Application of Built in Self-Test for Interconnect Testing of FPGAs*

Dereck A. Fernandes
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
dafernan@ecs.umass.edu

Ian G. Harris
Department of Computer Science
University of California Irvine
Irvine, CA 92697
harris@ics.uci.edu

Abstract

Field Programmable Gate Arrays (FPGAs) are becoming more difficult to test due to their increasing complexity and density. Test methodologies for FPGAs consist of generating numerous configurations of programmable switches that connect wire segments to create signal flow paths. We have developed a system that takes an arbitrary FPGA interconnect network and automatically generates configurations for test. These configurations detect single stuck-at faults, pair-wise bridging faults and wire-open faults. We have modified the traditional Ford-Fulkerson Max-Flow algorithm to enable the efficient definition of test configurations. The system also generates the bitstreams, programs the FPGA, and displays the result. We have tested a Xilinx Virtex XCV150 and have presented the number of configurations and the test time for the device.

1 Introduction

Field Programmable Gate Arrays are gaining widespread acceptance in digital design. Such devices are used in many different configurations making it important that manufacturers ensure that each potential configuration does not fail due to device defects. An FPGA is a programmable logic device, which consists of an array of regular logic blocks surrounded by a dense network of wire segments. Programmable memory controls the interconnections between the wire segments and the logic block. A configuration is the data loaded into the programmable memory which determines interconnect and logic functionality. With increased use in high volume and critical applications coupled with increasing device densities, the test methodology and reliability of FPGAs is gaining importance. This paper evaluates the manufacturing test of FPGAs.

We have developed a system to test interconnect in FPGAs and have applied it to test the Virtex XCV150. As seen from Figure 1, our system takes a FPGA interconnect structure and generates various configurations for the stuck-at fault, pair-wise bridging fault and wire-open fault models. These configurations are converted into bitstreams using Xilinx JBits [2], which is used to program the FPGA. These configurations perform a self-test and display the result at the output pins.

To test every stuck-at, bridging and wire-open fault, every wire is connected to every other connectable wire (i.e. needs to have a programmable switch between them), in at least one configuration. Since we can connect only a subset of the possible connections, we need to configure the device many times for complete fault coverage. To reduce the test time it is important to keep the number of reconfigurations to the minimum. Our comprehensive system performs test on a

*This work was supported in part by the National Science Foundation under Grant No. 0081343

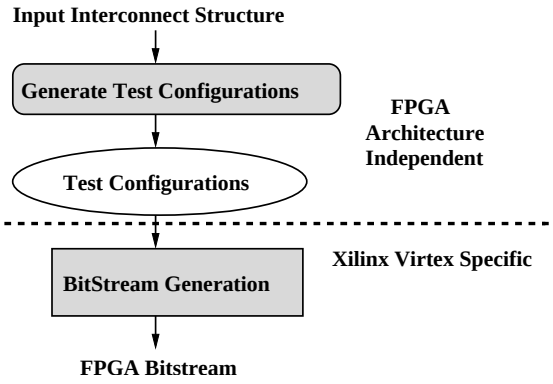

Figure 1: FPGA Test System

commercial FPGA and provides acceptable test times. The organization of the paper is as follows. Section 2 investigates the previous work in this area, and Section 3 describes the island-style FPGA architecture. Section 4 explains the interconnect faults and Section 5 describes the test structure. Section 6 presents the configuration generation algorithm and test application. Section 7 describes the Virtex architecture and the JBits programming tool used to configure the device. Section 8 reports the results of the experiments and Section 9 concludes the paper with a review of future work.

2 Previous Work

FPGA test consists of the logic block test [7] [20], interconnect test of the wire segments surrounding the logic blocks [4] [5] [13] [14] [19], input/output block test, embedded memory test and miscellaneous tests that include delay test [3], clock and power wires test etc. This paper focuses on the test of FPGA interconnect which comprises the majority of the area in modern FPGAs. Since FPGAs have limited IO pads, recent research has focused on Built in Self-Test (BIST) [10]. We also use the BIST technique, which uses part of the FPGA resources as test generation and response analyzer circuitry while testing the remaining circuitry. Previous work has targeted the Xilinx XC4000 series and Lucent ORCA series architectures [8] [9].

