BIST Based Interconnect Fault Location for FPGAs

Nicola Campregher[1], Peter Y.K. Cheung[1], and Milan Vasilko[2]

[1] Department of EEE, Imperial College, London, UK.
[2] School of Design, Engineering and Computing, Bournemouth University, UK.

Abstract. This paper presents a novel approach to interconnect fault location for FPGAs during power-on sequence. The method is based on a concept known as *fault grading* which utilizes defect knowledge during manufacturing test to classify faulty devices into different defect groups. A Built-In Self-Test (BIST) method that can efficiently identify the exact location of the interconnect fault is introduced. This procedure forms the first step of a new interconnect defect tolerant scheme that offers the possibility of using larger and more cost effective devices that contain interconnect defects without compromising on performance or configurability.

1 Introduction

The area occupied by wiring channel and interconnect configuration circuits in an FPGA is significant, occupying 50 to 90 percent of the chip area [1]. With current trends aiming to reduce the area occupied by wire segments in the routing channels, wire width and separation have been reduced. This has in turn led to higher occurrences of wiring defects, such as breaks and shorts, and manufacturing yield decrease [2]. As an alternative to increase once again wire widths and separation, we propose a method to categorize devices exhibiting similar functional defects, in order to provide a solution to tolerate such physical defects and increase manufacturing yield.

Our work aims to take advantage of the deep knowledge manufacturers have of the defects occurrences in their devices [11], while trying not to affect the user's load and device performance.

This paper will introduce a new method to categorize faulty devices, as well as providing test procedures to locate device defects whenever needed. The Built-In-Self-Test (BIST) requires a relatively small amount of configurations to efficiently locate and identify a specific type of defect, determined by the defect grade of the device. Our BIST architecture is easily scalable and can detect multiple functional faults.

This paper will introduce a new approach to defect tolerance. In Section 2 a brief summary of the relevant work carried in this field is given. We go on to provide essential background information in Section 3, and introduce the *fault grading* concept in Section 4. Section 5 will introduce the testing procedure. Section 6 provides some details of the implementation, and finally, in Section 7 we give a brief summary and describe future developments of the work.

2 Previous Work

There are two main strategies to interconnect testing in FPGAs. One is the application of BIST approach to interconnects [3,7,8,9]. The BIST technique proposed by *Stroud et al.* [3] is a comparison based approach. Two WUTs (Wires Under Test) receive identical test patterns and their outputs are compared by ORAs (Output Response Analyzers). This approach however fails to detect multiple faults that have identical faulty behavior in the two WUTs groups. This BIST technique is also aimed at dynamic faults, and can be used during device operation. It provides complete fault coverage, however it requires an external reconfiguration controller.

A similar concept has been proposed by *Niamat et al* [7]. Two sets of 4 wires are applied with test vectors and their output compared. The ORA in this case does not produce a pass/fail indication but a bit sequence. The output response is then stored in a LUT and used at later stages to locate the position of the fault.

A different implementation based on parity check was proposed by *Sun et al* [10]. In this approach the outputs of the WUTs, connected in snake-like chains, are checked for parity against the parity from the original test vectors at the ORA to produce a pass/fail result. This approach however, due to the way parity checking is done, has only 50% error detection efficiency and is very time consuming. The authors in [8] presented a BIST scheme for cluster-based FPGA architectures. Based on the concept of test transparency, they define configurations which enable test access to high density logic clusters embedded within each FPGA tile.

In general, the BIST approaches reported so far require many configurations and are unsuitable if fast testing is needed. The other strategy is a more classical approach not using BIST. Various researchers have proposed different models [4,5,6], based on different assumptions. All these methods, albeit very fast and compact, are limited in functionality by the number of I/O pins needed to access the chip at various stages of the testing process. They are thus unsuitable to applications requiring resource inexpensive testing.

3 Background

3.1 SRAM FPGA Architecture and Physical Layout

The target FPGA architecture is an island-style FPGA. As stated in previous sections, our work is targeting problems that may arise as devices grow in size. We are therefore targeting the latest top-end devices [12,13]. These have clear architectural characteristics, such as hierarchical routing and multiple LUTs in their configurable logic blocks. Because of their advanced development, the switch matrices inside these elements are very complex. Unfortunately, not much is known about the their structure and connectivity. We will therefore assume a simple switch matrix block, where any incoming wire can connect to only one other wire in all 3 directions. This simplified model, while sufficient to demonstrate basic principles of our method, can easily be extended to cope with complex switch matrices.

