Fault Grading FPGA Interconnect Test Configurations

Mehdi Baradaran Tahoori
Center for Reliable Computing
Stanford University
http://crc.stanford.edu

Subhasish Mitra*
Intel Corporation
Sacramento, CA

Shahin Toutounchi
Xilinx Inc.
San Jose, CA

Edward J. McCluskey
Center for Reliable Computing
Stanford University
http://crc.stanford.edu

Abstract

Conventional fault simulation techniques for FPGAs are very complicated and time consuming. The other alternative, FPGA fault emulation technique, is incomplete, and can be used only after the FPGA chip is manufactured. In this paper, we present efficient algorithms for computing the fault coverage of a given FPGA test configuration. The faults considered are opens and shorts in FPGA interconnects. Compared to conventional methods, our technique is orders of magnitude faster, while is able to report all detectable and undetectable faults.

1. Introduction

A Field Programmable Gate Array (FPGA) is a two-dimensional array of identical Configurable Logic Blocks (CLBs), surrounded by programmable input/output blocks (IOBs). The CLBs are connected through a programmable interconnect network, which consists of switch matrices and wires. Almost 80% of transistors in an FPGA are inside this programmable routing network for programmable switches and buffers. In current FPGAs, more than eight layers of metal are used, most of them for routing resources.

In order to test an FPGA, it has to be configured into test configurations and test vectors, input values at primary inputs, must be applied in each configuration. Sometimes, both together are called *test patterns* in the FPGA industry, and in this paper we follow that convention.

In an industrial setting, test engineers write test patterns, and the fault coverage of these patterns must be computed. Also, if the process of generating the test configuration is automated using an automatic test configuration generation tool, fault coverage must still be quantified as a part of this tool. The objective of this paper is to develop techniques to calculate fault coverage for a given set of test patterns. The presented technique is implemented and is run on actual test patterns developed and used internally at Xilinx Inc.

For traditional testing of integrated circuits, fault simulation techniques are used to quantify the fault coverage of a given set of test vectors. However, for FPGAs, fault simulation is complicated and too time consuming (several hours even for the smallest FPGA at transistor switch-level description). The FPGA model is too detailed and complicated (equivalent to millions of gates for medium to high capacity parts) for fault simulation, and there is no specific commercial tool available for it. And while fault simulation tools for logic networks (ASICs) exist, they are not appropriate for FPGA. The fault emulation technique developed and used in the FPGA industry is not only incomplete, but also very time consuming (several hours even for medium-size parts) [Toutounchi 01]. Further, this method can be used only after the part is manufactured. As will be described later in this paper, many faults, especially most of the shorts, cannot be emulated by present techniques.

Although we implemented this technique on Xilinx Virtex FPGAs, it can be used for other families of Xilinx FPGAs. Also, with some minor adjustments, this work can be generalized for other types of FPGAs.

Section 2 of this paper presents the FPGA model used. Section 3 describes the interconnect fault model. Section 4 presents previous work. Section 5 presents some properties about FPGA interconnect test for opens, which form the basis of our fault grading technique. Section 6 describes testing for short circuit faults. Section 7 presents some enhancements of our technique for including logic blocks. Section 8 presents the results for different actual FPGAs of different sizes. . Finally, Sec. 9 concludes the paper.

* This work was done at the Center for Reliable Computing, Stanford University.

ITC INTERNATIONAL TEST CONFERENCE

0-7803-7542-4/02 $17.00 © 2002 IEEE

2. FPGA Model

2.1. Virtex FPGA Architecture

Virtex, and its variations namely VirtexE and Virtex II, are the leading families of Xilinx FPGAs. They have a capacity of 40K to 10M system gates depending on the size of the part, and offer up to a 420 Mhz internal clock speed and 840 Mb/s I/O [Xilinx 01].

Virtex FPGAs have a regular architecture that comprises a two dimensional array of configurable logic blocks (*CLBs*) surrounded by programmable input output blocks (*IOBs*). The CLBs are all interconnected by a rich programmable network of routing resources. Virtex FPGAs are *SRAM-based*, i.e. the configuration information is stored in internal memory cells (SRAMs). A particular design can thus be implemented in an FPGA by configuring the SRAMs appropriately.

The basic building block of the Virtex CLB is the logic cell (*LC*). An LC includes a 4-input look up table (*LUT*), carry logic, and a storage element. Each CLB contains four LCs, organized in two similar *slices.*

The programmable routing network consists of local and general-purpose routing resources. Adjacent to each CLB is a *switch matrix* through which horizontal and vertical routing resources connect. Also, using these switch matrices and through input and output multiplexers (*IMUXes* and *OMUXes*), CLB gets access to the general purpose routing. Figure 1 shows the abstract view of Virtex FPGA architecture.

2.2. Switch Matrix

The global interconnect network of an FPGA can be viewed as a two-dimensional array of identical programmable switch matrices. These switch matrices are connected to each other by a rich network of wires, which are called *line segments*. Some of these line segments connect adjacent switch matrices (single lines), some others connect switch matrices which are three or six blocks apart (Hex lines) and, others connect switch matrices which are six or twelve blocks apart (Long lines) [Young 99c].