3 Island-style FPGA Architecture

Island-style FPGAs primarily contain configurable logic blocks (CLBs) and switch matrices (SM). A **Tile** is composed of a switch matrix and a CLB as shown in Figure 2. We refer to the wires connecting a tile to neighboring tiles as tile I/O (TIO). The switch matrix provides the interconnections between various types of wire segments. The CLB contains a **programmable logic block** composed of look-up tables, flip-flops, and assorted special-purpose logic such as a carry chain. The output of each LUT can be connected to a flip-flop or to an unregistered output. The CLB also contains set of input and output multiplexers (IO muxes) that connect the programmable logic to the switch matrix.

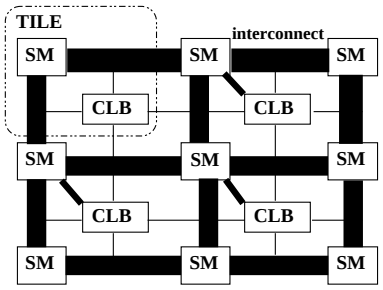

Figure 2: Virtex Architecture

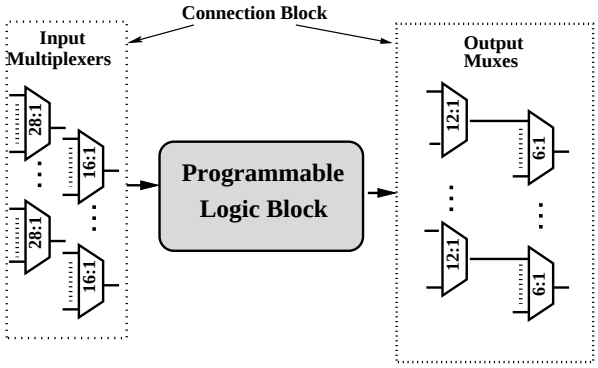

Figure 3: CLB Overview

To eliminate sequential behavior during test, we connect each LUT output directly to an unregistered output, bypassing the flip-flop. As a result, the length of the required test sequence is linear in the number of TIO. We refer to the inputs of a programmable logic block as CI and the outputs of a programmable logic block as CO.

4 FPGA Interconnect Faults

We target four classes of faults as described by Renovell *et al* [8] and one class which is the wire-open fault.

- Permanent Connection: A short circuit between any pair of wires.
- Permanent Disconnection: An open circuit between any pair of wires, which have the programmable switch between them turned on.
- Stuck-At Zero: A short-circuit between a wire segment and ground.
- Stuck-At One: A short-circuit between a wire segment and power.
- Wire-Open: A permanent break in a wire.

5 Test Structure

This paper uses a test structure similar to that described by Stroud *et al* [11]. In each configuration, a portion of the FPGA circuitry contains a test generator, tiles under test and a response analyzer as seen in Figure 4. Both the tiles have an identical logic and interconnect configuration. The LUTs are configured as 4-input XOR gates to pass any fault effect. XOR gates provide good controllability and observability properties at gate inputs and outputs. The LUT output is connected to the unregistered CLB output. All other configurable resources in the programmable logic are set to be off. The same test stimuli are simultaneously applied to the two tiles. The presence of a fault is determined by comparing the corresponding outputs. If their outputs differ, a fault is detected.

The structure of the test generator as shown in Figure 4, is a shift register that shifts a one through zeros (1000..., 0100..., 0010..., 0001...). The outputs of the shift register route to the TIO of the tiles under test. The response analyzer consists of XOR gates that compare the signal values from corresponding output wires from the two tiles. The output of these XOR gates are ORed together and the result stored in a flip-flop which is connected in a loop circuit with an OR gate as shown in Figure 4. This flip-flop output is connected to an LED which lights if a fault is present.

6 Configuration Generation Algorithm

Each FPGA tile configuration is generated by representing the tile interconnect structure as a graph and solving the Max-Flow problem [1] successively to identify paths through which signal should flow. Since all the switch matrices are identical, we require the description of only one switch matrix. The resultant configurations can be applied to any tile that needs to be tested.

