Transparent Structural Online Test for Reconfigurable Systems

Mohamed S. Abdelfattah*, Lars Bauer**, Claus Braun*, Michael E. Imhof*, Michael A. Kochte*
Hongyan Zhang**, Jörg Henkel** and Hans-Joachim Wunderlich*

*Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany
**Chair for Embedded Systems, Karlsruhe Institute of Technology, Germany

***Abstract*—FPGA-based reconfigurable systems allow the online adaptation to dynamically changing runtime requirements. However, the reliability of modern FPGAs is threatened by latent defects and aging effects. Hence, it is mandatory to ensure the reliable operation of the FPGA's reconfigurable fabric. This can be achieved by periodic or on-demand online testing.**

In this paper, a system-integrated, transparent structural online test method for runtime reconfigurable systems is proposed. The required tests are scheduled like functional workloads, and thorough optimizations of the test overhead reduce the performance impact. The proposed scheme has been implemented on a reconfigurable system. The results demonstrate that thorough testing of the reconfigurable fabric can be achieved at negligible performance impact on the application.

***Keywords*-FPGA, Reconfigurable Architectures, Online Test**

I. Introduction and System Overview

Computing systems based on reconfigurable architectures are gaining relevance. Their applications range from various embedded systems to high-performance computers and large-scale research systems. Often, such systems are implemented using *Field Programmable Gate Arrays (FPGAs)*. The functionality of FPGAs can be changed by configuration data, allowing the implementation of different designs at different points in time. Moreover, latest generation FPGAs support the concept of *partial runtime reconfiguration*, i.e. selected regions of the fabric can be reconfigured during runtime without affecting other regions. This concept provides a very high degree of flexibility that can be used to adapt such runtime reconfigurable systems to different requirements.

Some architectures use their reconfigurable fabric to implement application-specific accelerators that are integrated into a processor core as so-called Special Instructions [1] or that are attached to a processor as a co-processor [2]. Other architectures implement entire application tasks as hardware blocks and reconfigure them to the fabric on demand [3]. For practical reasons, basically all FPGA-based implementations partition the reconfigurable fabric statically into so-called *containers* and the accelerators/tasks are configured into these containers.

Manufactured in latest semiconductor process technologies (e.g. Xilinx Virtex-7 in 28nm), FPGAs are increasingly prone to aging effects and latent defects in the fabric [4]. Classic production and burn-in tests are no longer sufficient to guarantee reliable reconfigurable systems throughout the lifetime. For such systems it is mandatory to ensure that the reconfiguration process itself is reliable. This requires thorough testing of the part of the FPGA to be reconfigured (i.e. the container). Such tests are challenging, because they have to be executed online with very limited resources and they should cause only minimal overhead. The diversity of FPGA architectures and the limited availability of documentation make these tests a non-trivial task. Such a reliable reconfiguration process is also required for approaches, which utilize reconfiguration to achieve reliable system operation [5–7].

In this work, a novel system scheme for the online *pre-configuration test (PRET)* of FPGA structures is introduced. The scheme is non-intrusive and allows all required *test configurations* to be scheduled independent of each other prior to any user or accelerator configuration. Containers are configured either with application-specific accelerators or with PRET test configurations. PRET is extensible and allows the seamless integration of additional test configurations for specialized functional blocks or future FPGA structures. Therefore, the proposed PRET method is independent of the reconfigurable system into which it is integrated. However, to illustrate the constraints and desired properties for designing test methods for reconfigurable systems, we explain its integration into a typical system architecture that is also used for evaluation in section V (more details in the experimental setup). This system consists of a core pipeline and a set of reconfigurable containers that are connected to it (see fig. 1). Special inter-container buses are used to connect the containers to each other, to the processor pipeline (access to register file), and the data memory hierarchy (access to cache/scratchpad).