In each switch matrix, line segments connect from north, south, east and west. The switch matrix consists of a number of *Programmable Interconnect Points* (*PIPs*). Each PIP is a pass transistor or series of pass transistors (i.e. path from input to the output of multiplexers implemented at switch-level). These PIPs connect selected pairs of line segments coming to (or going out of) the switch matrix. The gates of these transistors are controlled by SRAM cells. The values of these SRAMs are part of the configuration data which are loaded into the chip when the chip is configured. The PIP acts as a switch. If the switch is closed, the connection between two lines is established; otherwise, these lines are not connected. The line segments connected to switch matrices may have some driver or buffer, and so can be uni-directional or bi-directional, buffered or unbuffered

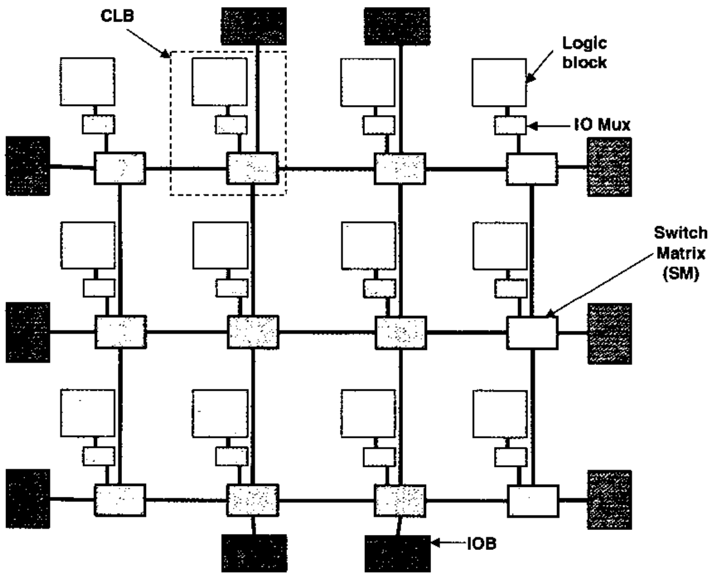

Figure 1 Virtex FPGA architecture. Hex lines and long lines are not shown here.

[Bauer 99][Young 99b]. Therefore these PIPs can be modeled as uni-directional (buffered) or bi-directional switches.

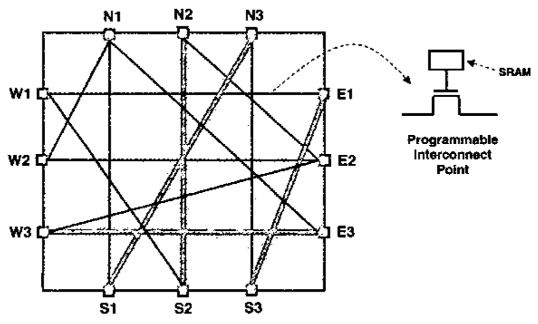

Figure 2 An example of a switch matrix and possible connections inside it

The PIPs inside the switch matrices connect only some selected pairs of line segments; otherwise, it would take a very large amount of area. The interconnection network provides rich connectivity among line segments and also reduces area and delay. This greatly reduces the number of transistors inside the FPGA while preserving the connectivity and routability of a design mapped onto the FPGA with reasonable propagation delay [Tavana 97]. Figure. 2 shows an example of a switch matrix for a given configuration. The PIPs corresponding to bold lines are *closed* - i.e., these connections are "used". The rest are *open* (unused) ones.

3. Interconnect Fault Model

Faults in the interconnect can be categorized in two major groups, namely opens and shorts. An *open* fault can be a PIP stuck-open or an open on a line segment. A PIP *stuck-open* fault causes the PIP to be permanently open regardless of the value of the SRAM cell controlling the PIP. Note that in this work we are not dealing with resistive opens. A *short fault* can be a PIP stuck-closed or a short between two routing resources. Each of these resources can be a PIP or a line segment (namely, PIP-PIP, PIP-line segment and line segment-line segment). Note that a line segment can be shorted to a PIP which is not connected to in the fault-free FPGA. A PIP *stuck-closed* fault causes the PIP to be permanently closed regardless of the value of memory cell controlling the PIP.

The *fault list* for opens consists of opens for all line segments and stuck-opens for all PIPs. Each line segment has one open fault associated with it. Also, each PIP has one stuck-open fault associated with it. The fault list for shorts consists of stuck-closed for all PIPs and shorts for all given pairs of routing resources. Again, each PIP has one stuck-closed fault associated with it. Because all possible pairs are too many (quadratic with the number of all PIPs and line segments in the FPGA), we consider the list of possible shorts (short fault list) as a given input to our algorithms. This fault list can be obtained by a fault inductive technique.

$$\text{Fault coverage of a given set of test patterns} = \frac{\text{Number of faults detected}}{\text{Total number of given faults}}$$

4. Previous Work

4.1. Fault Emulation

A fault emulation technique is used in the FPGA industry to evaluate the coverage of a given set of test patterns. In this method, based on the reconfigurability feature of FPGA, a fault is injected in the chip by configuring the FPGA in such a way that it behaves as a faulty FPGA chip [Toutounchi 01]. The test configuration bit stream is first modified to obtain a faulty configuration, which is then loaded into the chip. Next, test vectors are applied. The test configuration is able to detect that injected fault if the chip (loaded with faulty configuration) produces an incorrect output in response to the applied test vectors.

