CONFIGURATION SELF-TEST IN FPGA-BASED RECONFIGURABLE SYSTEMS

Wasim Quddus, Abhijit Jas, and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
{quddus, jas, touba}@cat.ece.utexas.edu

ABSTRACT

An FPGA-based reconfigurable system may contain boards of FPGAs which are reconfigured for different applications and must work correctly. This paper presents a novel approach for rapidly testing the configuration in the FPGAs each time the system is reconfigured. A low-cost configuration-dependent test method is used to detect faults in the circuit. The "original configuration" is modified by only changing the logic function of the CLBs to form "test configurations" that can be used to quickly test the circuit. The test procedure is rapid enough to be performed on the fly whenever the system is reconfigured. The technique is independent of any fault model since it partitions the circuit into segments and tests each segment exhaustively.

1. INTRODUCTION

A field-programmable gate array (FPGA) can be configured in the field to implement a desired logic function. A static RAM based FPGA architecture has a matrix of configurable logic blocks (CLBs), programmable interconnect, and programmable I/O cells. A user can specify a logic function and then use compiler software to map the logic function to a network of CLBs which are then placed and routed. The end result is a particular configuration for the FPGA which implements the logic function.

One powerful and exciting application of FPGAs is in constructing *reconfigurable systems* (or custom computing machines). The hardware in such systems can be reconfigured to adapt to different computing requirements for different applications. Reconfigurable systems offer higher computational density and higher throughput for many applications compared with conventional fixed hardware systems.

An important and challenging issue for reconfigurable systems is reliability. There are two reasons why it is important to check each time the hardware is reconfigured that the new configuration is fault-free:

1. A defect in the reconfigurable hardware may only cause faulty behavior for certain configurations. It is possible that a defective FPGA component may pass manufacturing test and may work correctly for many different configurations, but then may suddenly fail for one particular configuration.
2. There are many failure mechanisms that can cause an FPGA component to fail over time. Periodic testing of the reconfigurable computing machine allows failed components to be identified and replaced.

In order to describe the relationship between the work presented here and previous work in FPGA testing, it is important to make the distinction between *configuration-independent testing* and *configuration-dependent testing*. In configuration-independent testing, no assumptions are made about the way in which the FPGA will be configured by the user. The goal is to maximize the fault coverage for all possible configurations. Configuration-independent testing is done when the FPGA is manufactured (i.e., before it is shipped to the user). Previous work in FPGA testing has focused on configuration-independent testing. Configuration-dependent testing, on the other hand, involves testing that a particular FPGA configuration is fault-free. A higher fault coverage (for the particular configuration) can be achieved with less test time. The approach described here is a configuration-dependent test technique. Configuration-independent techniques for testing FPGAs have been described in [7][10], and techniques for diagnosis are described in [2][3][4].

The approach of detecting, locating, and avoiding faulty hardware in reconfigurable systedms is used in the Teramac Custom Computer [1]. A configuration-independent test approach is used where each resource in the system is tested several times with different test configurations. When faults are detected, the faulty hardware is marked so that the compiler will avoid mapping to those resources. Three drawbacks of the diagnostic tests are described in [1]:

1. Time to develop (through experimentation) test configurations that provide a high coverage.
2. Time it takes to develop the diagnostic tests necessary to locate (not just detect) the faults.
3. Time to run the tests.

The configuration-dependent test method presented in this paper addresses the problem of detecting faults in a given configuration for reconfigurable systems. By adopting a functional testing technique this method does not depend on any fault model. However it should be noted that for the configuration under test, this method will provide 100% single stuck-at and multiple stuck-at fault coverage. Moreover this technique is independent of the underlying FPGA architecture and hence can be used in future generations of FPGA chips with no modifications. It provides a systematic procedure for testing a particular configuration. This is done by modifying the "original configuration" by only changing the logic function of the CLBs to form "test configurations" that can be used to quickly test the circuit. A systematic procedure is applied to the "original configuration" to generate a small set of test configurations. The logic function of the CLBs in the test configurations are chosen in such a way that the "segment verification testing" [5] approach can be used to detect all stuck-at faults in the circuit with a small set of test sessions. Since only the logic functions of the CLBs are changed, time consuming placement and routing is avoided. The process of generating the test configurations and test vectors is fully

0-7803-5471-0/99/$10.00©1999 IEEE

automated and very fast. It can be performed whenever the system is reconfigured.

