND} = 01$ **then**
7. P corresponds to net with ***stuck-at-0*** fault
8. **else if** $P_{OR}P_{AND} = 10$ **then**
9. P′ corresponds to net with ***stuck-at-1*** fault
10. **else if** $P_{OR}P_{AND} = 11$ **then**
11. P corresponds to net with ***open*** fault
12. **else** /*$P_{OR}P_{AND} = 00$ */
13. Apply $2\lceil \log_2 n \rceil - 1$ adaptive steps to diagnose ***bridging*** fault (Sec. 3.3.3)

Figure 9. Pseudo code for the detection and diagnosis flow

After faulty nets are diagnosed, if the exact failing interconnect resources (line segment or programmable switch) are required to be identified, similar techniques to those presented in [Tahoori 02] can be exploited afterwards.

3.4. Results

Consider an FPGA with N LUTs, such that each LUT has K inputs. The maximum number of nets for any designs to be mapped into this FPGA is $N\times(K+1)$. This means that one separate net is associated with every input and the output of each LUT in the FPGA. Using this upper bound on the number of nets, the upper bound on the number of test configurations for both detection and diagnosis is:

$$\mathbf{Max}_{\text{configs}} = 3\log_2[N\times(K+1)+2]+1.$$

Table 1 shows upper bounds on the number of test configurations for the largest designs mapped into Xilinx Virtex II FPGAs [Xilinx 03]. Note that these LUTs have 4 inputs ($K = 4$). The column denoted by "Max Bridging Faults" shows the number of all pair-wise bridging faults associated with the maximum number of nets.

As mentioned before, the number of test configurations for bridging fault diagnosis can be reduced if a smaller fault list is used. Note that a considerable number of $n(n-1)/2$ bridging faults (in a design with n nets) cannot happen based on physical layout information using inductive fault analysis (IFA) techniques [Ferguson 88]. If such faults are removed from the fault list, the number of test configurations can be reduced in a logarithmic scale.

Table 1. Maximum number of test configurations for Xilinx Virtex II FPGAs

Device Name	LUTs	Max Nets	Max Bridging Faults	#Config (detect& diagnosis)
XC2V40	516	2580	3.3 M	**37**
XC2V80	1024	5120	13.1 M	**40**
XC2V250	3072	15360	118 M	**43**
XC2V500	6144	30720	472 M	**46**
XC2V1000	10240	51200	1310 M	**49**
XC2V1500	15360	76800	2949 M	**52**
XC2V2000	21504	107520	5780 M	**52**
XC2V3000	28672	143360	10276 M	**55**
XC2V4000	46080	230400	26542 M	**55**
XC2V6000	67584	337920	57095 M	**58**
XC2V8000	93184	465920	108540 M	**58**

4. Logic Block Diagnosis

4.1. Linear Compactor for Diagnosis

For logic block (including intra-CLB interconnects) testing and diagnosis, the configuration of the original used logic blocks is preserved while the configuration of interconnects and unused logic blocks are changed to exhaustively test and diagnose all used logic blocks.

The idea of application-dependent logic block testing is presented in [Tahoori 04]. In this technique, a linear feedback shift register (LFSR) or a binary counter is connected to the inputs of all used logic block generating test vectors. The outputs of all these logic blocks are connected to a response compactor (e.g. an XOR tree) to generate the pass/fail signal. The LFSR and the XOR tree are implemented in the available unused logic blocks. Since the LFSR generates all possible patterns (2^n patterns for an n-input logic block) and the XOR tree propagates any single fault to its output, any single *functional* fault in the used logic blocks will be propagated to the output of the XOR tree and will be detected. Functional faults are any faults that change the truth-table of an LUT, including stuck-at faults. An example of this scheme is shown in Fig. 10.