Furthermore, we make the assumption that the defect occurs into only one type of interconnect resource and that all others are functionally faultless. This in turns leads to the assumption that all wires of the same segment type lie on the same physical layer. Furthermore, wires of the same type in the horizontal and vertical channels do not lie in adjacent layers, thus eliminating the issue of cross-connections between them. All these assumptions can easily be relaxed without affecting the basic method described.

3.2 Fault Models

Our structural fault model only targets FPGA's interconnect resources. The interconnect structures of a typical FPGA include wires and programmable switch matrices.

Typical faults affecting wirings are short and open faults. Switch Matrices faults are limited to switches being *stuck-off* or *stuck-on*. *Stuck-off* faults are treated in the same way as wire opens. *Stuck-on* faults, however, are more difficult to detect and diagnose. *Stuck-on* faults mean that two wires are permanently connected via the switch matrix. In the presence of stuck-on faults the signal is propagated through an unwanted resource and hence the fault location procedure has to account for all possibilities. This means that all possible combinations of switch matrix configuration have to explored. Figure 1 shows all the possible fault occurrences.

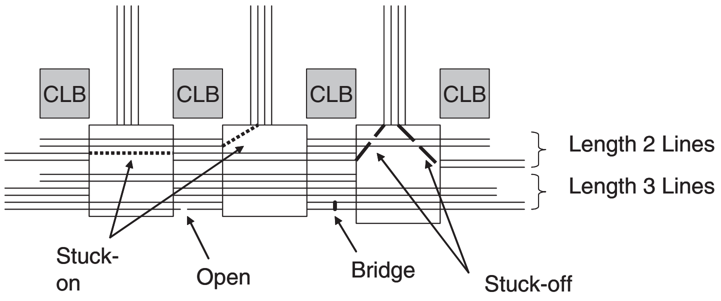

Fig. 1. Fault Models

4 Fault Grading

Devices undergo a multitude of manufacturing tests at various stages of the manufacturing process. Some degree of test is performed at the wafer level, where defective devices are marked and discarded after cutting. Parametric tests are performed on packaged devices; failing devices are once again discarded, whereas devices that pass parametric tests are "binned" into different speed categories depending on how they performed during tests. Failed devices are mainly discarded even though the total amount of resources affected by a physical defect is minimal. Some of these devices can be used, albeit not in full capacity. Manufacturers have already looked at ways to reuse faulty devices. One such solution is offered by Xilinx with their Easypath devices[14]. Easypath devices are tested only for the resources used for a specified design. This means that devices do not have to pass all manufacturing tests, but only a limited number of them. Customers are then offered with devices mapped exclusively for their design, but at a reduced cost. They however lose the possibility to reconfigure the devices for future upgrades.

Instead of using the Easypath approach, we propose that devices can be categorized with respect to the functional fault they exhibit. Functional faults are independent of physical faults, such as open vias or masking defects found during manufacturing tests [11]. A specific functional fault only affects part of the device, and if it can be avoided, the rest of the chip can be made to work with only slightly altered characteristics. Our fault grading scheme aims to provide fault categories for defective devices that have failed similar functional tests.

The concept of fault grading is very similar to that of speed grading: devices will always exhibit different characteristics, and are therefore categorized according to specific parameters. Devices are marked and designs compiled according to those specific parameters. It is therefore possible to generate new categories, and using this information defective devices can be used to implement the majority of designs.

The fault grades contain information about the fault the device exhibits. The amount of information the fault grades contain is a trade-off between what is needed in order to avoid the fault and generalization of the defect. One extreme is a fault grade that contains the exact location and type of fault. The other extreme is a simple tag identifying a faulty device. A good compromise is a fault grade that indicates what type of resource is affected. This leads to a limited number of fault grades, that contain enough information to generate test configurations to locate the fault in the device during power-on.