The stuck-open fault of a PIP can be emulated by configuring that PIP as open (unused) in the test configuration. Similarly, the stuck-closed fault of a PIP is emulated by configuring it as closed (used) for that test configuration. However, there are some significant limitations with this technique.

Fault emulation is an after-silicon task. Therefore, evaluation of test patterns must be postponed until the real chip is manufactured.

Fault emulation is very time consuming (several hours even for medium size FPGAs). For each set of faults which can be emulated in the same configuration, the whole test configuration must be loaded and test vectors must be applied. Although the reconfiguration time can be reduced by ordering the configuration and using partial reconfiguration, this technique still takes a long time even for only stuck-open faults (several minutes for the smallest chip). Also, just generating these configurations takes a long time (almost the same time as the test application time).

Only faults in programmable resources such as PIPs can be emulated. This method is unable to deal with shorts between signal lines which cannot be directly connected by programming the FPGA. This is the major drawback for emulating shorts in the line segments.

4.2. Fault Simulation

An alternative to fault emulation is fault simulation. Fault simulation can be performed at two different levels: namely the switch level and the gate level.

A switch-level fault simulator accepts the detailed transistor-level structure of the FPGA as input. Due to the very large number of transistors in the FPGA (up to millions of transistors) this method takes too much time. For the smallest Xilinx FPGA VirtexE (xcv50e), a commercial switch level fault simulator takes more than 24 hours to complete the fault grading task (for each pattern).

For gate-level fault simulation, the FPGA model is converted to a gate-level netlist. In this netlist all interconnect and routing details are eliminated and only gate-to-gate connectivity information is preserved. In other words, no PIP or line segment usage information is available in this netlist. This method is much faster than switch level simulation because of the great reduction in complexity and detail of the FPGA model. It is comparable to fault emulation in terms of running time. Although this method is faster than switch-level fault simulation, it is not still practical; because there is no interconnect (PIP or line segment) information in this model. Hence, the result of fault simulation cannot be interpreted in terms of the faults in the PIPs and line segments. Moreover, the output report of the tool in terms of stuck-at faults cannot be mapped to opens and shorts in the line segments and PIPs. Even if some bridging fault simulator is used, the results are in terms of bridging faults for nets, not PIPs and line segments.

5. Testing for Opens

In this section, we first describe fault detection inside the switch matrices; then, based on the presented conditions for fault detection, we generalize the idea to detect the faults in the whole FPGA, including opens and shorts in line segments.

5.1. Used-Set and Neighbor-Set

The PIP stuck-open and PIP stuck-closed fault coverage of a test pattern inside the switch matrix can be calculated using the notions of Used set and Neighbor set. These are also the basic notions needed to express the fault coverage for the faults outside the switch matrices in terms of the faults covered inside the switch matrices.

Definition 1. *Used-Set*: The set of all closed PIPs inside each switch matrix form the Used set. Formally:

$$UsedSet = \{(u,v) \mid \text{PIP between } u \text{ and } v \text{ is closed}\}$$

Definition 2. *Neighbor-Set* is the set of all open PIPs that are connected to some closed PIP in the Used-set. Formally:

$$NeighborSet = \{(u,v) \mid (u,v) \notin UsedSet \wedge (\exists x \mid (x,u) \in UsedSet \vee (v,x) \in UsedSet)\}$$

Example. As an example, in the switch matrix of Fig. 2,

$UsedSet = \{(N_2,S_2),(N_3,S_1),(E_1,S_3),(E_3,W_3)\}$,

$NeighborSet = \{(N_1,S_1),(N_2,E_2),(N_3,S_3),(S_2,W_1),(E_2,W_2),(N_1,E_3),(E_1,W_1)\}$

The remaining PIPs are neither in the Used-Set nor in the Neighbor-Set.

The notion of the Used set is used for stuck-open of PIPs, and also open faults of line segments, as we describe later. Similarly, the Neighbor set is used for PIP stuck-closed faults and shorts among PIPs and line segments.

As we explain later through some theorems, in any test configuration, all detectable open faults are a subset of the Used-set. Similarly, all detectable short faults are also a subset of the Neighbor-set.

5.2. Open Detection in Switch Matrices

Figure 3 shows an example of switch matrices and line segments among them. Only closed PIPs in the switch matrices are shown for simplicity. A *point* is the switch matrix terminal (node), for example point A in this figure.

A *path* between two points u and v, is formed by the connection of line segments with closed PIPs between them, from u to v. For example in Fig. 3, there is a path from point *A* to point *L*.

Multiple independent paths between two points are paths that don't share a PIP or a line segment.