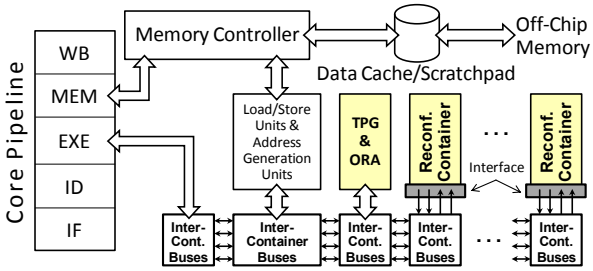

Fig. 1: System architecture with integrated test pattern generator (TPG), output response analyzer (ORA), and reconfigurable containers [8]

The system architecture is extended by the *test pattern generator (TPG)* and the *output response analyzer (ORA)* to test the containers, as shown in fig. 1. The inter-container buses are used to connect TPG/ORA to the containers. Therefore, the communication between TPG/ORA and a particular container-under-test is limited by the I/O interface of the containers (in this system two 32-bit inputs and outputs, respectively) and it has to operate at system frequency. Whenever a test configuration is configured into a container, no other configuration can be performed, i.e. reconfigurations requested by the applications are delayed. In addition to complying with container size and I/O constraints, these delays should be minimized. The runtime system, which controls the reconfigurations, is extended to schedule test configurations in regular intervals.

The remainder of this paper is structured as follows: section II gives a concise introduction to structural FPGA testing. A detailed description of the online pre-configuration test method is given in section III. Implementation details and experimental results are presented in sections IV and V. The paper is concluded in section VI.

II. State of the Art FPGA Test

The thorough test of the FPGA fabric requires structural knowledge. The test consists of the application of multiple test configurations (TCs). Such a TC configures the FPGA in a way that a part of the structures or the function of the structures is controllable and observable so that appropriate test stimuli can be applied to test for faults. The number of required TCs may range from a few up to a few hundreds if programmable interconnect structures are completely included [9].

Different test strategies are used for the logic and sequential parts of the Configurable Logic Blocks (CLBs), interconnects, I/O cells and specialized cores like RAM or DSPs. For memories, functional March tests [10] are used, for arithmetic structures like multipliers or DSPs, functional tests with high structural fault coverage are possible [11].

The test cost of FPGAs is dominated by the test time and test data volume. The test time mainly depends on the number of TCs, whereas the test data comprises the data volume for the configuration bits and test stimuli. If the mission logic is fixed and known, an application dependent test of the FPGA is possible [12]. However, this work focuses on dynamically reconfigurable architectures and hence requires application independent testing.

The stuck-at fault model is most commonly used for the test of logic resources. For testing interconnect, stuck-on and stuck-open faults in programmable interconnect points and stuck-at and bridge faults for wires [13, 14] are targeted. Tests for delay faults have been introduced in [15].

For an online in-field test of FPGAs, external equipment for test pattern generation (TPG) and output response analysis (ORA) [16, 17] is not available. Internal testing approaches based on built-in self-test principles include TPGs and ORAs in the unit under test. TPGs can be efficiently implemented by counters or linear feedback shift registers to exhaustively generate all stimuli for a module under test. ORAs can be implemented by mutually comparing corresponding outputs of similarly configured modules under test [18].

The highly regular nature of FPGAs allows to configure the structures under test into an iterative logic array such that the resulting test time is constant and does not depend on the array size (C-testability [19]). FPGAs with support for memory and partial memory readback allow a test strategy similar to scan design where the response is captured in sequential elements, read back, and finally analyzed [20, 21]. The readback increases the time for test.

While the programmable interconnect structures are to some extent already tested during the test of other structures, dedicated deterministic testing based on multiple test configurations has been proposed [14, 22]. Due to the complexity of the interconnect configuration circuitry, a very high number of TCs is required. In addition to testing, the homogeneous structure of an FPGA allows efficient diagnosis of faulty components. High resolution is achieved by failure data analysis and additional dedicated TCs to distinguish faults with equal signatures [23].

Using partial dynamic reconfiguration of FPGAs, the test reconfiguration can be conducted by an external [24] or embedded processor during runtime [25, 26]. Access to an internal high-bandwidth configuration port significantly reduces the time required for reconfiguration.

Both test and diagnosis can be executed offline, requiring idle periods of the unit under test, or online, allowing the parts of the array which are currently not under test to continue functional operations. The Roving STARs (self testing areas) method [27] partitions the FPGA into multiple regions, which can be either used functionally or tested by an online built-in self-test scheme controlled by an external module. By relocation of mission logic from yet untested regions to tested regions and test iterations, the whole array can be tested for faults.