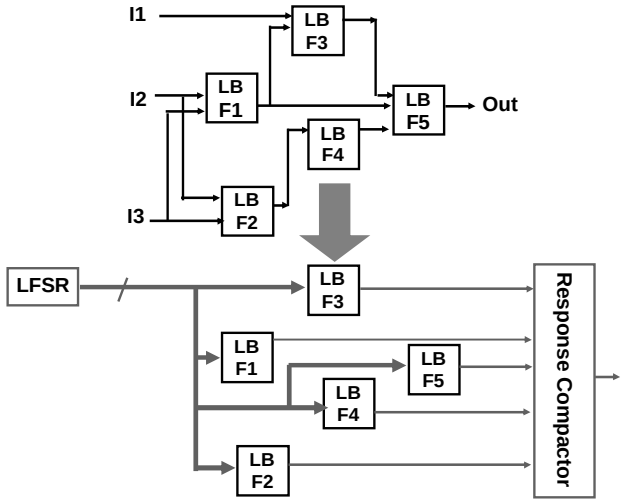

Figure 10. Application-dependent self-test architecture for logic blocks

However, the classical XOR tree doesn't provide any diagnosis capabilities. In order to improve the diagnostic resolution of this scheme, combinational compactor based on error correcting schemes can be exploited instead of the XOR tree. If a compactor with more than one output is used and logic block outputs are selectively connected to the compactor outputs (through the network of XOR gates), the failing pattern at the outputs of the compactor can identify failing logic block(s). Since exhaustive test patterns are applied to the inputs of each logic block, the exact faulty resource(s) inside the failing logic block(s) (LUT, flip-flop, intra-CLB interconnects, etc) can be uniquely diagnosed based on a fault dictionary.

The compactor circuitry can be represented as a binary *parity matrix* (matrix with only 0s and 1s) with n rows and m columns. Each row of the matrix corresponds to a logic block output and each column corresponds to a compactor output. The entry in row i and column j of the parity matrix is 1 if and only if the j^{th} compactor output depends on the output of the i^{th} logic block; otherwise, the entry is 0. The XOR tree can be represented by a vector (an $n\times1$ matrix) with 1 in all entries. For single fault diagnosis, the compactor must have at least $\lceil \mathbf{log}_2(n+1) \rceil$ outputs to differentiate the fault-free situation and n distinct (single) faulty logic blocks, i.e. $m = \lceil \mathbf{log}_2(n+1) \rceil$. As long as all rows are non-zero and distinct, any single fault can be detected and uniquely diagnosed. The easiest structure would be the counting sequence: the rows of this matrix correspond to binary numbers 1 to n represented in $\lceil \mathbf{log}_2(n+1) \rceil$ bits (columns).

To achieve higher diagnosis resolution, more sophisticated coding schemes can be used. For example, if every row of the parity matrix is non-zero, distinct, and contains an odd number of 1s, it can precisely identify all single faults and also distinguish the occurrence of double faults from single faults [Saluja 83]. One example of a matrix with this property for 8 logic blocks and the corresponding compactor circuit are shown in Fig. 11.

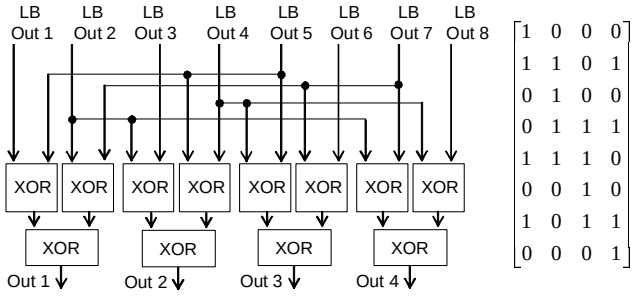

Figure 11. Parity matrix and the corresponding compactor circuit with 8 inputs and 4 outputs

One problem with this approach (even with the XOR trees) could be the routing congestion since the test signals (LFSR outputs) must be routed to the inputs of all used logic blocks. This could be a potential problem for very large designs. In order to solve this problem, the presented BIST architecture can be partitioned: instead of connecting outputs of one LFSR to all logic blocks, multiple LFSRs can be used and the outputs of each LFSR are connected to only a subset of logic blocks. Multiple compactors or one hierarchical compactor can be also used depending on the availability of I/O pins. Note that using multiple compactor increases the diagnosis resolution since multiple faults causing single or double faults in each partition can be precisely diagnosed. The example of partitioning using multiple LFSRs and compactors is shown in Fig 12 (F1,.., F9 are the original used logic blocks). The number and the structures of the partitions can be determined based on the routing

constraints, the availability of spare logic resources and the required diagnostic resolution.

When the LFSR and the compactor circuitry cannot fit into the unused logic resources, multiplexers and demultiplexers can be used for sharing the compactor and the LFSR, respectively.