6.1 Graph Model of Interconnect

FPGA interconnect is composed of wire segments, programmable pass transistors, and programmable multiplexers which may be implemented as a series of pass transistors. In the graph model, each wire segment is a node, each individual pass transistor is an edge, and each multiplexer is a set of edges. The mapping from FPGA interconnect to graph is shown in Figures 5a and b. Figure 5a shows the mapping of two segments with an intervening pass transistor and Figure 5b shows two segments $s1$ and $s2$, connected to $s3$ by a 2-way multiplexer. It is necessary to ensure that each segment connects to only one driver. We ensure that the Max-Flow

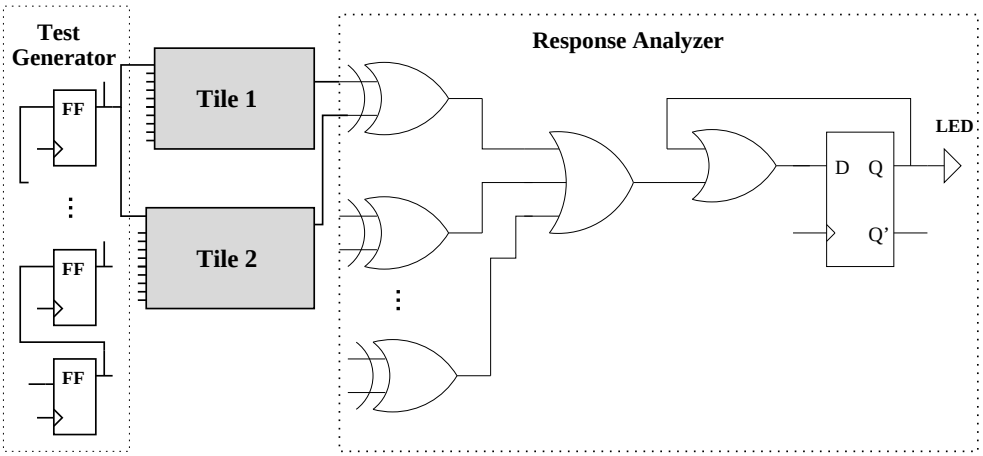

Figure 4: The Detailed Test Structure

process adheres to this constraint by applying a node capacity of one to each node in the graph. In this way, only one driver can push a signal through each segment.

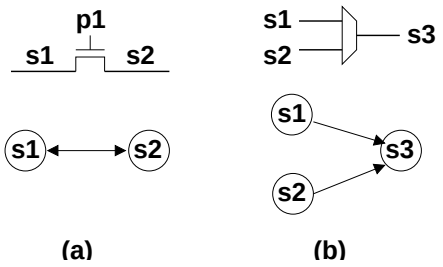

Figure 5: Mapping Interconnect Structure to a graph, (a) two segments connected via a pass transistor, (b) two segments connected to another via a 2-way mux.

6.2 Using Max-Flow

Figure 6 illustrates our configuration generation algorithm that uses Max-Flow. The **Max-Flow** function takes in two arguments, the node type of the source nodes and the node type of the sink nodes. This function executes three times to generate every configuration as described below.

1. **CLB Outputs (CO) to Tile Outputs (TIO):** Paths are identified which connect the outputs of all CLBs to the Tile IO. Due to limited routing resources between CO and TIO this is done first.

2. **Tile IO (TIO) to CLB Inputs (CI):** Paths are identified which drive the inputs of all CLBs from the Tile IO. We need to ensure that the CO is driven with certain values therefore, this is done second.

3. **Tile Inputs (TI) to Tile Outputs (TO):** Since the TIO can be driven bi-directionally the remaining TIO are randomly marked

```
1 Initialize flow and weights ∀ edges
2 repeat
    3 Max-Flow (CO, TIO)
    4 Max-Flow (TIO, CI)
    5 Max-Flow (TI, TO)
    6 Calculate Fault Coverage
    7 Increment the Weight on all edges
    with flow by 1
    8 Reinitialize flow along all edges
10 until Detect All faults
```

Figure 6: Pseudo code for the Configuration generation algorithm

Tile inputs (TI-sources) and Tile outputs (TO-sinks). We generated flow paths that connect Tile inputs to Tile outputs. This step increases the fault observability until all tile inputs and outputs are used.