The described online approaches require an external or internal reconfiguration controller (possibly embedded as soft core). In contrast, this work avoids this overhead by transparently integrating the reconfiguration process for PRET into the functional system scheduling. The existing reconfiguration mechanism in the system is reused for PRET.

III. Online Pre-Configuration Test Method

This section details our proposed test procedure for FPGA fabrics. After explaining the fault model, the test approach and characteristics are outlined. An in-depth explanation of each combinational logic block (CLB) subcomponent test and a global optimization method are given. Although dedicated configurations for the test of interconnects exist, they are not described here for the sake of conciseness. In this context, [28] reports that up to 80% of the global interconnects are implicitly tested by the CLB test configurations.

A. Fault Model

This work uses the stuck-at fault model for CLB subcomponents and local interconnects in which the available structural information is sufficient for fault derivation. For the remaining components, functional and cell faults are targeted during test generation resulting in a hybrid fault model. The tests are generated under the single fault assumption.

1) Lookup table in function mode: The lookup table (LUT) in function mode is treated as a combinational function of n inputs and m outputs, and the Cell Fault Model [29] is applied. The Cell Fault Model makes no assumptions on the structure of the unit under test. A cell fault is defined as any mismatch at the output of a unit under test for the possible inputs. As a result, the set of all cell faults covers all deviations from the correct combinational behavior. The number of cell faults equals the number of possible input patterns multiplied by the number of possible faulty output patterns $2^m(2^n - 1)$.

2) Lookup table in RAM mode: If the LUT is operated in RAM mode, the functional target faults are derived from the domain of memory testing and include address decoder faults, stuck-at faults, transition faults and data retention faults.

3) Sequential elements: CLBs in an FPGA may contain additional separate sequential elements such as flip-flops, latches, or LUTs in shift register mode. For these elements, the considered faults are stuck-at and transition faults (slow-to-rise, slow-to-fall). The four faults per flip-flop dominate stuck-at faults on the interconnection between flip-flops.

B. CLB Test Principle and Architecture

The CLB test approach presented in this and the following section is applicable to different types of FPGAs since the internal CLB structure is typically similar. The approach is applied to the Virtex-5 target architecture as described in section V-A.

The complexity of a CLB requires multiple TCs to exercise all components. Full coverage of faults inside the logic is ensured by deterministic design of the TCs and of the test stimuli applied to exercise the configured components. As shown in fig. 2, the CLBs are connected in a C-testable array for low test time. The TPG and ORA may differ between TCs. They are external to the container under test as shown in fig. 1. The TPG provides input stimuli to the container under test and the ORA receives its response via the inter-container buses, respectively.

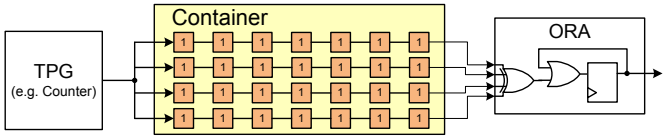

Fig. 2: Container configured into a C-testable array with external test pattern generator and output response analysis (connected via inter-container buses)

To avoid a long critical path and its dependence on the container size, the tested subcomponents are pipelined to test at system speed. Logic tests are pipelined by utilizing the sequential elements included in each CLB. To test for all faults, the unregistered outputs of CLBs must also be tested. An interleaving array scheme [28] as shown in fig. 3 is used.

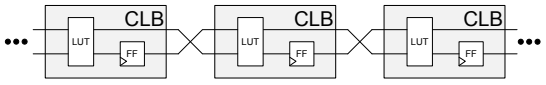

Fig. 3: CLB in fully interleaved/pipelined array configuration

C. Test Methods for CLB Subcomponents

1) Lookup Table - Function mode: All cell faults are targeted to cover all single and all multiple internal combinational faults. The exhaustive set of test patterns is applied to the inputs (2^n test patterns for an n-input LUT).

A LUT is configured with two complementary functions in order to test for all stuck-at faults in the configuration bits. It would suffice to use the tautology and its inverse (all 1's and all 0's) for testing a single LUT. However, XOR/XNOR configurations are used instead since they can be connected into C-testable arrays [30].