However in the worst case, if due to the routing complexity of the original design, the test signal cannot be routed to all logic blocks, the used logic blocks can be partitioned into two subsets and each subset can be tested and diagnosed in a separate test configuration. So, in the worst case two test configurations are required for testing and diagnosis of logic blocks. In this case, the number of used logic blocks in each test configuration is always less than the total number of logic blocks in the FPGA. Hence, the LFSR and the compactor circuitry will definitely fit in unused logic blocks.

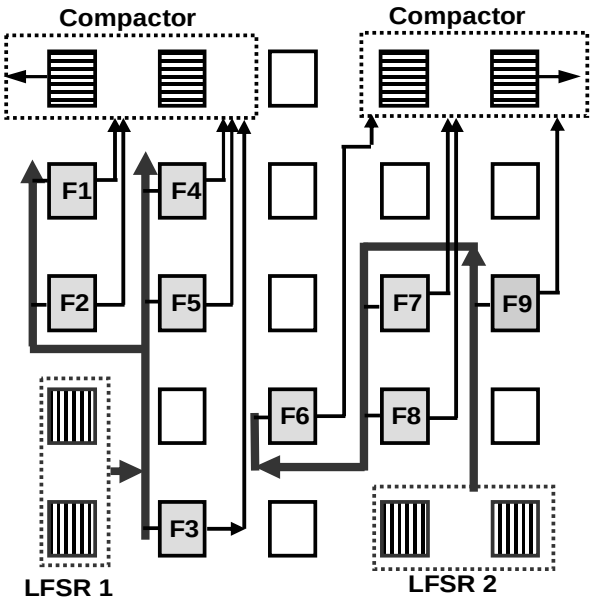

Figure 12. Test partitioning in logic BIST

4.2. Multiple Fault Diagnosis

The diagnosis technique presented for single faults can be extended for multiple faults diagnosis, as well. In a parity matrix, if the pair-wise sums (bitwise XOR) of any two rows are distinct, then any double faults can be uniquely diagnosed. Formally:

$$\forall i, j, k, l \mid i \neq j, k \neq l, j \neq k, R_i + R_j \neq R_k + R_l$$

where R_n's are rows of the parity matrix. Since double faults affect two inputs, the syndrome of any double faults should be different from that for other double faults. The above condition can be rewritten as:

$$\forall i, j, k, l \mid i \neq j, k \neq l, j \neq k, R_i \neq R_j + R_k + R_l$$

This basically means that any row should not be equal to the sum of any three other (distinct) rows. Since there are *n(n-1)/2* different double faults for *n* inputs, the lower bound for *m*, the number of columns, is equal to $\lceil \mathbf{log}_2 n(n-1) \rceil - 1$.

If $R_j = \underline{\mathbf{0}}$, then $\forall i, k, l \mid i \neq k \neq l, R_i \neq R_k + R_l$. In other words, if the all-0 row is included in the code, the syndrome of any single fault is different from the syndrome of any double faults. Hence, all single and double faults can be diagnosed. Since, all rows of the parity matrix should be non-zero, this all-zero row is only used for obtaining the code with the above property and it is not included in the parity matrix.

The parity matrix shown in Fig. 13 contains eight rows and five columns. Any row is different from the sum of three other rows. Since the first row is $\underline{\mathbf{0}}$, sum of any two rows is distinct from any single row. Hence, this parity matrix corresponds to a compactor which can precisely diagnose up to two faults in the inputs. Note that the first row, $\underline{\mathbf{0}}$, should be excluded from the parity matrix for the compactor implementation (the corresponding compactor has seven inputs and five outputs). Since $\lceil \mathbf{log}_2(8\times7)/2 \rceil = 5$, this compactor is optimum (the number of output pins is minimum).

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 13. Parity matrix for single and double fault diagnosis (the first row will be excluded)

This idea can be easily extended to other multiple faults. For example for triple faults, the sum of any three distinct rows should be distinct. Alternatively, no rows should be equal to the sum of five other distinct rows.

5. Summary and Conclusion

In this paper, application-dependent diagnosis techniques for faults in the interconnects and logic blocks of an arbitrary design mapped into an FPGA are presented. For interconnect diagnosis, single faults (open, stuck-at, or bridging fault) can be uniquely identified. As shown in the paper, the number of total test configurations for interconnects diagnosis is

logarithmic to the size of the design. For logic block diagnosis, single faults can be uniquely identified in only one extra test configuration. The presented technique can be modified for multiple fault diagnosis.