When generating a configuration, it is important to direct the Max-Flow solution to push flow through segments that are associated with undetected faults. The algorithm associates a *weight* with each edge. The weight of an edge is incremented when an edge has flow through it. The path selection process selects a path that has the minimum net weight, thus the algorithm is encouraged to push flow through edges with lower weights. In this way, untested wire segments will be tested in future configurations.

6.3 Fault Detection Criterion

As seen in Figure 6 for each configuration we calculate the fault coverage based on the following assumptions.

- Permanent connect faults for any two pair of nodes are detected if those two wire segments are in separate paths from source to sink and have flow going through them. Since both these paths will be running opposite values, we can detect if there is a bridge of any type between them.
- A Permanent disconnect fault between any two connectable pair of nodes is detected, if they lie in the same path from source to sink and have flow going through them.
- Stuck-At faults at a node are detected, if the node has signal flow moving through it at all times. This is sufficient because the test generator shifts both a zero and a one along each segment at some point during test. Thus, the wire will be passing a one and a zero at different times.
- A Wire Open fault is detected, if the wire segment has signal flow through it at all times and is driven by a zero and a one at a different point. This is sufficient because the output of this segment will be unknown, and since the test generator drives a zero and a one, the fault will be detected.

6.4 Benefits of Max-Flow Algorithm

The goal of our algorithm is to detect the maximum number of faults in a single configuration with low time complexity. We chose Max-Flow for two reasons. First, the number of faults detected in a configuration is related to the number of segments connected to the TIO. Our modified Max-Flow algorithm increases the number of segments connected to the TIO and therefore increases the faults detected in a configuration. Second, the use of the Max-Flow algorithm results in low computation time because the Max-Flow algorithm has polynomial time complexity. The complexity of Max-Flow remains polynomial for the first two calls in the algorithm. In the third call to Max-Flow we chose TI and TO very randomly from among the TIO, this causes the complexity of the algorithm to be non-polynomial. Therefore, the entire algorithm is non-polynomial in nature.

7 Virtex Architecture and JBits

The Virtex FPGA consists of configurable logic blocks (CLBs), switch matrices (SM), on-chip RAM (BRAM), and input, output blocks (IOBs). We will only describe the CLBs and SMs since our work is restricted to them.

7.1 Configurable Logic Blocks

In a Virtex FPGA, each CLB consists of two slices. Each slice contains two LUTs. A LUT is a 4-input function generator. There are 13 inputs to every slice. These inputs contain four inputs for each of the two LUTs and control inputs; clock, clock enable, synchronous set, syn-

chronous reset and a direct connection input. The LUT and inputs can be combined to represent functions of various sizes. For each input, there is a corresponding multiplexer that connects the input to the switch matrix. There are eight 28:1 input multiplexers for the LUT inputs and five 16:1 input multiplexers for the control inputs (clock, set, reset etc). Our work considers only 16 of 26 LUT inputs omitting control inputs Clock, Set, Reset, etc. A slice has six outputs, two unregistered, two registered and two carry outputs. The outputs of each slice are connected to eight 12:1 output multiplexers. The output of these multiplexers are each connected to a eight 6:1 multiplexers that connect to wire segments in the switch matrix and to the input of the LUTs. We consider only four of twelve CLB outputs omitting the registered and carry outputs.