Stroud, et al., [8][9] proposed a configuration-independent pseudo-exhaustive testing approach for the CLBs in an FPGA. However, since the area of an FPGA is dominated by the programmable interconnect, many faults occur in the interconnect. Thus, thorough testing of the interconnect for each configuration used in a reconfigurable system is also very important. The method presented here is a configuration-dependent approach that detects faults in both the interconnects and the CLBs used for that configuration.

2. OVERVIEW OF TEST METHODOLOGY

In order to quickly test a new configuration, a functional test that is independent of any specific structural fault model is needed (to avoid time consuming test generation). The most obvious approach is *exhaustive testing*, where all possible input combinations are applied to a circuit. However, with this approach the test length grows exponentially with the number of circuit inputs and hence this method can not be used in practice to test circuits with a large number of inputs. An alternative testing strategy is *pseudo-exhaustive testing* (*PET*), which involves testing portions of the circuit exhaustively instead of the whole circuit. *PET* takes on different forms depending on how exactly these portions (sub-circuits) are derived and tested. One such technique of accomplishing *PET* is *verification testing* [5]. Our proposed method uses this concept. But in our technique, we can avoid additional hardware as well as expensive simulations for path sensitization by using the "reprogrammable" capability of the FPGAs. In our method, paths are "sensitized" by programming CLBs in "pass-through" mode. A CLB in "pass-through" mode has its output function equal to one of its inputs. Note that a CLB can be put in "pass-through" mode without changing the interconnect routing.

Given a new configuration for a reconfigurable computing machine, the goal here is to test the reconfigurable hardware for that specific configuration to ensure that it is fault-free. In this paper, it will be assumed that the reconfigurable hardware consists of multiple static RAM based FPGA chips with programmable interconnect between them. However, the ideas and concepts described in this paper can be applied to other types of reconfigurable hardware.

Given a logic function to be implemented by reconfigurable hardware, compilers are used to translate the logic function into an interconnection of CLBs where each CLB implements some logic function. This is then mapped to physical CLBs and interconnect to form a configuration. The goal here is to test the physical CLBs and interconnects to make sure that they are fault-free (i.e., correctly implement the desired logic function).

The strategy for testing a particular configuration is to divide it up into single output segments such that the number of inputs to each segment is less than a user specified parameter N, where N is small enough to enable exhaustive testing of the segment. The segments have to be formed so that any fault in the overall configuration will be present in at least one of the configuration segments. So the strategy for detecting faults is to thoroughly test each configuration segment. Each configuration segment consists of only a subset of the total number of CLBs and interconnects in the reconfigurable hardware. Multiple segments are then tested simultaneously. The number of segments that can be tested simultaneously is limited by the number of primary outputs of the circuit. In order to test each segment we need to observe its output. This can be done by routing the segment-under-test's output to one of the primary outputs. In conventional hardware, observation points need to be inserted to do this if the segment output can not be sensitized to a primary output. But in reconfigurable hardware this can be done by programming some of the CLBs in "pass-through" mode and routing the segment output through such CLBs to one of the primary outputs of the circuit. This is one of the key advantages of this method. Also we cannot route more than one segment output to the same primary output because we need to observe the segment outputs independently to prevent aliasing. Thus the number of primary outputs is an upper bound on the number of segments that can be tested simultaneously. Similarly some segment inputs may not be primary inputs of the circuit. In this case also we can route some primary input to the segment input by selectively programming some of the CLBs in "pass-through" mode.

