In-System Testing of Xilinx 7-Series FPGAs: Part 1-Logic

Harmish Modi and Peter Athanas
Dept. of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24060
hmodi89@vt.edu Peter.Athanas@vt.edu

***Abstract*—FPGA fault recovery techniques, such as bitstream scrubbing, are only limited to detecting and correcting soft errors that corrupt the configuration memory. Scrubbing and related techniques cannot detect permanent faults within the FPGA fabric, such as short-circuits and open-circuits in FPGA transistors that arise from electromigration effects. Several Built-In Self-Test (BIST) techniques have been proposed in the past to detect and isolate such faults. These techniques suffer from routing congestion problems in modern FPGAs that have a large number of logic blocks. This paper presents an improved BIST architecture for all Xilinx 7-Series FPGAs that is scalable to large arrays. The two primary sources of overhead associated with FPGA BIST, the test time and the memory required for storing the BIST configurations, are also reduced when compared to previous FPGA-BIST approaches. The BIST techniques presented here also eliminate the need for using any of the user I/O pins, such as a clock, a reset, and test observation pins; therefore, it is suitable for immediate deployment in any system with Xilinx 7-series FPGAs.**

Keywords—FPGA; Built In Self Testing; Iterative Logic Array; Configurable Logic Block

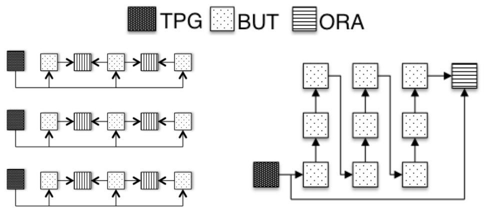
I. INTRODUCTION

The abundant availability of logic resources and the reconfigurability make FPGAs the perfect candidate for space, aviation, and military applications such as software-defined radios and synthetic aperture RADAR. The harsh environments present in space and military applications catalyze the faster degradation of the FPGAs; therefore, these FPGAs are more prone to logic failures [1]. The faults in FPGAs can be broadly classified into two categories. The first is soft-errors such as single bit-flips in the configuration memory, which can be corrected by refreshing it. Bit-stream scrubbers [1] are well known to combat the soft-errors. The second category of faults is permanent faults such as the faults arising from electromigration effects and time-dependent dielectric breakdown. These faults are observed as short circuits and open circuits in FPGA transistors. The bitstream scrubbers and related techniques cannot detect such faults. These faults, by nature, are not correctable, and the only alternative is to bypass such fault locations. Fortunately, FPGAs can be reconfigured to avoid such faulty locations given that the fault can be detected and isolated in some way. This motivation has led to the development of various test methods to detect and isolate the permanent faults within the FPGAs. A test that can detect such faults without involving the dedicated external test equipment is known as Built-In Self-Test (BIST).

FPGAs consist of a large number of Configurable Logic Blocks (CLBs) arranged in a regular 2-D array. Each CLB can be reconfigured to implement different logic functions. Each CLB also has an adjacent switching matrix, which connects it to the rest of the FPGA using wires. A comprehensive solution has been created to test both the CLBs and the routing resources. This paper particularly discusses the testing of the CLBs. Testing of the routing resources is addressed in another publication [14]. The regular arrangement of the CLBs within the FPGA and its reconfigurability are exploited to create the BIST for the CLBs [2]. The conventional strategy behind the CLB testing is to configure some of the CLBs as Test Pattern Generators (TPGs), and others as Output Response Analyzers (ORAs). The rest of the CLBs are referred to as Blocks Under Test (BUTs). The TPGs provide input vectors to the BUTs, and their response is validated by the ORA. In a single test session, all of the possible modes of the BUTs are tested using different configurations. In another test session, the role of the BUTs is swapped with the TPGs and ORAs through reconfiguration, and in this way, all of the CLBs within the FPGA are tested in two test sessions. Once the off-line testing is complete, the FPGA can be reconfigured again into normal operation, and the BIST logic disappears – this solution is free from the area overhead of conventional always-present BISTs. As the BIST is self-contained, it can be applied at any level starting from manufacturing test to in-system test. The sources of overhead associated with this approach are the additional memory that stores the BIST configurations and a small system downtime during the offline testing.