7.2 Switch Matrix

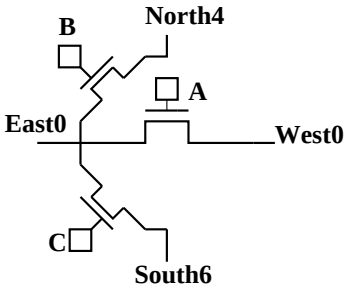

Figure 7: Switch Matrix Detail

The switch matrix contains the routing resources to connect a CLB to adjacent and distant CLBs. It contains 24 single wire segments, to route signals to adjacent SMs in each of the four directions [17] [18]. It also has 12 buffered hex wire segments, to route SM signals to other SMs six blocks away in each of the four directions. Finally, it contains 12 long wire segments that span the vertical height and horizontal width of the device. A direct connection wire provides a direct routing resource between two horizontally adjacent CLBs. There are four horizontal tri-state busses per CLB row that provide dedicated busses for the CLBs and there are two wires per CLB that propagate carry signals vertically to the adjacent CLBs bypassing the SM. In Figure 7, if the *A* SRAM bit is set to 1 then the wires East0 and West0 are connected. A Virtex XCV150 contains a array of 24×36 tiles. Each tile has a switch matrix that has 960 programmable switches (SRAM bits). Of these, 288, connect single wire-segments to other single wire-segments.The remaining switches connect the 96 single, 48 hex, tri-state busses and long wire-segments to the 26 inputs and 12 outputs of the CLB.

7.3 Tested Virtex Interconnect

We only test the general purpose interconnect in the FPGA. The special purpose interconnect that consists of global buses, IO routing, clock wires and tri-state busses are not tested. Within the general purpose interconnect we restrict our test to single wires, and omit hex and long wire segments. In each tile, we test 124 out of 142 wire segments and set 776 out of 960 switches.

7.4 The JBits Interface

JBits is a Xilinx bitstream interface, based on a set of Java classes, which provides an Application Specific Interface (API) for the FPGA bitstream. This interface permits all configurable resources in the device to be individually set. The following is a short code snippet that illustrates its programming capability.

```
jbits.set(0,1,OutMuxToSingle.
              OUT6_TO_SINGLE_NORTH20,
          OutMuxToSingle.ON);
```

This line of code uses the JBits set method to connect the output multiplexer 6 in row 0 and column 1 of the FPGA to the single wire North20 in the same tile. A proof of concept for testing FPGAs using JBits was presented by Sundararajan, *et al* [12].

We use the JBits JRoute [6] tool to do the routing between the shift register outputs and TIO, between TIO and the response analyzer and between the flip-flop and the Input/Output Blocks of the FPGA.

7.5 The JBits RTPCore Interface

JBits provides a Java core library, with cores for shift-register, logic gates, counters, etc. These cores perform the placement and assignments of nets. Our test generator is a shift-register constructed from the JBits flip-flop core. The four flip-flops in a CLB are connected in a chain. The shift-register is placed in a column fashion. The test generators are typically 10-11 CLBs in height and one CLB wide. The response analyzer is constructed from the XOR gate, wide OR gate and flip flop cores. The response analyzer core is two CLBs columns wide and twenty CLBs rows tall. In the response analyzer the first column implements the XOR gates and the second column the wide OR gate.

8 Experimental Results

A Xilinx Virtex XCV150 device was used to verify our technique. We have instantated the test structure shown in Figure 4 on the XCV150 device. The *configuration generation time* - the time required to generate FPGA test configurations, and the *test application time* - the time required to test a single FPGA part using the test configurations were determined through experimentation.

8.1 Configuration Generation Time

The Max-flow algorithm generated 59 test configurations per tile for the Xilinx Virtex XCV150 device. The algorithm was executed on a 1.6 GHz Pentium 4 machine, with 256 MB RAM, running RedHat Linux 8.0. The total time required to generate all 59 configurations was 2045.85 seconds. We used a random number seed of 1 to select which TIO would be TI and TO. The Stuck-At fault coverage requires *four* configurations. The permanent connect fault coverage requires *eleven* configurations, while the permanent disconnect fault / line-open coverage requires *59* configurations. The bitstream generation time is the time required for the Java JBits program to generate a device bitstream. Our bitstream generation results were obtained on a Pentium II 366 MHz machine with 256 MB RAM running Windows 2000 operating system and having JDK 1.2.2. Different computers were used because JBits only works on Windows platform. JRoute is a maze router that executes a fixed number of tries to route a particular set of nets. It takes the same time on average to route nets of similar widths at the same (X,Y) location. The JBits runtime for each configuration was 41.5 seconds. Thus for 59 configurations the runtime is 2452.6 seconds. A total of 4498.5 seconds were required to generate FPGA programmable bitstreams from our algorithm. This time is relatively insignificant because test generation needs to be performed only once for each FPGA product line.