This method can be used for defect tolerance by the manufacturer in order to increase the manufacturing yield, or as a part of online self-repair schemes for fault tolerant applications. Moreover, since the presented techniques are suitable for very large designs, they can be used as detection and diagnosis steps for defect and fault tolerance of reconfigurable molecular computing systems which share similarities with conventional FPGAs [Collier 99][DeHon 03] [Goldstein 02].

References

[Abramovici 00] M. Abramovici, C. Stroud, *BIST-Based Detection and Diagnosis of Multiple Faults in FPGAs*, Proc. Int'l Test Conf., 2000.

[Collier 99] C.P.Collier, E.W.Wong, M.Belohradsky, F.M.Raymo, J.F.Stoddart, P.J.Kuekes, R.S.Williams, J.R.Heath, *Electronically Configurable Molecular-Based Logic Gates*, Science, vol 285, pp. 391-394, 1999.

[Das 99] D. Das, N. A. Touba, *A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems*, Proc. of Int'l Conf. On VLSI Design, 1999.

[Dehon 03] A. DeHon, *Array-Based Architecture for FET-Based, Nanoscale Electronics*, IEEE Trans. on Nanotechnology, Volume 2, Number 1, Pages 23--32, Mar 2003.

[Doumar 03], A. Doumar, H. Ito, *Detecting, Diagnosing, and Tolerating Faults in SRAM-based Field Programmable Gate Arrays, A Survey*, IEEE Trans. On VLSI Systems, vol. 11, Issue 3, pp. 386 – 405, 2003.

[Doumar 99], A. Doumar, H. Ito, *Testing the logic cells and interconnect resources for FPGAs*, Proc. Asian Test Conf., pp. 369–374, 1999.

[Ferguson 88] F.J. Ferguson, J.P. Shen, *A CMOS fault extractor for inductive fault analysis*, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7 Issue: 11, pp. 1181-1194, Nov. 1988.

[Goel 82] P. Goel, and M. T. McMahon, *Electronic Chip-in Place Test*, Proc. of Int'l Test Conf., pp. 83-90, 1982.

[Goldstein 02] S.C. Goldstein,D.Rosewater, *Digital Logic Using Molecular Electronics*, Proc. IEEE Int'l Solid-State Circuits Conf., 2002.

[Harris 00a] I. G. Harris, R. Tessier, *Interconnect Testing in Cluster-Based FGPA Architectures*, Proc. DAC, pp. 49–54, 2000.

[Harris 00b] I. G. Harris, R. Tessier, *Diagnosis of Interconnect Faults in Cluster-Based FGPA Architectures*, Proc. ICCAD, pp. 472–475, 2000.

[Huang 01] Huang, W.-J., and E.J. McCluskey, *Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance*, Proc. of IEEE Symp. on Field-Programmable Custom Computing Machines, 2001

[Huang 98] W. K. Huang, F.J. Meyer, X. T. Chen, F. Lombardi, *Testing configurable LUT-based FPGAs*, IEEE Trans. On Very Large Scale Integration Systems, Vol. 6, pp. 276-283, 1998.

[Huang 96] W. K. Huang, X. T. Chen, F. Lombardi, *On the Diagnosis of programmable Interconnect Systems: Theory and Application*, Proc. of VLSI Test Symp., pp. 204-209, 1996.

[Inoue 98] T. Inoue, S. Miyazaki and H. Fujiwara, *Universal Fault Diagnosis for Lookup Table FPGAs*, IEEE Design and Test of Computers, pp. 39-44, January-March, 1998.

[Jarwala 89] H. Jarwala, and C. W. Yau, *A New Framework for Analyzing Test Generation and Diagnosis Algorithms for Wiring Networks*, Proc. of Int'l Test Conf, pp. 63-70, 1989.

[Liu 95] T. Liu, F. Lombardi, and J. Salinas, *Diagnosis of Interconnects and FPICs Using a Structured Walking-1 Approach*, Proc. VLSI Test Symp., 1995, pp. 256-261.

[Lombardi 96] Lombardi, F., D. Ashen, X. Chen, and W.K. Huang, *Diagnosing Programmable Interconnect Systems for FPGAs*, Proc. Int'l Symp. on FPGAs, pp. 100-106, 1996.