II. BACKGROUND AND MOTIVATION

The early BIST work [3][4] was focused on Xilinx-4000, Spartan and Virtex-4 FPGA architectures. These solutions were not adequate to test the newer FPGA architectures (Virtex-5 onwards) because the CLBs in newer architectures contain more data-paths and more interface pins as compared to the previous generation FPGAs. Stroud presented the BIST for Xilinx Virtex-5 devices [5], however this work still relied on the older test architectures. In these BIST architectures, a single TPG provided the input vectors to multiple BUTs using long wires as shown in Fig. 1(a). High fan-out on the TPGs, limited availability of the long wires and the synchronization difficulties related to them made the routing difficult. Hence,



(a) (b)

Figure 1. (a) BIST using long wires (b) BIST using Iterative Logic Array

these techniques failed for larger devices, which have a large number of CLBs, due to the routing congestion [10].

The BISTs, using Iterative Logic Array (ILA), were presented to combat the problem of routing congestion. In the ILA architecture, as shown in Fig. 1(b), instead of a single TPG providing the input vectors to multiple BUTs, the output of one BUT is propagated as the input to its adjacent BUT. This way, each BUT acts as a TPG for its successor BUT. If a fault exists in the BUT, then the BUT will provide an incorrect input to its successive BUT, and, hence, the error will propagate until it reaches the final ILA output. Early ILA-based work [6][7][8][9] provided test coverage only on the limited CLB functionalities. Stroud and Abramovici [10] provided full test coverage for ORCA CLBs. In this approach, each BUT required one extra CLB as a helper cell to form the ILA, and, therefore, the test required two test sessions (one session to test the BUT, and a second session to swap the roles of the BUT and the helper CLB). The CLBs in the contemporary Xilinx FPGAs are significantly different than the ones in the ORCA FPGAs, and, therefore, the test for the ORCA FPGAs cannot be extended to the contemporary Xilinx FPGAs.

Contemporary BIST techniques require a limited number of user I/O pins, such as a clock, a reset, and the observation pins, in order to administer the test. As a result of this, the BIST has to be designed prior to the system design phase of the project, and may be subjected to constraints that cannot be satisfied. The BIST discussed in this paper eliminates the need for any of the user I/O pins, and thereby makes it suitable for immediate deployment in any system with Xilinx 7-series FPGA.

This paper provides the following contributions in the field of FPGA-BIST. 1) The test detects and isolates single stuck-at-fault in the CLBs of Xilinx 7-series FPGAs 2) The test ensures the full scalability across different device sizes by eliminating the routing congestion problem for the larger devices. 3) The test completes in a single test session, thereby reducing the test time. 4) The need for the user I/O pins is eliminated; hence, the test can be deployed in any system without making any system-level changes.

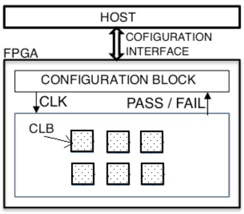
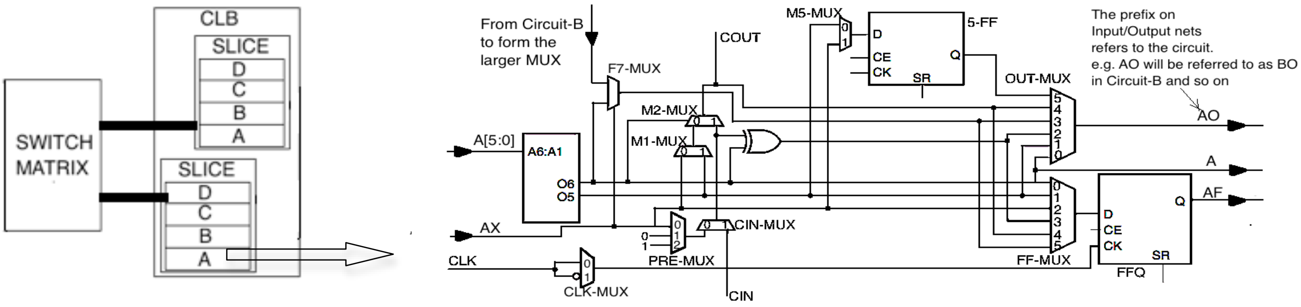