8.2 Test Application Time Computation

The total test application time is the sum of the *reconfiguration time* - the time required to reconfigure the FPGA for all 59 configurations, and the *pattern time* - the time required to apply all test patterns in each configuration.

8.2.1 Reconfiguration Time Estimation

The Xilinx AFX-BG352 prototype board contains a Virtex XCV150 device with a 50 MHz Clock. A Xilinx Parallel III cable connects the board and the computer. The configuration clock for the FPGA was 4 MHz and the Slave Serial Mode was used to program the device. In real production testing, the configuration clock could be set as high as 66 MHz and the SelectMap programming mode that writes 8

bits/clock rather than 1 bit/clock in Slave Serial could be used. In addition, production test would use Automatic Test Equipment (ATE), which would eliminate the speed limitations we have due to the download cable and operating system. This paper presents the estimated configuration time for an ATE using the SelectMap mode of programming with a 66 MHz clock.

The Virtex configuration memory bits are grouped into vertical frames that are one bit wide and have a fixed length depending on the FPGA. The FPGA reconfiguration mechanism consists of a startup activity that activates the circuitry that identifies the frame length and the frame address. It then loads the frame into a temporary register and performs a parallel Cyclic Redundancy Check (CRC) on the frame data. The configuration controller compares this value to the value embedded in the bitstream. Once the CRC check passes, the configuration controller writes the entire frame in parallel into the location specified in the bitstream. The XCV150 has a configuration bitstream of 1,040,096 bits.

Thus, the reconfiguration time is,

$$StartupTime + (ShiftFrameTime + CRCTime) \quad (1)$$

1. **Startup Time** is the time required to start the process of configuration. It involves loading and execution of around 39 32-bit words from the bitstream. This process takes $39 * 32 * (1/66MHz) = 18.9$ μs.

2. **Shift Frame Time** is the time required to write the frame into the appropriate location. A Virtex XCV150 has 48 frames per column and 36 columns. It has a frame size of 512 bits. Therefore, the time to write the frame bits is $(36 * 48 * 512) * (1/(66 * 8)MHz) = 1.6ms$. The 8 in the denominator is the bits/clock write for SelectMap.

3. **CRC Time** is the time to do a parallel CRC on 512 bits. The generated CRC value is compared with the CRC value embedded in the bitstream. Therefore, this time is $512 * (1/66MHz) = 7.7$ μs.

Thus, configuration time for one configuration is approximately 1.7 msec. The time required for all 59 configurations is 100.3 msec.

8.3 Pattern Time

The time required to apply all test patterns in a single configuration is dependent on the size of the shift register. The largest shift register we required was of 53 bits. We assume that the FPGA uses the 66 MHz functional clock frequency. The time required to apply all 53 patterns in a single configuration is $53 * (1/66) = 0.7$ microseconds. We estimate a bound for the total time required for all the configurations by scaling this time to all configurations. Therefore, $59 * 0.6 = 46.9$ microseconds was required to apply test patterns in all 59 configurations.

8.4 Total Test Application Time

The total test application time for 59 configurations, found by adding the configuration time and the pattern time, is **100.3 mseconds + 0.046 ms = 100.34 ms**.

9 Conclusions and Future Work

This paper presents a new and practical technique for testing FPGA interconnects. The results of this technique are very encouraging. Our technique does not add any hardware overhead to the FPGA architecture. The configuration generation mechanism is generalizable and can be applied to island-style FPGAs. However, the test application mechanism, which consists of generating the bitstream, is currently restricted to the Xilinx Virtex series because JBits only supports the Virtex series. For other Xilinx FPGAs the circuit description could be written in

the Xilinx Design Language (XDL) and the circuit could be routed with Xilinx Place and Route Tools (PAR) or other third party tools. Unlike JRoute, where one can completely lock the routing that is required for test, other commercial routers like PAR use the rip-up and re-reroute method that removes part of the routing. The goal of all commercial routers is to complete the route rather than to use specific wire resources, this has been a major problem to all FPGA interconnect test methods. An ideal router solution would rip-up and re-route route with the ability to route to particular wire segments and not just LUT inputs. JRoute has the ability to route to particular wire segments but does not have a rip-up and reroute capability. Hence the routes provided by JRoute can be very inefficient. Although we have avoided specific interconnect in the FPGA like hex wires and long wires, our approach could be enhanced in the future to incorporate them. Stuck-At fault coverage for hex wires and long wires is not difficult but the detection of bridging faults between single wires and the hex and long wires is quite a problem because the observability of these wires could span multiple blocks. Since we consider system test, we would investigate how multiple configurations could be stored in a EEPROM and programmed recursively into the FPGA using a CPLD. This would make the FPGA test system self contained [15].