As an example, consider a Xilinx Virtex II PRO device and its general routing matrix [12]. This device offers 4 different types of lines: direct connections, double lines, hex lines, long lines. Four fault grades could be used to categorize fault on the wire resources. Two grades could be used for switch matrices faults, one to identify stuck-on faults and one for stuck-off faults. These grades are chosen with a defect tolerance scheme in mind, and how to avoid certain defects with the lowest overhead possible. Assuming that all other resources are unaffected, we can efficiently test all the interconnects of the same type that could possibly be faulty, during the power-on sequence, in order to provide an alternative design to avoid the faulty resource.

5 Testing Strategy

We propose a BIST methodology to detect and diagnose a functional fault on a known interconnect resource type. Our strategy consists of a point to point analysis, where a wire is forced to a logical value at one end and observed at the other. If the observed output is not equal to the input, a fault is present. A BIST environment consists of the *Test Vector Generator*(TVG), the *Wires Under Test*(WUT), and the *Output Response Analyzer*(ORA). TVGs select the pattern of 0's and 1's to be applied on the WUTs, while the ORAs compare the the WUTs response against a predefined sequence and issue a pass/fail signal accordingly.

Considering the nature of modern FPGAs, where routing channels are considerably large, it is feasible to group the Wires Under Test together and perform an analysis at the ORA of all grouped wires.

TVGs and ORAs can be implemented using CLBs. As most modern devices are made of large CLBs (comprising of multiple LUTs) we can implement a TVG *and* a ORA using a single CLB. The TVG/ORA combinations are arranged in a chain that spans the entire width or height of the device.When a fault is detected, a 'fail' signal is passed on through to the chain end. The propagation within the chain is synchronized with a clock, so that an ORA in the N^{th} position in the chain will only be allowed to access the chain at the N^{th} clock cycle. When a 'fail' signal is detected at the end of chain, the position of the chain in the array is found by the BIST controller using simple decoding logic. A diagram of such a system is shown in Figure 2.

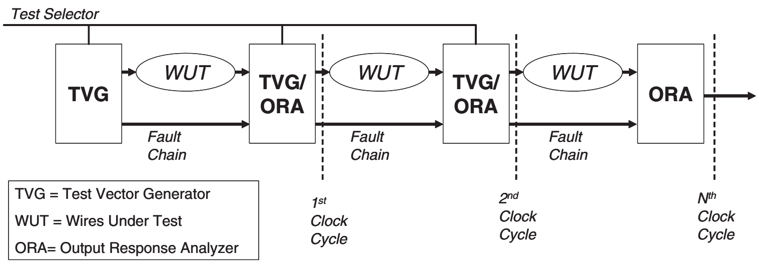

Fig. 2. Testing Strategy

5.1 WUTs Grouping

Taking into account that 4-input LUTs are the main building block in most modern FPGA, the simplest ORA implementation is by using one such element. The TVGs are implemented using multiple LUTs, one for each wire in the set of WUTs. This allows complete independence of test vectors between wires in a set of WUTs. As a compromise between TVGs and ORAs implementations, it was decided to group the WUT in groups of 4. This arrangement would require a single 4-input LUT for the ORA, whereas 4 4-input LUTs would be required for the TVGs. Such quantities are not uncommon in readily available devices [12]. The implementation can be altered to best fit any device with different architectural characteristics. The resulting arrangement is shown in Figure 3. The dotted lines in Figure 3 represent wires from the adjacent set of wires, which have to be driven with opposite signals as the adjacent WUT to account for bridging faults across assigned sets. Those wires are not considered at the ORA but might nonetheless have an effect on the WUT in case of bridging faults.

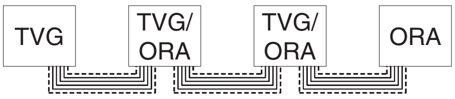

Fig. 3. Grouped WUTs between ORAs and TVGs

5.2 TVG and ORA Operation

TVGs generate bit sequences to account for all possible faults that could develop in the interconnect resource. They are implemented as simple look-up tables, where the output is selected from the Test Selector input. The Test Selector input is generated from the BIST controller, and is a global signal routed to all TVGs. For wiring faults, four basic test vectors can detect any defective occurrence. These, defined as the four *basic* vectors, are:

- 0000 Tests for stuck at 1 faults.
- 1111 Tests for stuck at 0 faults.
- 1010 No two adjacent wires are applied the same value. Tests for bridging faults.
- 0101 Alternating 1's and 0's, in reverse order from the previous test vector. Tests for bridging faults.