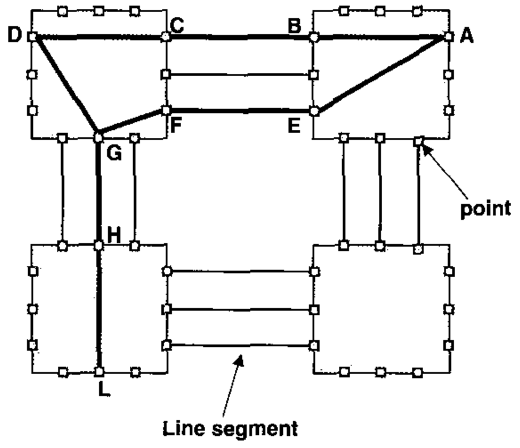

Figure 3 An example of switch matrices and line segments among them.

In Fig.3, there are two independent paths between points *A* and *G*. The first one goes through the points *A, B, C, D,* and *G*. The second one goes through *A, E, F,* and *G*.

Theorem 1. Consider a path between two points, *A* and *B*, such that the logic value on *A* can be controlled, and the logic value on *B* can be observed. All stuck-open faults on closed PIPs on this path are detectable if and only if there are no multiple independent paths between points *A* and *B*.

Proof. As long as there are at least two different paths (i.e. not sharing a PIP) between some controllable point and observable point, say u and v respectively, by removing one PIP, u and v are still connected and the signal can be conducted, so the fault is undetectable. This may change the AC characteristics of the path, but as far as DC testing is concerned, the fault is undetectable.

Otherwise, by removing the PIP from the path, the path is broken and the signal cannot go through and by applying appropriate vectors, the fault can be detected.

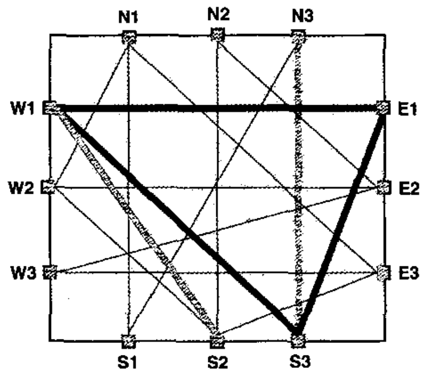

Figure 4 An example of detectable and undetectable stuck-open faults

An example of this situation is shown in Fig. 4. The set $\{(W_1,E_1),(E_1,S_3),(W_1,S_3)\}$ forms a cycle. In other words, there are two independent paths between the points S_3 and W_1. The first one consists only of (S_3, W_1). The second consists of the PIPs (S_3, E_1) and (E_1, W_1). Hence, the stuck-open faults of all of these PIPs are undetectable, but for (W_1,S_2) *and* (N_3,S_3), the stuck-open faults are detectable.

Note that this kind of cycle is possible in the actual test patterns if the goal of the test pattern generation algorithm is only maximizing the PIP utilization.

Figure 3 shows a more general case where more than one switch matrix is involved in forming cycles. In the path from *A* to *L*, only the faults on the line segment from *G* to *H* and the PIP between *H* and *L* are detectable, because there are multiple independent paths between *A* and *G*.

5.3. Modeling of Opens in Line Segments

A net with several fanouts can be divided into its fanout stem and fanout branches. An example of a net is shown in Fig. 5, which has a fanout stem, from *A* to *C*, and two fanout branches, one from *C* to *F* and the other from *C* to *K*. We call each of these stem or branches, a *section* of the net. For example in Fig. 5, there are three sections, namely from *A* to *C*, from *C* to *F*, and from *C* to *K*.

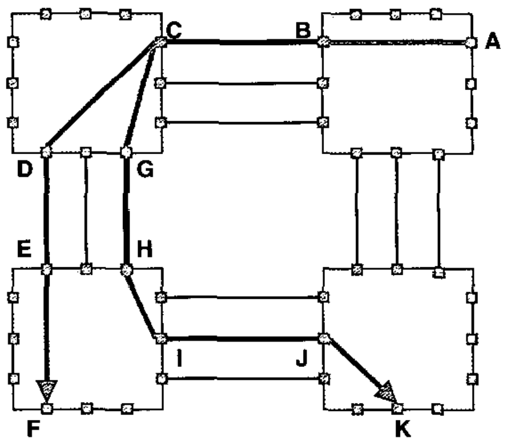

Figure 5 An example of a path with fanouts

Theorem 2. If a line segment lies on a path from a controllable input to an observable output or on a section of a net, then an open on that line segment is equivalent to the stuck-open fault in a PIP on the same section of that path or net.

Proof. When a path is disconnected, the exact location of the fault on the section of that path does not affect detectability of the open fault, provided the fault is located on an irredundant portion of the path, i.e. not on multiple independent segments. On a single conducting path (or at least the section of the path or net that has this property, e.g. the branch from *C* to *F* in Fig. 5), all the open faults on any resource (PIP or line segment) on that path are detectable. Therefore on each section of a path or net, either open faults of all resources are detectable or none of them are detectable. Hence, the open fault of a line segment is equivalent to stuck-open fault of the PIP connecting to that line segment in the same section.