Figure 2. System-level diagram of the test

III. Overview of the Test

Fig. 2 shows the system-level diagram of the test setup. The test clock is derived internally from the configuration block of the FPGA. The result of the test is written into the DONE bit of the FPGA status register, which can be read by the configuration interface. The test involves the following sequence of steps – 1) The host downloads a BIST configuration to the FPGA using the configuration interface, such as SelectMap or JTAG 2) The test runs for a small duration and writes a PASS or FAIL result into the status register of the FPGA 3) The host reads back the result of the test using the same configuration interface. 4) If the test result is FAIL, only then the host performs the fault isolation. All of the required BIST configurations are precomputed, and are stored in external memory along with the user configuration. The generation of the BIST configurations is automated using a script, which takes the FPGA part name and the coordinates of the rectangular test area as the input parameters, and generates the set of BIST configurations to test all of the CLBs within the specified rectangular test area. The BIST configurations are discussed in detail in Section IV. If the fault is detected by the BIST configurations, only then all of the flip-flops in the device are read back using the configuration interface to isolate the fault location. Fault isolation is discussed in detail in Section V.



The prefix on Input/Output nets refers to the circuit. e.g. AO will be referred to as BO in Circuit-B and so on

(a) (b)

Figure 3. (a)Structure of CLB (b) Logic Circuit A within the SLICE

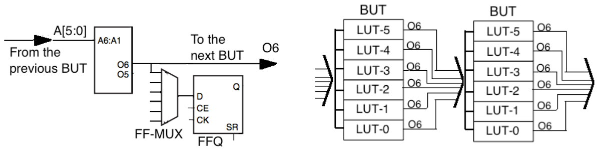
*The nomenclature shown in this figure is used throughout the paper to refer the different MUX, MUX inputs, FF and nets.

IV. BIST CONFIGURATIONS

Fig. 3 shows the internal architecture of the CLB [11]. The nomenclature shown in this figure is used throughout the paper to refer the different MUXs, MUX inputs, FFs, and nets. Each CLB is composed of two SLICE blocks, which in turn are composed of four identical logic circuits, referred to as circuits A, B, C, and D. All of the LUTs and the data-paths within the SLICE are tested using the Iterative Logic Array (ILA) architecture as described in sections IV.A and IV.B. A subset of the SLICEs (usually 1/3rd of the total SLICEs) can configure their LUTs as RAM and shift-registers. Testing of these modes is discussed in sections IV.C and IV.D.

A. LUT Testing

In order to cascade the LUTs to form an ILA (Fig. 1(b)), the output bus width of the LUT should match the input bus width of its successor. However, Fig. 3 shows that each LUT has six input address pins - A[5:0] and only two output pins - O5 and O6. In order to balance the mismatch in bus widths, six LUTs are grouped to form a single BUT. The output O6 of each LUT is used to form the output bus (of width six). This output bus is connected as the shared input address bus to all of the six LUTs in the successive BUT as shown in Fig. 4(b). The output O6 is also registered into the flip-flop to enable the fault isolation, which is discussed in detail in Section V.



(a) (b)