2) Lookup Table - Shift Register Mode: In shift register mode, the flip-flops are configured into a long shift register. In addition to stuck-at faults, transition faults are tested for by applying standard scan chain test patterns [31]. The "01100" test pattern is used because it contains the two transitions between 0 and 1.

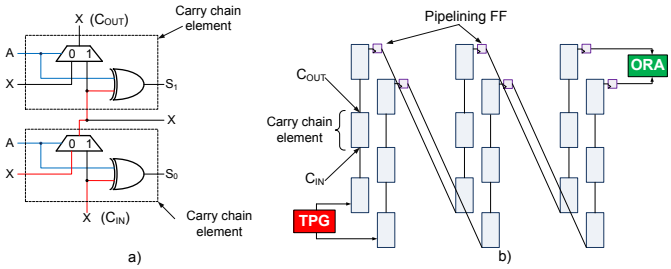

Fig. 4: Carry chain test architecture: (a) Two-stage chain under test (b) Pipelined test setup

The LUTs in shift register mode are connected into multiple scan chains of which the outputs are compared against one another for response evaluation. To minimize test configurations, separate flip-flops in each CLB can be simultaneously tested in the same TC by including them into the chain.

3) Lookup Table - RAM mode: Each n-input LUT can implement a 2^n-bit RAM. Test patterns are generated at a global TPG implementing the MATS++ [10] algorithm to ensure coverage of all stuck-at faults, address decoder and transition faults. Note that only 5×2^n March operations are required since the initialization step is specified directly in the TC.

Each CLB contains multiple RAM modules. Test response analysis is done by comparing the output of these RAM blocks and aggregating the results into the global ORA, which will detect multiple errors as long as there is at most one error per CLB.

4) Multiplexer: Multiplexers are tested by applying all possible configurations to exercise all select combinations. The data path is tested for stuck-at faults by applying the 0 and 1 stimuli. Multiplexer testing is often included in other tests since they are on the sensitized path used for testing other subcomponents.

5) Fast Carry Chain: Many FPGA architectures contain dedicated carry chains. An example is given in fig. 4 (a), consisting of multiplexers and XOR cells. To test for all the stuck-at faults in the carry chain efficiently, the elements must also be configured into pipelined C-testable arrays. Two TCs are required to fully test the carry chain.

The first TC shown in fig. 4 (a) tests for faults in the XOR gates and a subset of the faults in the multiplexers. The inputs (X) are driven with identical values and propagated through the multiplexer to the XOR gate. In this scenario the multiplexers are transparent and the carry-chain blocks are configured into XOR arrays. The test signals are generated in the LUTs and the outputs S_0, S_1 are compared and pipelined using the CLB flip-flops.

In the second TC shown in fig. 4 (b), the *carry-in* and *carry-out* pins are tested by connecting the chain elements in a long carry chain and propagating the 0 and 1 values through it to test for both stuck-at faults. A flip-flop is used at the end of each column to pipeline the test.

6) Latches and Flip-Flops: Flip-flop testing is identical to testing the LUT in shift register mode. If sequential elements can be configured as level sensitive latches, a separate test is required to guarantee proper latch function. To test for the correct function of the latches, including all stuck-at faults and

transition faults, two non-overlapping clocks are used as input to the scan chain. The same test pattern used for the flip-flops (01100) is also used for testing the latches [31].

D. CLB Test Configuration Minimization

Since some CLB subcomponents can be tested in parallel, the global number of TCs can be reduced. The method of [32] maps the minimization of TCs to a set covering problem. Three testability conditions must hold for each tested subcomponent: (1) All required TCs of the submodule are applied; (2) All inputs of each submodule must be controllable such that all input combinations can be applied for the test; (3) All outputs of each submodule must be observable to evaluate the results of all test stimuli.

If all of the testability conditions of each subcomponent are satisfied using a set of TCs, this set is sufficient to perform a full structural test of the CLB, and all other TCs are redundant. The lower bound of the number of global TCs is given by the maximum number of TCs per submodule. For the CLB of the used architecture (Virtex-5, c.f. section V-A), the lower bound is 5 TCs as determined by the largest multiplexer in the CLB.