For example in Fig. 5, the open fault on the line segment from *D* to *E*, is equivalent to stuck-open faults of PIPs (*C,D*) and (*E,F*). Hence that open fault is detectable if and only if those stuck-opens are detectable.

5.4. Extension to The Entire FPGA

In this section, we extend the detectability conditions explained in the previous section and generalize it to the entire FPGA.

In previous sections we have only considered PIPs and line segments. Each FPGA test pattern consists of both routing resources and logic blocks. One important point about test patterns for interconnects is that no logic is implemented in most cases. Only *transparent logic* (i.e. identity function $F(x) = x$, which may or may not be followed by a flip-flop) is implemented in the LUTs.

Consider a net whose input is the output of some logic block and its outputs are inputs of other logic blocks. An example of this net is shown in Fig. 6, which extends from output of LUT_1 to the inputs of LUT_2 and has two fanouts X and Y. In this figure, LUT_2 has one other input, Z, which comes from another net.

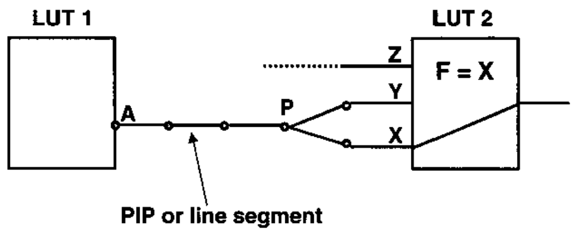

Figure 6 An Example of a net.

When transparent logic is implemented in LUTs, the detectable branch is the one that is passed through the transparent logic, while all the rest are undetectable. For example in Fig. 6, the transparent function $F = X$ is implemented in LUT2. In this case, all the open faults in the branch from P to X are detectable while all the faults on the branch from P to Y or on the net which comes to Z, are undetectable.

Therefore, if only transparent logic is implemented in logic blocks, all detectable and undetectable sections of each net can be found based on the above technique. Hence, there is no need to run any fault simulation and fault coverage can be precisely and quickly calculated only based on the presented conditions.

6. Testing for Shorts

Ideas similar to those presented in the previous section for opens can be applied to detect shorts in FPGA interconnects. All the ideas discussed in the previous section are based on the notion of Used-set. Here, we use the notion of Neighbor set to express the detectability conditions for short faults. We first consider detectability of stuck-closed faults of the PIPs inside the switch matrices, and then generalize the idea to short faults in all routing resources.

6.1. Stuck-Closed PIP Detection in Switch Matrices

Figure 7 shows an example. In this figure, the stuck-closed fault of the PIPs (E_3,W_3), (N_2,S_2), (N_1,S_1),(E_2,W_2) from the Neighbor-Set are detectable. The stuck-closed fault of (N_1,W_2) is undetectable because there is a path of $\{(N_1,E_3),(E_3,S_2),(S_2,W_2)\}$ belonging to the Used-set connecting two end points, N_1 and W_2, of that PIP.

Note that as the unused PIPs are *driven* to a known value (e.g. all are tied down to ground), the stuck-closed fault of (N_1,S_1) is detectable. Even if the line segment connected to S_1 is unused, it is driven to a known value.

Theorem 3. Consider an open PIP (A,B) in the Neighbor-Set such that the logic value on A can be controlled, and the logic value on B can be observed. The stuck-closed fault on this PIP is detectable unless there exists a path connecting A and B (with closed PIPs).

Proof. If a PIP is in the Neighbor-set, it is connected to some closed PIP from at least one end. So, if the PIP is conducting, it connects two already disconnected paths. By applying appropriate test vector, the fault can be detected. The only way that the fault is undetectable is that the PIP connects the previously connected paths. It means that the two ends of the PIP are already connected (conducting). This case is excluded from the condition of this theorem.

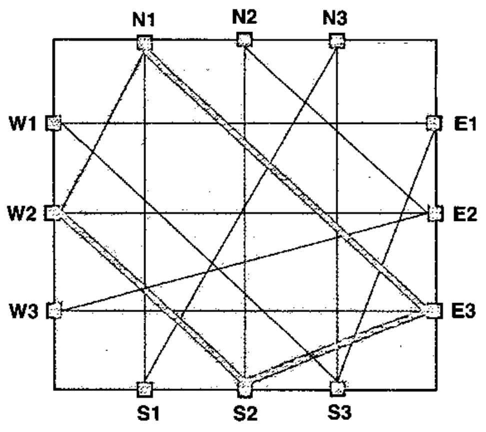

Figure 7 An Example of detectable and undetectable stuck-closed faults

The above theorem is not limited only to one switch matrix. Figure 8 shows another example in a portion of an FPGA where this theorem is still applicable. In this example, where the configuration of a net is shown, the PIP (E_1,W_3) belongs to the Neighbor-Set, but the stuck-closed fault of this PIP is undetectable because E_1 and W_3

are already connected in a net passing through multiple switch matrices.

6.2. Short Detection in The Entire FPGA

To address detectability of shorts in the entire FPGA, we divide the routing resources used in a test configuration to a set of nets. All the resources inside a net are carrying the same signal value.