Figure 4. (a) Individual LUT in the BUT (b) Cascading of the BUTs

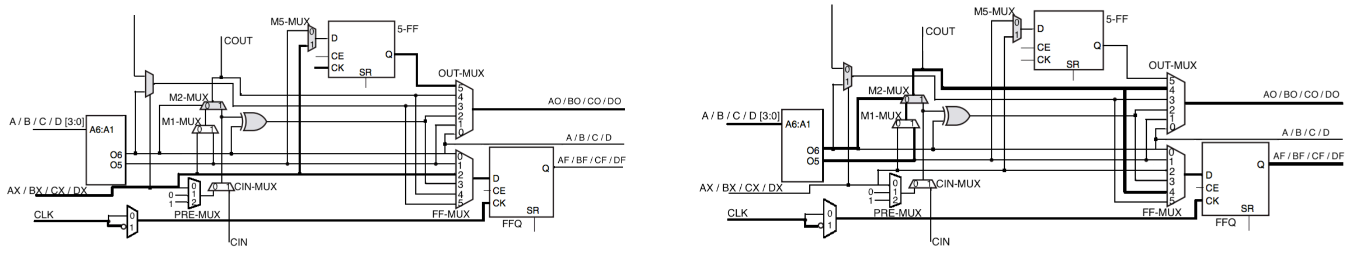
The final output of an ILA is the cumulative function of all of the functions that are implemented in each of the cascaded BUTs. To simplify the cumulative function, the LUTs of the BUT are configured in such a way that the BUT implements the identity function, i.e. the output of the BUT is identical to the input applied to the BUT. For example, by applying the input address as 6'b000000, the 0th memory location in all of the LUTs is selected. In order to get the output of the BUT as 6'b000000, the 0th memory location of each LUT is programmed as a logic-0. If all of the BUTs were fault-free, then an ideal output of the ILA would be 6'b000000. However, if any LUT has a stuck-at-1 (S-A-1) fault in the 0th memory location, then the output of the BUT containing the faulty LUT would differ from the ideal output (6'b000000). As all of the subsequent BUTs are also implemented as the identity functions, the faulty output will propagate until it reaches the final ILA output. The Test Pattern Generator (TPG) provides all of the possible 2^6 address vectors to the ILA to test each memory location of the LUT. The Output Response Analyzer (ORA) compares the final output of the ILA with the input applied by the TPG, and any mismatch will be reported as the presence of a fault in the ILA. However, the identity function would test each memory location of the LUT for either a stuck-at-0 or stuck-at-1 fault, but not for both of the faults. For example, the identity function tested the 0th memory location for a stuck-at-1 fault only. However, to test a stuck-at-0 fault in the 0th memory location of the LUT, another configuration is required. In this configuration, all of the BUTs are configured to implement the complement function. For example, if the input 6'b000000 is applied to the BUT, then the ideal output of the BUT should be 6'b111111. This implies that the 0th memory location of each LUT should be programmed as a logic-1. If any of the LUTs has a stuck-at-0 fault in the 0th memory location, then the output of the BUT containing that LUT, and subsequently the output of the ILA would deviate from the ideal output. In this way, both of these functions together test the stuck-at-0 and the stuck-at-1 faults in each memory location of the LUT.

Faults in the address decoder logic of the LUT, which are only visible in the gate-level model, can be tested by swapping the six LUTs of the BUT among themselves [15], i.e. in the second phase, LUT0 will become LUT1, LUT1 will become LUT2, and so on. Swapping of the LUTs will yield to a total of 12 configurations - 6 for the identity function and 6 for the complement function. These 12 configurations are sufficient to test all of the gate-level faults in the LUT [15].

The TPG is implemented as a 6-bit counter that generates all of the required 2^6 address vectors. The ILA, as a whole unit, implements the identity or complement function, and therefore, the ideal output should be the identical or complementary to the input applied by the TPG. The function of the ORA is to compare the output of the ILA with the input applied by the TPG. The TPG and the ORA are implemented inside a single DSP resource, and do not use any of the CLBs. This makes it possible to test all of the LUTs in a single test session.

B. Data-path Testing

The multiplexers (MUXs), flip-flops, and XOR gates are tested by creating a data-path through these components, and then by exciting the data-path to a logic-1 and a logic-0. Fig. 5(a) shows one possible data-path within the SLICE. The same data-path can also be represented using only the MUX settings as shown in Configuration 6 in Table I. By applying the input



(a) (b)

Figure 5. (a) Configuration-6 in Table I (b) Configuration-3 in Table I

TABLE I. DETAILS OF DIFFERENT CONFIGURATION FOR TESTING THE DATAPATHS

#	MUX Settings								Connectivity Between Successive SLICEs
	CLK	***OUT***	***FF***	***M1***	***M2***	***M5***	***CIN***	***PRE***	

1	0	0	0	X	X	X	X	X*	S0(AO, AF) → S1(A/B[0], C/D[0])**
									S0(BO, BF) → S1(A/B[1], C/D[1])
									S0(CO, CF) → S1(A/B[2], C/D[2])
									S0(DO, DF) → S1(A/B[3], C/D[3])