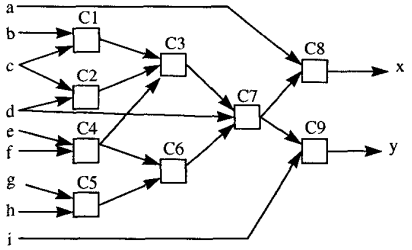

Fig. 1. A Circuit Composed of CLBs and Interconnects

The proposed technique is best explained with an example. Consider the circuit of Fig. 1. It has 9 primary inputs $a,b \ldots i$; 2 primary outputs x and y; and 9 CLBs $C_1, C_2, \ldots, C_9$. The first step in the algorithm is to compute the *dependency set* for every CLB in the circuit. The dependency set for a CLB is defined as the set of primary inputs that have a path to the CLB. For example, the dependency set for C_3 is $\{b,c,d,e,f\}$ and that for C_6 is $\{e,f,g,h\}$. If the cardinality of the dependency set of the CLB is less than or equal to N (the user specified threshold), then the CLB and its driving cone of logic forms a "*testable segment*". For example, if N is 5 in our example, then the initial testable segments are $S_1 = \{C_1, C_2, C_3, C_4\}$ and $S_2 = \{C_4, C_5, C_6\}$. In the example circuit, two test sessions will be needed to test the two segments as we can not route both the segment outputs to primary outputs along independent paths. This is shown in Figs. 2 and 3.

The bold lines indicate the routed path and the dotted CLBs are the ones that are programmed in "pass-through" mode. The bold CLBs define the segments that are being tested. After the segments are tested, the CLBs are marked as tested. Some of the tested CLBs can then be programmed in "pass-through" mode to reduce the input dependency of the other untested CLBs to form additional testable segments. For example, if we program the CLBs C_1, C_3, C_5, and C_6 in "pass-through" mode, we can test the remaining segment consisting of the CLBs C_7,

C_8, and C_9. This is illustrated in Fig. 4. Thus the example circuit can be tested in 3 test sessions.

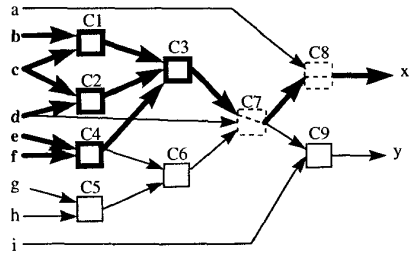

Fig. 2. Testing Segment $S_1 = \{C_1, C_2, C_3, C_4\}$

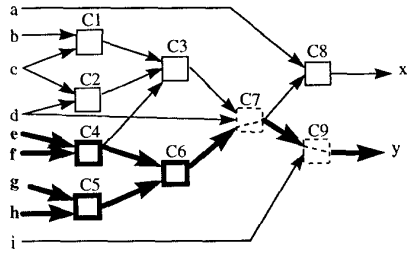

Fig. 3. Testing Segment $S_2 = \{C_4, C_5, C_6\}$

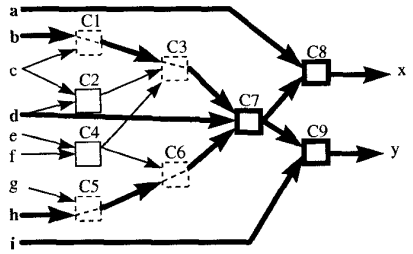

Fig. 4. Testing Segment $S_3 = \{C_7, C_8, C_9\}$

The test configurations are determined by the procedure described in the following section. When the system is reconfigured, each of these "test configurations" are loaded one by one and the pseudo-exhaustive patterns are applied to each configuration. If no faults are detected, then the "system configuration" is loaded and the system continues operating. The pseudo-exhaustive patterns can either by applied by external hardware, or they can be generated using a pseudo-exhaustive pattern generator on-chip (e.g., having an LFSR feed the boundary scan around the chip) to provide a self-test functionality.