The *basic* test vectors can identify the set of WUTs that contains a fault. To correctly identify the exact faulty wire within a given set of WUTs, extra test vectors can be used. This second set of vectors is dependent upon the result of four *basic* test vectors and is decided by the BIST controller. The function of the second set of vectors is purely to improve the fault resolution.

The ORA function is to generate a pass/fail signal according to some predefined parameters. The ORA is designed to fit in a single 4-input LUT and under our scheme, it will issue a 'pass' signal only if the four WUT have logical values corresponding to the 4 basic test vectors. Under all other circumstances it will issue a 'fail' signal.

5.3 BIST Controller Operation

The BIST controller operation during test is shown in Figure 4. While the test vectors are being propagated through the chains, the BIST controller checks the end of all chains for any 'fail' signal being issued. If such a signal is found, the current counter value (representing how far along the chain the vectors have been propagated) and the chain end identifier represent the coordinate of the ORA that has detected the fault. The output from the BIST controller is a string of four bits regarding which of the four basic test vectors has found a fault. If, for instance, the string of results from the BIST controller is 1000, test vector 1 has caused a fault. This means that the fault present in the system is a *stuck-at-1* fault, as test vector 1 could not have caused or detected any other unexpected behavior.

```
var ChainEnds: array of binary(0 to N-1) :=(all=0);      //Chain Ends
var result: array of binary (0 to 3) := '0000'           //Test results
var counter, x_coord, y_coord: integer

begin
 for ( j in 0 to 3)
      case j is
           when (0) - apply 0000                          //Test vectors
           when (1) - apply 1111
           when (2) - apply 1010
           when (3) - apply 0101
      end case
      counter := 0
      for (x in 0 to M-1)
          for (i in 0 to N-1)
               if ChainsEnds(i) = 1 then                  //Fault found
                    result(j) = 1                         //Fault recorded
                    x_coord := counter
                    y_coord := i
                end if
          end for
          counter:=counter + 1
      end for
 end for
end
```

Fig. 4. BIST operation during test

From the inspection of the test results of the *basic* test vectors the BIST controller can determine what type of fault is present in the system and apply other test vectors to identify exactly which wire in the group of WUTs is faulty. Note than any fault or combination of faults confined within the set of WUT would cause at least two tests to fail. The only possible fault not confined within the set of WUT is a bridge onto adjacent set of wires. This causes only one of the bridging test vectors to fail. From the combination of failed tests the BIST controller can reduce the fault resolution to 2 or 3 wires or pairs of wires in the set of WUTs, as shown in Table 1. The second set of test vectors is designed purely to increase the fault resolution by selection of any one of the already selected wires. During propagation of the extra test vectors, the pass/fail signal from the ORAs are used as selection between wires to identify the faulty one.

If, for example, the combined test results are 1010, the fault is limited between Wire 2 or Wire 4 being stuck at 1. The next test vector, 1110, is then propagated.

Table 1. BIST selection

Test Vectors (Wire 1 - Wire 4)					
(1) - 0000	*(2) - 1111*	*(3) - 1010*	*(4) - 0101*	*Fault*	*Next Vector*
0	0	0	0	No Fault	*N/A*
1	0	1	0	Wire 2 or Wire 4 s-a-1	1110
1	0	0	1	Wire 1 or Wire 3 s-a-1	0111
0	1	1	0	Wire 1 or Wire 3 s-a-0	1000
0	1	0	1	Wire 2 or Wire 4 s-a-0	0001
0	0	1	1	Bridge	All previous 4
0	0	0	1	Bridge onto next set	All previous 4
0	0	1	0	Bridge onto next set	All previous 4
All others				Multiple faults	Composite

As by this point we have eliminated the possibility of any other fault, the ORA inputs can only be 1110, if Wire 2 is s-a-1, or 1111, if Wire 4 is s-a-1. The first option will result in a 'fail' signal from the ORA, whereas the second option will result in a 'pass' signal. From the ORA response we can therefore increase our fault resolution to identify precisely the faulty wire.