2	1	1	1	1	0	X	X	X	Same as 1
3	0	4	4	1	0	X	X	X	Same as 1
4	1	4	4	X	1	X	0	2*	Same as 1
5	0	4	4	X	1	X	0	1	Same as 1
6	1	5	2	X	X	1	X	X	S0(AO, BO, CO, DO)→ S1(AX, BX, CX, DX)
7	0	2	3	0	0	X	0	0	Same as 6
8	1	2	3	0	0	X	0	0	S0(AF, BF, CF, DF)→S1(AX, BX, CX, DX)
9	0	5	X	X	X	0	X	X	Same as 6
10	0	5	2	X	X	1	X	X	Same as 8
11	X	X	X	1	1	X	1	X	S0(COUT) → S1(CIN)
12	0	3	5	X	X	X	X	X	Same as 1 and AX/BX/CX/DX are tied to 0
13	1	3	5	X	X	X	X	X	Same as 1 and AX/BX/CX/DX are tied to 1

*X means MUX is configured to be OFF, any other number means the particular MUX input line in the MUX is selected.

** S0(AO, AF) → S1(A/B[0], C/D[0]) can be interpreted as: The output net AO of SLICE S0 is connected to the 0th address bit of the A and B LUTs in the successive SLICE S1, whereas the output net AF of SLICE S0 is connected to the 0th address bit of the C and D LUTs in the successive SLICE S1.

as a logic-0, and then a logic-1 to the AX line, all of the logic nodes on the highlighted path can be tested for a stuck-at-1 and a stuck-at-0 fault respectively. All of the four circuits A, B, C, and D within a single SLICE are configured identically and are excited with the identical inputs to test them in parallel. Table I summarizes all of the required configurations, which cumulatively test all of the MUXs, flip-flops, and XOR gates within a SLICE.

If the data-path does not involve LUT, then the SLICE inputs propagate to the SLICE outputs through only the MUXs and flip-flops as shown in Fig. 5(a). As the MUX and the flip-flop act just as the pass-through, the SLICE output should be the same as the SLICE input in the absence of any fault. However, if the data-path involves a LUT, then the LUTs within the SLICE are configured in such as way that the SLICE implements the identity function. For example, consider Configuration 3 illustrated in Fig. 5(b). The LUTs in this configuration are configured such that the O6 is always set to a logic-0 to select the highlighted path in the figure. The O5 outputs of all of the LUTs in a single SLICE are implemented in such a way that the output of the SLICE is same as the input applied to the SLICE in the absence of any fault, i.e. the output buses {DO, CO, BO, AO} and {DQ, CQ, BQ, AQ} of the SLICE will have the same values as the input A/B/C/D[3:0] applied to the SLICE. In order to test all of the SLICEs in the device simultaneously, they are cascaded to form an ILA. Column 2 of Table I specifies the connections between the successive SLICEs in the ILA. As each SLICE is implemented as the identity function, the final ILA output should be identical to the input applied by the TPG. The ORA can be implemented as a comparator just as the one described for the LUT testing.

The faults on the selection line of the MUX are only visible in the gate-level model. Fig. 6 illustrates the testing of these faults. While testing a particular input through the MUX, all of the other inputs of the MUX are maintained at a logic-1 to cover a stuck-at-1 fault on the selection line [10]. Consider the highlighted fault on the selection line as shown in Fig. 6(a). This fault is invisible to the MUX output as it is not present on any of the data-paths through the MUX. However, while testing the data-path through the input I0 for a logic-0, if the inactive input (I1) is held at a logic-1, then the output Y of the MUX is observed as a logic-1. In the absence of this fault, the output Y would be a logic-0, and, hence, the fault on the selection line becomes visible at the output of the MUX. A stuck-at-0 fault (Fig. 6(b)) on the selection-line will block the corresponding input (I1 in the figure) to propagate to the MUX output, and hence a stuck-at-0 fault on the selection-line of MUX can be observed at the MUX output (Y in the figure).

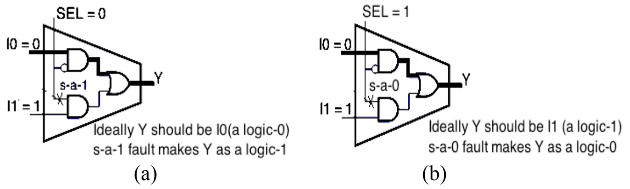

SEL = 0
I0 = 0
s-a-1
Y
I1 = 1
Ideally Y should be I0(a logic-0)
s-a-1 fault makes Y as a logic-1
(a)
SEL = 1
I0 = 0
s-a-0
Y
I1 = 1
Ideally Y should be I1 (a logic-1)
s-a-0 fault makes Y as a logic-0
(b)