A pair of routing resources, namely PIP-PIP, line segments-PIP and line segment-line segment, are a *connectable pair* if there is a PIP in the FPGA between those two. Therefore a connectable pair can be directly connected by configuration.

If there is a PIP in the Neighbor-Set between a connectable pair, there is a possibility of detecting the short between those connectable pair.

Two nets are *unrelated nets* if they carry different signal values for at least one test cycle, under a given set of input vectors. A necessary condition for detecting shorts between two elements is that they belong to different unrelated nets. All these conditions are summarized in the following theorem.

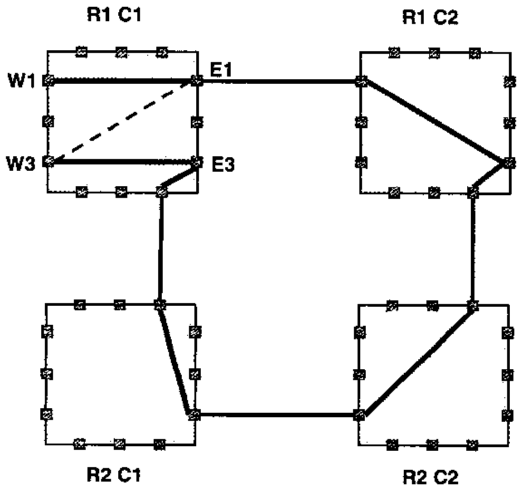

Figure 8 An example of undetectable stuck-closed fault.

Theorem 4. The short fault between two routing resources, PIPs or line segments, is detectable, if they are a connectable pair via a PIP in the Neighbor set whose stuck-closed fault is detectable and they belong to different unrelated nets.

Proof. If the routing resources under test belong to different unrelated nets, according to the definition of unrelated nets, there is at least one test vector under which the value of the nets are different. Therefore the short can be detected.

Otherwise, if they are a connectable pair, and there is a PIP between them whose stuck-closed fault is detectable, it mean that there is not any path between these two elements in this configuration and they are disconnected. Hence, the short between them is detectable.

Figure 9 shows an example detectable short fault between two line segments. In this example, the short between line segments *A* and *B* is detectable because the PIP *(W3,E2)* is in the Neighbor-Set (connected to *(W2,E2)* in the Used-Set), and whose stuck-open fault is detectable.

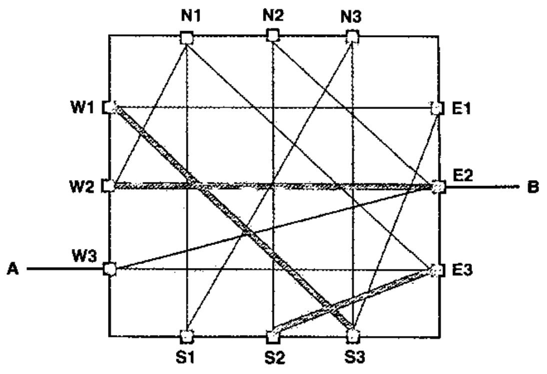

Figure 9 An Example of detectable short

Figure 10 shows another example where the short is not detectable. In this figure, the configuration of two nets is shown. There is a logic block between these two nets implementing transparent logic. Therefore these two nets always carry the same value and are not unrelated. Hence, the short fault between PIPs *(W1,E1)* and *(W3,E3)* is not detectable although they are connectable pair via the PIP *(W3,E1)* which is in the Neighbor-Set.

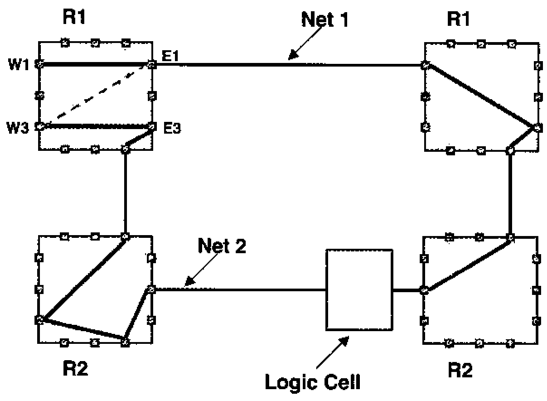

Figure 10 An example of undetectable short

6.2.1. Shorts in Non-connectable Pairs

As described in Sec. 4.1, fault emulation technique can deal with shorts only between connectable pairs.

In our technique, when the routing resources are not directly connectable pair, we can assume that there is a *fake PIP* between them in the software model of the FPGA, a dummy PIP which has no actual realization in the hardware, and apply the presented technique to determine if stuck-closed fault of that fake PIP is detectable. If so, the short between those two resources is detectable, otherwise undetectable. In other words, we convert a non-connectable pair to a connectable one through a fake PIP, and apply the previously described algorithms to determine if the short is detectable.

Using this method, our technique is not limited to only connectable pairs, unlike fault emulation. This is one of major differences of our technique and fault emulation.