In this work, the search for a covering set of global TCs is implemented using a randomized greedy heuristic with polynomial runtime complexity. However, the results cannot be applied directly since they do not necessarily satisfy two additional constraints imposed on all TCs: Interconnection in C-testable arrays and pipelining. Including these additional constraints would require to significantly enlarge the set covering problem instance. Therefore, the results are used as a starting point for manual optimization. A global TC that cannot be pipelined and connected in a C-testable array is divided into two TCs until all TCs satisfy these conditions.

An example is a global TC which requires all outputs of the CLB to be observable for fault effects, but which cannot generate at its outputs all the test stimuli for the inputs of the next downstream CLB. By dividing this TC, the targeted CLB subcomponents are distributed and the freedom to establish C-testable and pipelined configurations increases. For the considered CLB, the number of global TCs increases from 5 to 9 to satisfy the C-testability and pipelining constraints.

IV. Implementation

For testing the reconfigurable fabric of a Xilinx Virtex-5 FPGA, a series of test configurations (TCs) is first generated from test configuration templates which are obtained as described in the previous section. This is implemented in Java based on the RapidSmith Java framework [33]. The output is a set of XDL (Xilinx Design Language) files which are used to generate bitstreams for the TCs (see fig. 5). First, the container size is selected by specifying the CLB coordinates at the lower-left and upper-right coordinates of a rectangular container on the FPGA. The PRET tool then generates the TPG and ORA for the TCs targeting the CLB subcomponents. Because of the C-testability property, all container sizes require the same number of test patterns. The container setup is then created by instantiating the test configurations for all targeted CLBs. At this point, the input and output connections are defined at the boundary of the C-testable CLB array. While the connections

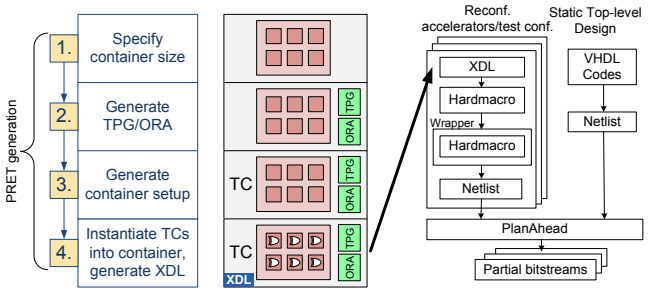

Fig. 5: XDL file generation flow for CLB TCs; Partial reconfiguration flow for test configurations

within a CLB are defined by the test configuration XDL file, the routing within the container between CLBs is done later using the Xilinx place & route tool.

In order to integrate the TCs into the runtime system of the reconfigurable architecture, each TC must be packed into a partial bitstream and stored in memory. The runtime system of the reconfigurable architecture fetches the partial bitstream of the TC from memory and transfers it over the *internal configuration access port* into the FPGA so that the corresponding container is configured as a test array and ready for being tested.

The standard partial reconfiguration flow generates the partial bitstreams and uses PlanAhead as the tool to synthesize and place & route the design. The input source files for PlanAhead are netlists synthesized from VHDL before technology mapping. As the TCs are mapped and placed designs describing the low-level configuration details of the hardware resources to be tested, they cannot be used directly as input to PlanAhead. They require additional handling for the integration into the partial reconfiguration flow (see fig. 5): Each TC is converted from its XDL description to a pre-mapped and placed hardmacro and then wrapped in a VHDL entity. This wrapper with the included hardmacro is then synthesized to a netlist that is used as the input source file of the reconfigurable accelerator. Together with the netlist of the static top-level design (core pipeline, TPG, ORA and the runtime system) it is provided to PlanAhead, which generates the partial bitstream for the TC.

Since TCs and containers share the same communication infrastructure (see fig. 1), TCs must have the same interface to the inter-container buses as the containers have. In the tool-flow, the mapping from CLB pins at the boundary of the C-testable array in the hardmacro to symbolic identifiers in VHDL is achieved by a wrapper. Although different TCs may differ in their connection to the TPG and ORA, these complexities are not exposed to the communication infrastructure but masked by the wrapper so that TCs and containers have the same I/O interface specification.

The runtime system establishes the communication between the TC and the TPG/ORA after the TC is reconfigured to a container. The test only lasts a short time (see section V). During that time, Special Instructions cannot be executed in hardware as the inter-container buses are used by the test. The runtime system either stalls the CPU or triggers an emulation of the Special Instructions in software. This software emulation (realized by an unimplemented-instruction

trap) is also used when a Special Instruction shall execute but the reconfiguration of the accelerator is not completed yet.