Figure 6. (a) Testing a S-A-1 fault on the selection line of the multiplexer (b) Testing a S-A-0 fault on the selection line of the multiplexer

C. *SelectRAM testing*

A subset of the SLICEs has the ability to configure their LUTs as RAM. These SLICEs are known as SLICEMs. The ratio of the total number of the SLICEs to the number of the SLICEMs is usually 3. Taking advantage of this fact, the SLICEs, which have already been tested using the ILA architecture and don't have the capability to be configured as the RAM, are used to build multiple distributed instances of TPG as shown in Fig. 7(a). Each TPG generates the required vectors to perform the MATS (Modified Algorithmic Test Sequence) [13] test on the RAMs. To constrain the routing locally, each TPG provides the input only to its nearby RAMs. A total of 3 configurations are required to test all of the three possible modes (32x2 dual port, 32x2 single port, 64x1 single

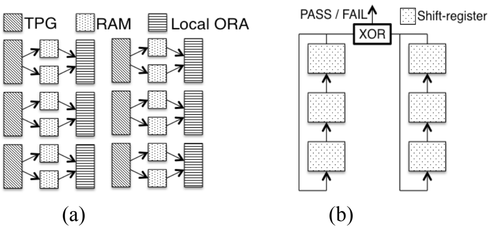


Figure 7. (a) Testing of RAM logic (b) Testing of shift-register logic

port) of the RAM. The output of two nearby RAMs is compared in the adjacent SLICE, referred as local ORA, and any mismatch is registered as a logic-1 in the flip-flop of the local ORA. If the logical OR of all of the local ORA outputs is found to be a logic-1, then it indicates the presence of a fault in at least one RAM.

D. *Shift-Register Testing*

Each LUT in a SLICEM can be configured as a 32-bit shift register or a 16-bit shift register. Two separate configurations are created to verify both of these modes separately. To verify the shift-registers, the device is divided into two regions. All of the shift-registers within a single region are cascaded to form a longer circular shift-register as shown in Fig. 7(b). Both of the circular shift-registers are then loaded with the alternate 1s and 0s (101010…) bit-pattern. All of the shift registers are provided with the same clock, and the bit-0 of both of the shift-registers is compared with each other. If the test runs for enough time, then any stuck-at fault in the shift-register would eventually be observed at the bit-0. This way, in any clock cycle, if the ORA finds a mismatch in the bit-0 of the two shift-registers, it is indicated as the presence of a fault in the shift-register.

V. Fault Isolation

Fault isolation is carried out to determine the location of a fault within the FPGA. Such information is useful because if the faulty CLB can be isolated, then the CLB can be avoided in the future designs using location constrained synthesis. The fault isolation method discussed in this paper relies on the partial readback capability of Xilinx FPGAs [12]. With this capability, a host can read back the state of all of the memory components in the device. Upon detection of a fault, a signal indicating the presence of the fault (output of the ORA) can be used to stop the TPG by de-asserting its Clock Enable (CE) signal. Once the TPG is shutdown, the FPGA preserves the faulty state. At this time, the states of all of the flip-flops in the device are read back, and this information is further analyzed to narrow down the faulty CLB.

While testing the RAM logic, each local ORA registers a PASS/FAIL result of its nearby RAM, and, therefore, the location of the faulty RAM is determined by finding the local-ORA with its flip-flop value as a logic-1. Any fault in the shift-register logic will cause all of the subsequent bits after the fault location to have the same value (all 0s or all 1s), whereas in an ideal case, the bits in the shift-register would always have the alternate 1s and 0s bit-pattern. By determining the location from where the deviation in the bit-pattern occurs, the fault location can be narrowed down.