7. Generalization for Arbitrary Logic

So far, we have assumed that only transparent logic is implemented by LUTs in the test configuration for interconnects. This is the reason we do not run any fault simulation and we can very efficiently calculate the fault coverage of each test configuration.

However, this method is still applicable if any arbitrary logic, i.e. other than transparent logic, is implemented in the LUTs. When fanout branches of a net go to the same LUT, because of the redundancy, there are undetectable open and shorts faults in the resources of at least one branch. We mark the branches from the same net that reconverge to a LUT as *tentative* detectable branches. Open faults of all resources on those branches are also tentatively detectable. Similarly, the short faults between the resources of these branches are also tentatively detectable.

An example is shown in Fig. 11..In this example, two fanout branches of net go to a LUT implementing *OR* function. Open fault of each PIP or line segment from *P* to *X* or to *Y* is undetectable. Also, the short between each pair is undetectable.

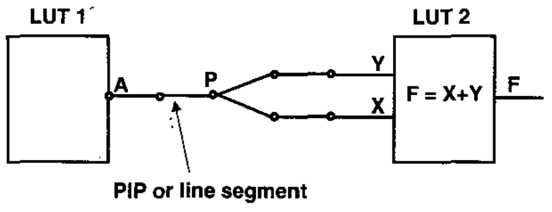

Figure 11 Undetectable faults with logic function

When two or more fanout branches of a net are connected to the inputs of a logic block, we first consider that logic block as a *convergent point*, as shown in Fig. 12., and apply the technique presented in Sec. 5 and Sec. 6. When there are reconvergent fanouts, we mark the resources on those branches as tentatively detectable opens and shorts. To resolve these tentative faults, we exploit a quick fault simulation method which is tractable.

So the complete method works as follows. First, we consider all the logic implemented in the LUTs rather than transparent, as convergent points, as shown in the example of Fig. 12. This way we get similar graphs to those in Sec. 5 and 6. Then, we execute our algorithms on this graph. For each net, the algorithms mark some fanout branches as detectable, some as undetectable, and others as tentatively detectable, as described in Sec. 5.4 and 6.2.

Note that when a net is marked as detectable, it means that the fault can be detected in the configuration by applying appropriate test vector (test vector dependencies), and undetectable means that the fault cannot be detected, regardless of applied test vector. For those nets whose some branches are tentatively detectable, we convert the portion of the configuration containing that net to the equivalent gate-level netlist and run fault simulation for those faults. Figure 13 shows the flowchart for the complete method.

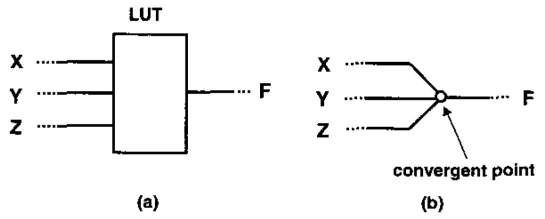

Figure 12 (a) Arbitrary logic in LUT (b) After converting to graph

We can run conventional fault simulators for stuck-at faults and bridging faults and interpret the results in terms of open and short faults as follows. An open fault in a net is undetectable if and only if either stuck-at-1 or stuck-at-0 fault on that net is undetectable. In other words, an open fault is detectable if and only if both stuck-at-1 and stuck-at-0 on that net are detectable. Note that detection of only one stuck-at fault is not sufficient, because the net should be able to transmit both values. A short fault between two nets is detectable if and only if stuck-at-1 on one of them is detectable while 0 is justified on the other one by the same test vector and vice versa. This guarantees that the two nets are set to different values and the effect can be propagated to the observable output(s). We can run a fault simulator for stuck-at faults on the nets with tentatively detectable opens and shorts and interpret the results as mentioned above.

In this reduced gate-level netlist, all the details (PIPs and line segments) inside each section are removed and only connectivity between logic blocks are considered. Therefore, all the resources of a section are reduced to a single wire in the netlist, so are the faults to be simulated by the fault simulator. The logic blocks are modeled as logic gates equivalent to the configuration of those blocks.

As mentioned earlier, this gate-level netlist is abstract so that fault simulation can be quickly run on it. Moreover, only a small portion of test configuration is used as gate-level input to the fault simulator. Also, the fault list contains only open faults for those tentatively undetectable branches. This greatly reduces the amount of work to be done by the fault simulator, and makes it tractable to run this simplified version of fault simulation along with our algorithms.