V. Evaluation and Results

A. Experimental Setup

A Leon-3 processor is used as core pipeline with 10 attached containers, where each container consists of 10x40 CLBs. The actual hardware prototyping is performed on an XUPV5 FPGA board with a Xilinx Virtex-5 LX110. A SystemC-based simulator is used for evaluating different system parameters like the number of containers. The system operates at a clock frequency of 100 MHz and has a reconfiguration bandwidth of 66 MB/s. Although the internal configuration access port of the Xilinx FPGA can operate at up to 400 MB/s, the actual configuration bandwidth is limited by off-chip DRAM speed.

To analyze the PRET overhead, a sophisticated H.264 video encoder is chosen as target application. The encoder is a representative application since it uses many different accelerators (e.g. transformations, filters, SADs, arithmetic etc.) and it frequently reconfigures these accelerators in the containers. In detail, the H.264 encoder consists of three different functional blocks that are executed in sequence for each video frame: motion estimation, encoding, and in-loop deblocking filtering. Each of these functional blocks requires different Special Instructions, hence the system reconfigures the containers accordingly. In total, 9 Special Instructions are implemented for the encoder by using combinations of 10 different accelerators [34]. The implementation of the H.264 encoder on the reconfigurable system leads to a speedup of more than 20x compared to the same system without using Special Instructions.

B. Test Configurations

Table I gives an overview on the nine generated test configurations (TCs). Column one shows the configuration number. Column two shortly describes the portion of each CLB being tested. Columns three and five list the PRET overhead in CLBs used for the TPG and ORA and the maximum achievable test frequency. The nine test configurations achieve complete coverage of the targeted faults. The total hardware overhead introduced by PRET is 17 CLBs.

The test time for a container consists of two parts: the container configuration time and the test application time. The latter is the sum of the number of pattern (c.f. table I) plus the sequential depth of the pipelined TC. The configuration dominates the test time with tens of thousands of cycles. The configuration time is directly proportional to the size of the configuration data (bitstreams) and the reconfiguration bandwidth. Column four shows the sizes of the partial bitstreams for the generated 9 TCs. Based on the configuration bandwidth of 66 MB/s, configuration of a TC requires between 1.49 ms and 1.99 ms.

While TCs 3 and 4 seem to be identical, two configurations are needed to satisfy the C-testability and pipelining constraints. The same holds for TCs 5 and 6. Each TC has a common test interface consisting of one system clock, two non-overlapping clocks, one 8-bit input from the TPG and one 8-bit output to the ORA. The I/O interface of the containers with 1 system clock and two 32-bit inputs and outputs is extended with 2 non-overlapping clocks so that it becomes a superset of the signals needed by the TCs.

TC	Tested CLB subcomponents	PRET overh. [CLBs]	Bitstr. size [KB]	Freq. [MHz]	Num. patterns
1	LUT as XOR, via FF	2	108	207	64
2	LUT as XNOR, via FF	2	108	207	64
3	Carry MUX, via latch	1	129	168	6
4	Carry MUX, via latch	1	122	154	6
5	Carry XOR, via FF	1	129	168	6
6	Carry XOR, via FF	1	131	154	6
7	Carry-I/O multiplexed	1	131	183	6
8	LUT as Shift Reg. with slice MUX	1	110	157	6
9	LUT as RAM with slice output	7	98.4	225	320

TABLE I: Characteristics of CLB test configurations

C. Concurrent Test Scheduling

Tests are regularly scheduled to containers like functional workloads, when these containers are currently not used and the configuration port is available. A simple test strategy is applied here, which schedules one test configuration after a certain number of accelerator configurations (ACs) to demonstrate that the test has negligible effect on the performance while still maintaining high test effectiveness. Here, one TC is scheduled before each AC (or before every 2nd, 4th, 8th, … AC). This corresponds to one AC per TC (or 2 ACs per TC etc.) Fig. 6 shows the simulation results for the performance loss under different test frequencies from 1 AC/TC to 64 ACs/TC for three different numbers of containers. The performance loss is mainly caused by the delay of the accelerator configurations due to testing. Depending on the test frequency, the time required to iterate through all nine TCs per container differs. For the investigated cases, table II lists the test completion time in seconds for different test frequencies. For example, in the case of 12 containers and 32 ACs/TC, the test completion time is approx. 59s. This is sufficient when targeting latent and permanent faults which may emerge for example due to aging. With this test frequency, the performance loss is only 0.16% which is negligible for the system performance.