[Kautz 74] W. H. Kautz, *Testing for Faults in Wiring Networks*, IEEE Trans. On Computers, vol. C-23, No. 4, pp. 358-363, 1974.

[Michinishi 96] H. Michinishi, T. Yokohira, T. Okamoto, *A Test Methodology for Interconnect Structures of LUT-Based FPGAs*, Proc. of Asian Test Symp., pp. 68-74, 1996.

[Mitra 98] S. Mitra, P.P. Shirvani, and E.J. McCluskey, *Fault Location in FPGA-Based Reconfigurable Systems*, Proc. IEEE Intl. High Level Design Validation and Test Workshop, 1998.

[Quddus 99] W. Quddus, A. Jas, N. A. Touba, *Configuration Self-Test in FPGA-Based Reconfigurable Systems*, Proc. ISCAS'99, pp. 97-100, 1999.

[Renovell 01] M. Renovell, P. Faure, J.M. Portal, J. Figueras, Y. Zorian, *IS-FPGA: A New Symmetric FPGA Architecture with Implicit SCAN*, Proc. IEEE Int'l Test Conf., pp. 924-931, 2001.

[Renovell 00] M. Renovell, Y. Zorian, *Different Experiments in Test Generation for XILINX FPGAs*, Proc. of Int'l Test Conf., 2000.

[Saluja 83] Saluja, K.K., and M. Karpovsky, *Testing Computer Hardware through Data Compression in Space and Time*, Proc. Int'l. Test Conf., pp. 83-89, 1983.

[Shi 95] W. Shi, W. Kent Fuchs, *Optimal Interconnect Diagnosis of Wiring Networks*, IEEE Trans. On VLSI, Vol. 3, no. 3, pp. 430-436, 1995.

[Stroud 98] C. Stroud, S. Wijesuriya, C. Hamilton, M. Abramovici, *Built-in self-test of FPGA interconnect*, Proc. Int'l Test Conf., pp. 404-411, 1998.

[Stroud 97] C. Stroud, E. Lee, and M. Abramovici, *BIST Based Diagnostics of FPGA Logic Blocks*, Proc. Int'l Test Conf, pp. 539-547, 1997.

[Stroud 96] C. Stroud, S. Konala, C. Ping, M. Abramovici, *Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)*, Proc. of VLSI Test Symp., pp. 387 –392, 1996.

[Sun 00] X. Sun, J. Xu, B. Chan, P. Trouborst, *Novel Technique for Built-In-Self Test of FPGA Interconnects*, Proc. of Int'l Test Conf., 2000.

[Tahoori 04] M. B. Tahoori, E. J. McCluskey, M. Renovell, P. Faure, *A Multi-Configuration Strategy for an Application Dependent Testing of FPGAs*, proc. VLSI Test Symp., 2004

[Tahoori 03a] M. B. Tahoori, S. Mitra, *Automatic Test Configuration Generation for FPGA Interconnect Testing*, Proc. IEEE VLSI Test Symposium, 2003.

[Tahoori 03b] M. B. Tahoori, *Using Satisfiability in Application Dependent Testing of FPGA Interconnects*, Proc. Design Automation Conference, pp. 678-681, 2003.

[Tahoori 03c] M. B. Tahoori, *Application Dependent Testing of FPGA Interconnects*, Proc. Defect and Fault Tolerance of VLSI, 2003.

[Tahoori 02] M. B. Tahoori, *Diagnosis of Open Defects in FPGA Interconnects*, Proc. IEEE Int'l Conf. on Field-Programmable Technology, pp. 328-331, 2002.

[Wagner 87] P. T. Wagner, *Interconnect Testing with Boundary Scan*, Proc. Int'l Test Conf., pp. 52-57, 1987.

[Wang 97] S. J. Wang, et al., *Test and diagnosis of faulty logic blocks in FPGAs*, Proc. ICCAD, pp. 722-727, 1997.

[Xilinx 03] *The Programmable Logic Data Book 2003*, Xilinx Inc., 2003.

[Xilinx EasyPath] *Xilinx easypath solutions*, 2003.

[Yinlei 98] Y. Yinlei, J. Xu, W. K. Huang, F. Lombardi, *A Diagnosis Method for Interconnects in SRAM Based FPGAs*, Proc. Asian Test Symp., pp. 278-282, 1998.