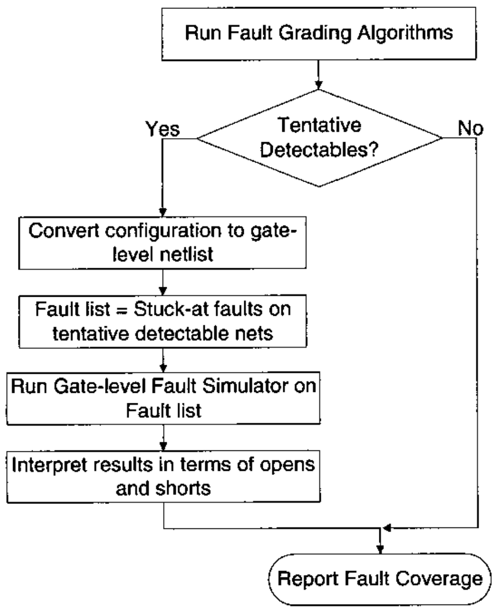
Run Fault Grading Algorithms
Yes
Tentative Detectables?
No
Convert configuration to gate-level netlist
Fault list = Stuck-at faults on tentative detectable nets
Run Gate-level Fault Simulator on Fault list
Interpret results in terms of opens and shorts
Report Fault Coverage

Figure 13 Flowchart for the complete procedure

8. Results

The algorithms are implemented using C++ on UNIX platform. We implemented these algorithms and executed them on the VirtexE family of Xilinx FPGAs. The details of algorithms can be found in [Baradaran 02]. The results are shown in the Table 1. The test configurations implemented in these FPGAs are the patterns developed and used internally at Xilinx Inc. for interconnects. Only transparent logic was implemented in these patterns, therefore there was no need to run fault simulation after executing our algorithms. Hence the row showing the execution time for our technique in Table 1 is just the execution time of our algorithms. As we see in Table 1, our technique is orders of magnitude faster than fault emulation.

In the Xilinx fault emulation technique, it takes about 30 seconds to emulate 1000 open faults. This is possible due to sorting the faults based on FPGA columns and using partial reconfiguration. The time required to emulate all open faults of a given test pattern is equal to the number of closed PIPs in the configuration multiplied by the time spent to emulate a single fault. This is only the test application time. Test preparation, which consists of generating and sorting emulated patterns, takes almost the same time as test application.

As mentioned previously, the switch-level fault simulation takes more than 24 hours for the smallest-size part. The gate-level simulation takes almost the same time as fault emulation, although this method hides all PIP information and reports only stuck-at faults, which is hard to relate to faults in the interconnect. All these show that our method is quite faster than other alternatives.

9. Summary and Conclusion

In this paper, we presented a new technique to calculate the fault coverage of a set of test configurations for FPGA interconnects. In most cases, there is no need to run any kind of fault simulation. Therefore, this method is very fast. It only checks for some detectability conditions in FPGA interconnects which are computationally inexpensive. In this method, the switch matrices are converted to an appropriate graph model and efficient graph algorithms are executed on these models to calculate the fault coverage. This fault coverage is expressed in terms of open faults for line segments, shorts between pairs, stuck-open and stuck-closed for PIPs.

When any arbitrary logic is implemented in the test configuration, the algorithms report the special case of tentatively detectable faults, along with detectable and undetectable faults. The coverage calculation is completed by running a gate-level fault simulator on the gate-level netlist of the test configuration only for those tentative undetectable faults, which form the fault list. Because of these reductions in the amount of work to be done by the fault simulator, the overall method is still quite fast.

Acknowledgement

This work was supported by Xilinx Inc. under Contract No. 2DSA907. The authors would like to thank Dr. Shahin Toutounchi for providing information about Xilinx FPGAs and insightful comments about this work.

Table 1 Execution time for different FPGAs

Part Name	XCV50E	XCV100E	XCV300E	XCV600E	XCV1000E
Number of CLBs	384	600	1536	3456	6144
Number of Logic Gates	~50,000	~100,000	~300,000	~600,000	~1,000,000
Number of PIPs	~620,000	~940,000	~2,280,000	~5,020,000	~8,760,000
Fault Emulation Time	~25 min	~40 min	~100 min	~225 min	~400 min
Our Technique Time	< 1 sec	< 1 sec	2 sec	4 sec	8.5 sec

References

[Baradaran 02] M. Baradaran Tahoori, S. Mitra, E. J. McCluskey, *Techniques and Algorithms for Fault Grading of FPGA Interconnect Test Configuration*, CRC TR, 2002.

[Bauer 99] T. J. Bauer, S. P. Young, *FPGA interconnect structure with high-speed high fanout capability*, US Patent US05907248, 1999.

[Tavana 97] D. Tavana, W. K. Yee, V. A. Holen, *FPGA architecture with repeatable tiles including routing matrices and logic matrices*, US Patent US05682107, 1997.

[Toutounchi 01] S. Toutounchi et al., *Fault Emulation, A Method of FPGA Test*, US Patent pending, April 2001.

[Young 99a] S. P. Young, B. J. New, N. Camilleri, T. J. Bauer, S. Bapat, K. Chandhary, S. Krishnamurthy, *Configurable logic element with fast feedback paths*, US Patent US05963050, 1999.

[Young 99b] S. P. Young, T. J. Bauer, K. Chandhary, S. Krishnamurthy, *FPGA repeatable interconnect structure with bi-directional and unidirectional interconnect lines*, US Patent US05942913, 1999.

[Young 99c] S. P. Young, K. Chandhary, T. J. Bauer, *FPGA repeatable interconnect structure with hierarchical interconnect lines*, US Patent US05914616, 1999.

[Xilinx 01] *The Programmable Logic Data Book 2001*, Xilinx Inc., 2001.