In the ILA architecture, a fault always propagates through the flip-flop. Because the order in which the BUTs are connected in an ILA is known, the first BUT with non-ideal flip-flop values can be found, and this information is used to narrow down the fault location. For example, assume that Configuration 3 (Fig. 5(b)) is in operation, and the input to the OUTMUX in Circuit A has a stuck-at-1 fault in some SLICE S0. Because of this fault, the output pin AO of the S0 will be at a logic-1 even when the input vector 8'h00 is applied to the ILA. The AO pin of the S0 drives the 0th address bit of the A-LUT and the B-LUT in the successive SLICE S1 (as indicated in Column 2 of Table I). In this case, the A-FF and the B-FF in the S1 will register the value as a logic-1, whereas the values of the A-FF and the B-FF should be logic-0 in the absence of any fault. By reading the values of all of the flip-flops in the device, it would be found that the flip-flops in all of the BUTs after the SLICE S0 have incorrect values. The first BUT with incorrect values of the flip-flops (the A-FF and the B-FF in the S1) is identified, which implies the presence of a fault in the SLICE S0 logic.

The other possibility is that a stuck-at-1 fault on the A-FF or the B-FF in the SLICE S1 itself could have caused these flip-flops to have incorrect values. However, with the assumption of the single stuck-at-fault model, only one flip-flop in the S1 would read an incorrect value in this case. If multiple flip-flops in a single BUT have incorrect values, then it can be explained only by a fault propagated from its predecessor BUT. If a single flip-flop in the BUT has an incorrect value, then it implies the presence of a fault in the same BUT. The pattern created by the flip-flop values yields the required information to isolate the fault location in the ILA architecture.

VI. Summary and Results

A. *Test Coverage*

A total of 30 BIST configurations (12 for LUT testing, 13 for data-path testing, 3 for RAM testing, 2 for shift-register testing) are generated to test all of the functionalities of the CLBs in XC7Z020 FPGA. The faults in the FPGA were emulated by configuration memory bit fault injection. The intermediate files during the generation of the configuration bitstream can intentionally be corrupted to emulate a physical fault in the FPGA. For example, the content of a particular LUT location can be forced to a logic-0 to model a stuck-at-0 fault in the LUT. Similarly, the faults can be emulated in all of the logic and the memory resources in the CLB by manipulating the configuration bits associated with the particular resource. The 30 BIST configurations cumulatively detect all such faults. These 30 BIST configurations are grouped according to the CLB functionality that they cover, i.e. LUT configurations, data-path configurations, RAM configurations, and shift-register configurations. The graph in Fig. 8 shows the fault coverage in a single CLB by the different configuration groups. The left Y-axis in the graph shows the absolute number of faults covered by each configuration group, and the right Y-axis shows the fault coverage in percentage.

B. *Testing Overhead*

The primary sources of overhead associated with FPGA-BIST are the test time and the external memory storage required to store all of the BIST configurations. Fig. 9 shows that the test time is dominated by the time required to

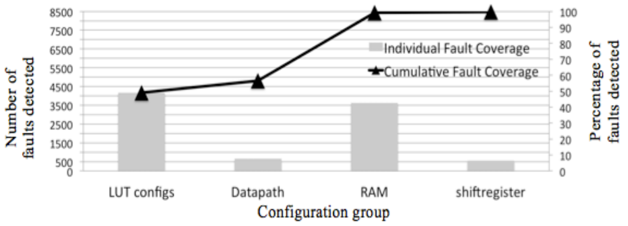


Figure 8. Fault coverage in CLB

download the BIST configuration to the FPGA, which in turn is proportional to the speed of the configuration interface and the total size of the BIST configurations. The partial reconfiguration capability of Xilinx FPGAs is used to reduce the total size of the BIST configurations. The majority of the bits in the configuration bitstream contain the routing information. Hence, if the routing is kept constant across multiple bitstreams, then the common routing information can be stored in only one bitstream. For example, the first five configurations in Table I have common connectivity between the successive BUTs implying that the routing will remain constant in all of these configurations. Out of the five configurations, only the first configuration is stored as a full bitstream, and the remaining configurations are stored as partial bitstreams. A full bitstream of Xilinx XC7Z020 device has the size of 3.4MB. If all of the five configurations were stored as the full bitstreams, then the total memory requirement would be 17 MB. However, by enabling the partial bitstream generation, the total memory requirement for the five configurations is reduced to only 5.85MB. Out of the 30 BIST configurations, only 12 configurations have unique routing, and, hence, only those configurations are needed to be stored as full bitstreams. By doing this, the average bitstream size is reduced by approximately 34%.