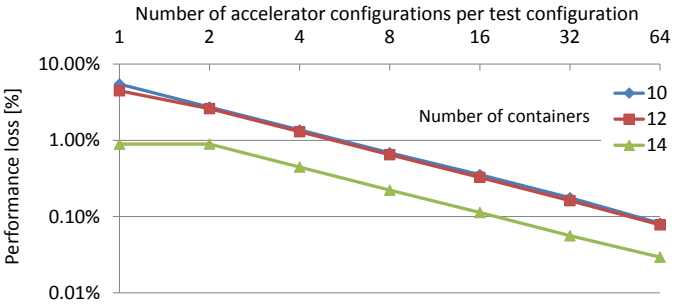

Fig. 6: Performance loss under different test frequencies

#Cont.	Number of accelerator conf. per TC						
	1	2	4	8	16	32	64
	Test completion time [seconds]						
10	1.4	2.8	5.6	11.2	22.6	45.0	89.8
12	1.8	3.7	7.4	14.8	29.6	59.1	118
14	3.3	6.9	13.9	28.2	56.3	112	225

TABLE II: Test completion time (i.e. test latency)

VI. Conclusion

This work presents a novel pre-configuration test (PRET) scheme for runtime reconfigurable architectures. Parts of the reconfigurable fabric, called containers, are thoroughly tested at runtime by scheduled test sessions. The impact of testing on the system operation is minimized. The structural test method is implemented and integrated into an existing reconfigurable system. For this reconfigurable architecture with 12 containers and a complex H.264 video encoding application, the test overhead in terms of performance loss can be reduced to less than 0.2% while providing a very fast test completion time of only 59s. The proposed PRET approach requires only minimal adaption at system level and introduces a negligible hardware overhead of 17 CLBs.

Testing is the very first step towards dependable reconfigurable architectures. Once faults in containers are detected using PRET, countermeasures can be taken to recover from resulting errors, to quarantine defective resources, and to reconfigure the system for graceful degradation.

References

[1] J. E. Carrillo and P. Chow, "The effect of reconfigurable units in superscalar processors", in *Int'l Symposium on Field Programmable Gate Arrays (FPGA)*, 2001, pp. 141–150.

[2] S. Vassiliadis *et al.*, "The MOLEN polymorphic processor", *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1363–1375, 2004.

[3] E. Lübbers and M. Platzner, "ReconOS: Multithreaded programming for reconfigurable computers", *ACM Trans. on Embedded Computing Systems*, vol. 9, pp. 8:1–8:33, 2009.

[4] N. Metha and A. DeHon, "Variation and aging tolerance in FPGAs", in *Low-Power Variation-Tolerant Design in Nanometer Silicon*. Springer Science+Business Media, 2011.

[5] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Enhanced FPGA reliability through efficient run-time fault reconfiguration", *IEEE Trans. Reliability*, vol. 49, no. 3, pp. 296–304, 2000.

[6] S. Mitra *et al.*, "Reconfigurable architecture for autonomous self-repair", *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 228–240, 2004.

[7] C. Bolchini *et al.*, "A Reliable Reconfiguration Controller for Fault-Tolerant Embedded Systems on Multi-FPGA Platforms", in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2010, pp. 191–199.

[8] L. Bauer *et al.*, "OTERA: Online test strategies for reliable reconfigurable architectures", in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, June 2012.

[9] C. Stroud, "Ch. 12.4 field programmable gate array testing", in *VLSI Test Principles and Architectures*, L. Wang, C. Wu, and X. Wen, Eds. Morgan Kaufmann, 2006.

[10] M. Renovell *et al.*, "SRAM-Based FPGAs: Testing the Embedded RAM Modules", *Journal of Electronic Testing (JETTA)*, vol. 14, pp. 159–167, 1999.