10 Acknowledgments

The authors are very grateful to Mr. Alex Carreira, graduate student at the University of Calgary, Alberta for his immense support in our JBits work and for providing the Shift-Register and Response Analyzer Cores. The authors also want to thank Xilinx Corporation for their generous donation of the XCV150 devices, AFX-BG352 prototype board and the Xilinx Foundation series software. We also thank Mr. Prasanna Sundarajan and Mr. Eric Keller of the JBits team at Xilinx for helping us use JBits and JRoute.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.

[2] S. A. Guccione and D. Levi. XBI: A Java-Based Interface to FPGA hardware. *In John Schewel, editor, Configurable Computing: Technology and Applications, Proc. SPIE 3526 SPIE - The International Society for Optical Engineering*, November 1998.

[3] I. G. Harris, P. Menon, and R. Tessier. BIST-Based Path Delay Testing in FPGA Architectures. In *International Test Conference*, pages 932–938, October 2001.

[4] I. G. Harris and R. Tessier. Interconnect Testing of Cluster-based FPGA Architectures. In *Design Automation Conference (DAC)*, pages 49–54, June 2000.

[5] 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*, 21(11), November 2002.

[6] E. Keller. JRoute: A Run-Time Routing API for FPGA Hardware. *7th Reconfigurable Architectures Workshop (RAW 2000)*, May 2000.

[7] M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. SRAM-based FPGAs: Testing the LUT/RAM modules. In *International Test Conference*, pages 1102–1111, October 1998.

[8] M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. Testing the Interconnect of RAM based FPGAs. *IEEE Design and Test of Computers*, 15(1):45–50, Jan-Mar 1998.

[9] M. Renovell and Y. Zorian. Different Experiments in Test Generation for Xilinx FPGAs. In *International Test Conference*, pages 854–862, October 2000.

[10] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici. BIST-Based Diagnosis of FPGA Interconnect. In *International Test Conference*, pages 618–627, October 2002.

[11] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self Test of FPGA Interconnect. In *International Test Conference*, pages 404–411, October 1998.

[12] P. Sundararajan, S. McMillan, and S. A. Guccione. Testing FPGA Devices Using JBits. *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, September 2001.

[13] M. B. Tahoori, S. Mitra, S. Toutounchi, and E. McCluskey. Fault Grading FPGA Interconnect Test Configurations. In *International Test Conference*, pages 608–617, October 2002.

[14] S. Toutounchi and A. Lai. FPGA Test and Coverage. In *International Test Conference*, pages 559–607, October 2002.

[15] Xilinx Application Notes. *Configuring Virtex FPGAs from Parallel EPROMs with a CPLD XAPP137 (v1.0).* Xilinx, 1999.

[16] Xilinx Application Notes. *Xilinx Prototype Platforms User Guide for Virtex and Virtex-E Series FPGAs DS020 (v1.1).* Xilinx, 1999.

[17] Xilinx Databook. *Virtex 2.5V FPGA Detailed Functional Description DS003-2 (v2.6).* Xilinx, 2002.

[18] Xilinx JBits. *JBits SDK Version 2.8 for Virtex.* Xilinx, 2001.

[19] L. Zhao, D.M.H Walker, and F. Lombardi. Bridging fault detection in FPGA interconnects using IDDQ. In *International Symposium on Field Programmable Gate Arrays*, pages 95–104, February 1998.

[20] L. Zhao, D.M.H Walker, and F. Lombardi. Detection of Bridging Faults in Logic Resources of Configurable FPGAs using IDDQ. In *International Test Conference*, pages 1037–1046, October 1998.