3. SEGMENTATION PROCEDURE

In order to make the whole testing procedure fast, it is very important to minimize the number of test sessions. In the example, it was shown that the entire circuit can be tested in three test sessions. This process of forming the test segments needs to be automated. Moreover we need to minimize the number of test sessions. We have developed an efficient algorithm for solving this test session minimization problem. The key components of our algorithm are described below.

The first step in our algorithm is identification of a "good" segment. It is desirable that the segment should include as many CLBs as possible without exceeding the limit on its *fan-in*. The fan-in of a segment is defined as the number of inputs to the segment. Note that some of these inputs may not be primary inputs of the circuit. The *circuit partitioner* tries to identify a "good" segment for testing. At the beginning all CLBs are marked "untested". Starting from each of the primary outputs the circuit partitioner does a depth-first search of the circuit until it reaches a CLB marked "untested". This CLB forms the output of the single-output segment that is to be formed by the circuit partitioner now. Starting from this CLB, the circuit partitioner traces the cone of logic back towards the primary inputs one level at a time by doing a breadth-first search from this CLB and recording at every step the fan-in of the segment formed by the cone of logic up to that point. Once this is done, the largest size cone (number of CLBs) with input fan-in less than N (the user specified threshold) is identified as a testable segment.

The *routability analyzer* checks for the existence of independent paths. Two paths are said to be independent if they do not have a common CLB or interconnect. Given a testable segment the routability analyzer finds out whether the segment inputs that are not primary inputs of the circuit can be routed to primary inputs through independent paths. This is necessary as we need to apply all 2^m patterns to the segment input where m is the number of segment inputs. This is done by a *branch and bound* backtracking algorithm. Once the routing path is identified, the routing can later be done easily by programming the CLBs on the path in the "pass-through" mode. If the *input router* fails to find a routing solution it tries to modify the segment by selectively adding and deleting CLBs from it until it finds a solution. It is to be noted that in every case a routing solution will always be found eventually although the segment size may be very small. This is because a segment comprising a single CLB will always have a routing solution (if the number of primary inputs of the circuit is greater than the number of inputs of the CLB, which is always true). If not then the CLB inputs will also not have independent primary inputs driving it during normal circuit operation. At the end of this step all the CLBs and interconnects that are part of the segment are marked "tested". The whole process is then repeated again and again until all the CLBs and interconnects are marked "tested". The heuristic by which the total number of test configurations is minimized is by trying to maximize the size of each testable segment. The total number of test sessions however can further be minimized by concurrently testing multiple segments as explained below.

Once all the test configurations have been identified, we identify conflicting segments. Two segments S_i and S_j are said to be in conflict if they need to use the same resource in different ways while being tested. For example, if testing segment S_i requires that CLB C_m is to be programmed in "pass-through" mode and testing segment S_j requires that CLB C_m be programmed for normal operation, then it is obvious that these two segments can not be tested concurrently. The conflict is modeled by the *conflict resolution graph* where each vertex corresponds to a testable segment and there is an edge between

two vertices if the corresponding segments are not in conflict. The minimum number of test sessions is then found by a *minimum clique cover* of this graph. Efficient heuristic procedures for solving this NP-hard problem exist.

We have implemented our algorithm. Given a configuration, the configuration is mapped into a graph where each vertex corresponds to a CLB in the configuration and there is an edge between two vertices if the corresponding CLBs have an interconnect between them. The algorithm operates on the graph to generate the set of testable segments. The concurrency analysis is then done to identify segments that can be tested concurrently. The total test time for testing the whole configuration is thus minimized.

Note that our example circuit represents a directed acyclic graph (DAG). Our technique also works for sequential circuits by testing the flip-flops separately. The combinational logic is tested as described, and then additional test sessions are added in which the CLBs are configured (using "pass-through" mode) to create scan paths through the flip-flops to allow them to be tested. More details can be found in [6].