6 Implementation

We are proposing a BIST strategy to be used with prior knowledge of the faulty resource. Our BIST strategy is a point to point one, where test vectors are applied at one end to a set of WUTs and observed at the other. TVGs and ORAs are arranged in rows, so that pass/fail results are propagated and read from only one location for each row. The BIST controller decodes the outputs from the end of the ORA chains to provide fault location. The WUTs are grouped in sets in order to offer the highest degree of parallelism considering the architectural and strategic constraints. The total number of configurations needed to complete testing is dependent upon the total number of wires of the same type present in the device. The configurations are grouped into phases, where configurations belonging to the same phase aim to test different interconnects appertaining to the same channel.

6.1 Number of Configurations

To fully test a routing channel all lines originating or terminating from a single CLB have to be tested. If the architecture has L lines of a specific type originate from any CLB in a channel, then the test of all lines in a channel will need $\lceil L/4 \rceil$ number of configurations. Modern FPGAs rarely have more than 20 lines of any type generating from any CLB in one channel [12], hence 5 configurations are sufficient to test all lines in channel. These make up a test phase.

6.2 Wire Testing Phases

Considering an $M \times N$ array , with N CLBs in each horizontal channel and N CLBs in each vertical channel, $N+1$ vertical routing channels and $M+1$ horizontal routing channels exist [1]. Testing of each horizontal or vertical routing channel requires all the CLBs in a row or column, respectively. In the vertical and horizontal direction, testing of all channels requires at least 2 phases, where during the first N or M channels respectively are tested, and in the second the channel left over is tested. The second phase of the vertical and horizontal channels testing can be combined, as shown graphically in Figure 5. Three phases are required to test all the lines of the same type in all channels. If $\lceil L/4 \rceil$ are required for each phase, a total of $3 \times \lceil L/4 \rceil$ are needed for testing the whole device.

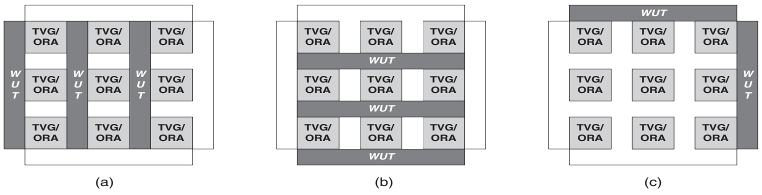

Fig. 5. Three configuration phases

6.3 Switch Matrix Testing Phases

To test for switch matrix faults the WUTs signals are routed trough switch matrices. *Stuck-off* faults are dealt dealt with just like open faults. In the event of *stuck-on* faults, bridges are created within the switch matrix configurations for detection.

The switch matrix configurations needed to test for all *stuck-on* and *stuck-off* faults ar shown in Figure 6. The diagram shows the routing inside the switch matrix in order to cause bridging faults under all possible matrix combinations. At the same time, the routing shown also explores all the possible connections within the matrix itself. The testing scheme remains unchanged: if an ORA

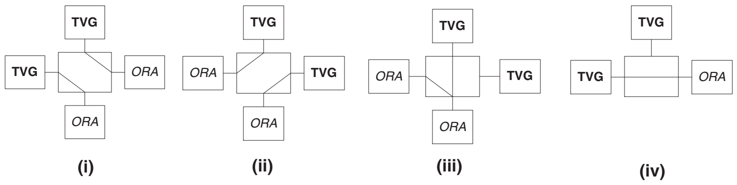

Fig. 6. Switch Matrix fault diagnosis phases

detects an unexpected behavior, a 'fail' signal is propagated through the end of the chain. The only tweak to the original scheme is that two TVGs connected to the same switch matrix produce opposite signals. In the case of *stuck-on* switches, this causes a behavior identical to that of a bridging fault.

The diagnosis of *stuck-off* faults is straight forward, as 'fail' signal from any ORA can only be caused by one faulty connection in all routing configurations. For *stuck-on* faults, however, a different analysis has to be performed. A permanent connection between two terminal will cause all the ORAs connected to the faulty switch matrix to detect a fault. But any permanent connection will only be detected during a specific number of routing configurations. From the analysis of the result of the 4 test phases, the BIST controller can determine the exact faulty connection. The faults and failures caused are summarized in Table 2.