If all of the BUTs in the device can't be routed in a single configuration, then the FPGA has to be divided into partitions, and the partitions have to be tested one by one. As a result of this, both the number of the BIST configurations and the test time multiplies by the number of the partitions. In the BIST discussed in this paper, the connections between the BUTs are constrained to a small length; therefore, all of the BUTs in the device could be routed simultaneously in different devices of Xilinx 7-series FPGAs. As the FPGA could be tested without creating the partitions, the effective number of BIST configurations is minimized, and, therefore, both of the overheads associated with the FPGA-BIST are minimized.

VII. Conclusion

This paper presented a complete BIST to detect and isolate any stuck-at-fault present in the CLBs of Xilinx 7-series FPGAs. The test uses only the configuration interface and does not use any of the user I/O pins, and, therefore, it can be deployed on any system without making any PCB changes. The BIST architecture presented in this paper solves the routing congestion problem for the larger devices, and enables the testing of all of the CLBs in a single test session. As a result of this, the test time and the external memory required to store all of the BIST configurations is reduced significantly. The work presented here is focused on the testing of the CLBs, and applies to all of the devices within Xilinx 7-Series FPGAs. In addition to the CLBs, testing of the programmable interconnects is also essential, and is described in a companion paper [14]. In all, these tests are freely available to affiliates of the Center for High-Performance Computing (CHREC).

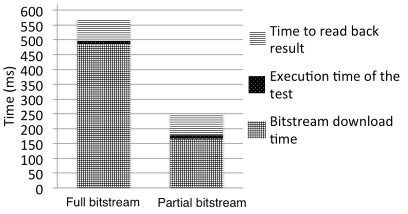


Figure 9. Breakup of the test time for XC7Z020 device

VIII. Acknowledgement

The authors would like to thank Matt French and Neil Steiner at USC-ISI for their guidance and support for this project. This work was supported in part by NSF I/UCRC IIP-1266245 via the NSF Center for High-Performance Reconfigurable Computing (CHREC).

References

[1] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of Internal vs. External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis" Proc. IEEE RADECS07, June 2008.

[2] C. Stroud, 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 Symp., pp. 387-392, April 1996

[3] C. Stroud, K. Leach, and T. Slaughter, "BIST for Xilinx 4000 and Spartan series FPGAs: a case study," Proc. IEEE Int. Test Conf., pp.1258-1267, 2003.

[4] S. Dhingra, D. Milton, and C. Stroud, "BIST for logic and memory resources in Virtex-4 FPGAs," Proc. IEEE North Atlantic Test Workshop, pp. 19-27, 2006.

[5] B. Dutton and C. Stroud, "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs," Proc, IEEE System Theory, 2009. SSST 2009.

[6] T.Sridhar and J. P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," IEEE Trans. on Computers, Vol. C-30, No. 8, pp. 563-571, August 1981.

[7] T.Sridhar and J. P. Hayes, "Design of Easily Testable Bit-Sliced Systems," IEEE Trans. on Computers,Vol. C-30, No. 11, pp. 842-854, November, 1981.

[8] C. Jordan and W. P. Marnane, "Incoming Inspection of FPGAs," Proc. European Test Conf.,April, 1993.

[9] V. K. Huang and F. Lombardi, "An Approach to Testing Programmable/Configurable Field Programmable Gate Arrays," Proc. IEEE VLSI Test Symp., pp. 450- 455, April 1996.

[10] C. S. Stroud, E. Lee, S. Konala, and M. Baranovichi, "Using ILA Testing for BIST in FPGAs," Proc. of the 1996 IEEE Interanational Test Conference

[11] 7 Series FPGAs Configurable Logic Block UG474 (v1.7), Xilinx Inc., San Jose, CA, November 2014, Available: www.xilinx.com

[12] 7 Series FPGAs Configuration UG470 (v1.9), Xilinx Inc., San Jose, CA, November 2014, Available: www.xilinx.com

[13] R. Nair, S.M. Thatte, J.A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories," Proc. IEEE Transactions on Computers, Volume:C-27 , Issue: 6, pp. 572 – 576

[14] S. Ma, P. Athanas, "In-System Testing of Xilinx 7-Series FPGAs: Part 2-Interconnectivity," in progress

[15] H.Modi, "In-system testing of CLBs in Xilinx 7-series FPGAs," M.S.thesis, Dept. Comp. Eng., Virginia Tech, Blacksburg, VA 24061, in progress