Our algorithm can very easily be integrated with the compiler that generates the configurations. This would enable the compiler to find out the testable segments for a given configuration. It can then download all the testable segment configurations for a given test session on the FPGA and have them tested. Then the next set of testable configurations will be downloaded and tested. This process will be continued until all the segments are tested. The final configuration will then be downloaded onto the FPGA.

4. EXPERIMENTAL RESULTS

The procedure described in this paper was used to generate test configurations for FPGA implementations of the ISCAS benchmark circuits. Results are shown in Table 1. The total number of test configurations required to allow pseudo-exhaustive testing where no segment has more than 20 inputs ($N = 20$) is shown for each circuit. A very small number of test sessions are needed to pseudo-exhaustively test the entire circuit. This is a lot less than that needed by any configuration-independent testing method providing equal fault coverage. Note that this method can be implemented as a self-test and compared to the self-test approach of [8][9] it requires fewer test sessions and yet provides complete fault coverage for both the CLBs and the interconnects.

5. CONCLUSIONS

We have presented a novel technique for rapidly testing the new configuration whenever a reconfigurable computing machine is reconfigured. Our technique is a functional testing approach, and hence it is independent of any structural fault model. It is also independent of the underlying hardware architecture. As a result, it can continue to be used after migrating to new generations of FPGAs. By testing all the test configurations before the actual configuration is downloaded, the correctness and reliability of the system is ensured.

Table 1. Number of Test Configurations

Circuit			FPGA		
Name	**PIs**	**POs**	**CLBs**	**Nets**	**Cnfgs**
c432	36	7	73	293	11
c499	41	32	70	296	2
c880	60	26	119	533	7
c1355	41	32	102	496	4
c1908	33	25	130	558	6
c2670	233	140	262	1160	23
c3540	50	22	428	1904	23
c5315	178	123	654	2752	14
c6288	32	32	1092	5076	17
c7552	207	108	831	3795	26
s9234	36	39	661	2898	5
s13207	62	152	894	3370	13
s38417	28	106	3339	14850	13

ACKNOWLEDGMENTS

This work is part of the ROAR project and is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract Number DABT63-97-C-0024.

REFERENCES

[1] Culbertson, W. B., R. Amerson, R. J. Carter, P. Kuekes, and G. Snider, "Defect Tolerance on the Teramac Custom Computer," *Proc. of the 1997 IEEE Symposium on FPGA's for Custom Computing Machines,* 1997, pp. 140-147.

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

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

[4] Lombardi, F., D. Ashen, X. T. Chen, and W. K. Huang, "Diagnosing Programmable Interconnect Systems for FPGAs," *Proc. Int. Symp. On FPGAs*, 1996, pp. 100-106.

[5] McCluskey, E. J., "Verification Testing - A Pseudo-Exhaustive Test Technique," *IEEE Transactions on Computers*, Vol. C-33, No. 11, pp. 866-875, Nov. 1981.

[6] Quddus, W., "Configuration Self-Test in FPGA-Based Reconfigurable Systems," Masters Thesis, Dept. of ECE, University of Texas at Austin, May 1999.

[7] Renovell, M., J. Figueras, Y. Zorian, "Test of RAM-Based FPGA: Methodology and Application to the Interconnect," *Proc. IEEE VLSI Test Symposium,* 1997, pp. 230-237.

[8] Stroud, C., S. Konala, P. Chen, and M. Abramovici, "Built-In-Self-Test of Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!)," *Proc. IEEE VLSI Test Symposium,* 1996, pp. 387-392.

[9] Stroud, C., E. Lee, S. Konala, and M. Abramovici, "Using ILA Testing for BIST in FPGAs," *Proc. of International Test Conference,* 1996, pp. 68-75.

[10] Zhao, L., D. M. H. Walker, F. Lombardi, "Bridging Fault Detetction in FPGA Interconnects Using I_{DDQ}," *Proc. ACM Int. Symp. On FPGAs*, 1998, pp. 95-103.