Table 2. Stuck-on faults resolution

Faulty Connection	***Routing configuration detected by***
North-East	ii,iii,iv
North-West	i,iii,iv
South-East	i,iii
South-West	ii
North-South	i,ii
East-West	i,ii,iii

6.4 Case Study: Xilinx Virtex II Pro

This FPGA device allows TVG and ORAs to be implemented in a single CLB, thanks to the high number of 4-input LUTs present. For simplicity purposes we consider the case where a double line in the general routing matrix is faulty for a XC2V20 device. The Virtex II Pro has 20 double lines originating from a CLB in both vertical and horizontal channels. This leads to a total of 5 configurations per test phase. Therefore a complete test would require 15 configurations. to fully test all double lines available in the FPGA. Assuming a worst-case scenario of JTAG download, each configurations requires $249ms$, so the total time required for reconfigurations is $3.74s$. This time can be considerably reduced if a SelectMap interface is used for download. In this case, total download time would be just over $0.3s$. The actual test time, in terms of clock cycles is in both cases much smaller than the configuration download time and thus it would not affect total testing time by a great amount.

7 Conclusions and Future Work

We have presented a new framework for FPGA testing for defect tolerance. The concept of device fault grading has been introduced, together with simple, effective testing procedures. Under our scheme it is possible to load testing configurations to the FPGA with the specific aim of locating a fault whose nature is already known. The testing is done completely on-chip.

This work provides manufacturers and users with a different approach to defect tolerance. The development of this framework is based around the assumption that defective devices will show similar functional faults spread around the chip area. It is possible to categorize these defects with respect to their functional faults. In the design process we can account for the fault to be found anywhere around the chip and limit the usage of a faulty resource to a minimum. The exact location of the fault can be found by loading the proposed test configurations during the power-on sequence

The next step in our work will be to integrate the fault grading and fault diagnostic into a complete defect tolerance framework, offering an alternative design to the most common defect tolerance problems.

References

1. S.D.Brown, R.J.Francis, J.Rose, and Z.G.Vranesic, *Field Programmable Gate Arrays.* Norwell, MA:Kluwer, 1992.
2. F. Hanchek, and S. Dutt, "Methodologies for tolerating cell and interconnect faults in FPGAs," *Computers, IEEE Transactions on* , Vol. 47(1), pp. 15-33, Jan. 1998.
3. C. Stroud, S.Wijesuriya, C.Hamilton, and M.Abramovici, "Built in self test of FPGA interconnect," *Proc.Int. Test Conf.*, pp. 404-411, 1998.
4. M.Renovell, J.Figueras, and Y.Zorian, "Test of RAM-based FPGA: Methodology and application to the interconnect structure," in *Proc. 15th IEEE Very Large Scale Integration (VLSI) Test Symp.*, 1997, pp. 204-209.
5. M.Renovell, J.M.Portal, J.Figueras, and Y.Zorian, "Testing the interconnect of RAM-based FPGAs," *IEEE Des. Test Comput.*, 1998, pp. 45-50.
6. H.Michinishi, T.Yokohira, T.Okamoto, T.Inoue, and H.Fujiwara, "Test methodology for interconnect structures of LUT-based FPGAs," *Proc. 5th Asian Test Symp.*, pp. 68-74, 1996.
7. M.Y.Niamat, R.Nambiar, and M.M. Jamall, "A BIST Scheme for testing the interconnect of SRAM based FPGAs," *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*, Vol. 2 pp. 41-44, 2002.
8. I.G. Harris and R. Tessier, "Testing and diagnosis of interconnect faults in cluster-based FPGA architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.21, pp.1337-1343, 2002.
9. J.Liu and S. Simmons, "BIST diagnosis of interconnects fault locations in FPGA's," *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, Vol. 1, pp.207-210, May 4-7, 2003.
10. X. Sun, S.Xu, J.Xum and P.Trouborst, "Design and implementation of a parity-based BIST scheme for FPGA global interconnects", *CCECE*, 2001.
11. Xilinx Inc., "The Reliability Report," Sep. 2003.
12. Xilinx Inc., "Virtex II Pro Platform FPGA Handbook," Oct. 2002.
13. Alter Corp., "Stratix II Device Handbook," Feb.2004.
14. Xilinx Inc., "Virtex II Pro EasyPath Solutions," 2003.