[11] K. Radecka, J. Rajski, and J. Tyszser, "Arithmetic Built-In Self-Test for DSP Cores", *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1358–1369, 1997.

[12] M. Tahoori, "Application-Dependent Testing of FPGAs", *IEEE Trans. Very Large Scale Integration Systems*, vol. 14, no. 9, pp. 1024–1033, 2006.

[13] C. Stroud *et al.*, "Built-in self-test of FPGA interconnect", in *Proc. International Test Conference*, 1998, pp. 404–411.

[14] M. Tahoori and S. Mitra, "Application-independent testing of FPGA interconnects", *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1774–1783, 2005.

[15] E. Chmelar, "FPGA interconnect delay fault testing", in *Proc. International Test Conference*, 2003, pp. 1239–1247.

[16] W. K. Huang *et al.*, "Testing configurable LUT-based FPGA's", *IEEE Trans. VLSI*, vol. 6, pp. 276–283, 1998.

[17] M. Renovell *et al.*, "Testing the interconnect of RAM-based FPGAs", *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 45–50, 1998.

[18] M. Abramovici and C. Stroud, "BIST-based test and diagnosis of FPGA logic blocks", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9, no. 1, pp. 159–172, 2001.

[19] M. Renovell *et al.*, "Test pattern and test configuration generation methodology for the logic of RAM-based FPGA", in *Proc. Asian Test Symposium*, 1997, pp. 254–259.

[20] C. Stroud *et al.*, "Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)", in *Proc. VLSI Test Symposium*, 1996, pp. 387–392.

[21] P. Sundararajan, S. Mcmillan, and S. A. Guccione, "Testing FPGA devices using JBits", in *Military and Aerospace Applications of Programmable Devices and Techn.*, 2001.

[22] C. Stroud *et al.*, "Built-in self-test of FPGA interconnect", in *Proc. International Test Conference*, 1998, pp. 404–411.

[23] M. Abramovici, C. E. Stroud, and J. M. Emmert, "Online BIST and BIST-Based Diagnosis of FPGA Logic Blocks", *IEEE Trans. VLSI*, vol. 12, no. 12, pp. 1284–1294, 2004.

[24] V. Verma, S. Dutt, and V. Suthar, "Efficient On-line Testing of FPGAs with Provable Diagnosabilities", in *Design Automation Conference (DAC)*, 2004, pp. 498–503.

[25] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks", *IEEE Trans. VLSI*, vol. 15, no. 2, pp. 216–226, 2007.

[26] B. F. Dutton and C. E. Stroud, "Soft core embedded processor based built-in self-test of FPGAs", in *Int'l Symp. on Defect and Fault-Tolerance in VLSI Systems*, 2009, pp. 29–37.

[27] M. Abramovici *et al.*, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications", in *Proc. International Test Conference*, 1999, pp. 973–982.

[28] C. Metra *et al.*, "Novel technique for testing FPGAs", in *Design, Autom. and Test in Europe*, 1998, pp. 89–94.

[29] M. Psarakis, D. Gizopoulos, and A. Paschalis, "Test Generation and Fault Simulation for Cell Fault Model using Stuck-at Fault Model based Test Tools", *Journal of Electronic Testing (JETTA)*, vol. 13, pp. 315–319, 1998.

[30] M. Renovell, "SRAM-based FPGAs: a structural test approach", in *Brazilian Symp. Integrated Circuit Design*, 1998, pp. 67–72.

[31] S. Makar and E. McCluskey, "Functional tests for scan chain latches", in *Proc. Int'l Test Conference*, 1995, pp. 606–615.

[32] M. Renovell *et al.*, "Test configuration minimization for the logic cells of SRAM-based FPGAs: a case study", in *Europ. Test Workshop*, 1999, pp. 146–151.

[33] C. Lavin *et al.*, "Rapid prototyping tools for FPGA designs: RapidSmith", in *International Conference on Field-Programmable Technology (FPT)*, 2010, pp. 353–356.

[34] M. Shafique, L. Bauer, and J. Henkel, "Optimizing the H.264/AVC video encoder application structure for reconfigurable and application-specific platforms", *Journal of Signal Processing Systems (JSPS)*, vol. 60, no. 2, pp. 183–210, 2009.