Test Strategies for Reliable Runtime Reconfigurable Architectures

Lars Bauer, *Member, IEEE*, Claus Braun, *Member, IEEE*,
Michael E. Imhof, *Student Member, IEEE*, Michael A. Kochte, *Student Member, IEEE*,
Eric Schneider, Hongyan Zhang, *Student Member, IEEE*,
Jörg Henkel, *Senior Member, IEEE*, and Hans-Joachim Wunderlich, *Fellow, IEEE*

Abstract—FPGA-based reconfigurable systems allow the online adaptation to dynamically changing runtime requirements. The reliability of FPGAs, being manufactured in latest technologies, is threatened by soft errors, as well as aging effects and latent defects. To ensure reliable reconfiguration, it is mandatory to guarantee the correct operation of the reconfigurable fabric. This can be achieved by periodic or on-demand online testing. This paper presents a reliable system architecture for runtime-reconfigurable systems, which integrates two non-concurrent online test strategies: *Pre-configuration online tests* (PRET) and *post-configuration online tests* (PORT). The PRET checks that the reconfigurable hardware is free of faults by periodic or on-demand tests. The PORT has two objectives: It tests reconfigured hardware units after reconfiguration to check that the configuration process completed correctly and it validates the expected functionality. During operation, PORT is used to periodically check the reconfigured hardware units for malfunctions in the programmable logic. Altogether, this paper presents PRET, PORT, and the system integration of such test schemes into a runtime-reconfigurable system, including the resource management and test scheduling. Experimental results show that the integration of online testing in reconfigurable systems incurs only minimum impact on performance while delivering high fault coverage and low test latency.

Index Terms—FPGA, Reconfigurable Architectures, Online Test

1 Introduction

Reconfigurable architectures are constantly gaining significance for a broad spectrum of applications. In particular, systems based on *Field-Programmable Gate Arrays* (FPGAs) can be found from high-performance computing [1] and large research systems [2], down to a plethora of sophisticated embedded applications [3]. *Partial runtime reconfiguration* is one of the key innovations that have been introduced in modern FPGAs. It allows the reconfiguration of selected parts of the FPGA's fabric at runtime without affecting other regions that are currently in use. Hence, such runtime-reconfigurable systems provide an impressive degree of flexibility and they allow designers to find the optimal balance between computational performance and power consumption for their applications at runtime [4].

Modern FPGAs are typically manufactured in latest semiconductor process technologies (e.g. 28 nm for Xilinx Virtex-7 and Altera Stratix V). They must not only cope with soft errors to a growing extent, but also with aging effects, variations, and latent defects in the reconfigurable fabric [5–7]. These reliability threats are aggravated by long mission times and harsh environmental conditions (e.g. temperature or radiation).

Test is the enabling technology that allows reliable and economic use of most recent semiconductor devices. Classic production and burn-in tests are no longer sufficient to guarantee reliable reconfigurable systems throughout the whole product lifecycle. For instance, the reconfigurable fabric of an FPGA has to be constantly monitored by thorough online testing to check its correct operation over time. In contrast to offline testing, online tests can be performed concurrently or non-concurrently to the system's operation. This task is particularly challenging for runtime-reconfigurable systems where the hardware configuration changes dynamically as part of the normal operation.

For the proposed non-concurrent online *pre-configuration online tests* (PRET) and *post-configuration online tests* (PORT), we adopt concepts of structural logic and interconnect tests. The PRET and PORT differ in their target fault models, test intervals, and test application times. PRET is designed to test the hardware structure of regions within the reconfigurable fabric periodically or on-demand [8, 9]. It consists of tests for all major FPGA structures, such as *Configurable Logic Blocks* (CLBs) and interconnects. PORT has the objective to perform tests on *accelerators* which are hardware modules that are dynamically instantiated by runtime reconfiguration. After reconfiguration, PORT checks that they have been configured correctly and deliver the expected functionality. At mission time, PORT is used to periodically check the accelerators for malfunctions in the programmable logic (e.g. caused by soft errors).

The test application interrupts the system operation only

- *L. Bauer, H. Zhang and J. Henkel are with the Chair for Embedded Systems, Department of Computer Science, Karlsruhe Institute of Technology, Building 07.21, Room 316.2, Haid-und-Neu-Str. 7, D-76131 Karlsruhe, Germany. E-mail: {lars.bauer, hongyan.zhang, henkel}@kit.edu*
- *C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider and H.-J. Wunderlich are with the Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart, Germany. E-mail: {braun, imhof, kochte, eric.schneider}@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de*

Manuscript received 15 Aug. 2012; revised 15 Jan. 2013; accepted 24 Feb. 2013; published online 6 Mar. 2013.
Recommended for acceptance by K. Benkrid, D. Keymeulen, U.D. Patel, and D. Merodio-Codinachs.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2012-08-0565.
Digital Object Identifier no. 10.1109/TC.2013.53.

for a minimal amount of time in the order of a few microseconds. Both test schemes are transparent with respect to the application and accelerator design, i.e. they don't have to be modified. The direct system integration of such test schemes into a runtime-reconfigurable system with minimum impact on application and system performance, combined with a high fault coverage is a challenging task that is addressed in this paper.

The main contributions of this work are as follows:

- Introducing a reliable system architecture for runtime-reconfigurable systems.
- Seamless integration of nonconcurrent PRET and non-concurrent PORT such that they are transparent to applications and users.
- Scheduling of PRET (including the reconfiguration of special test configurations) and PORT periodically and on-demand.
- Thorough evaluation of the introduced runtime-reconfigurable system with respect to performance and fault coverage.

The paper is structured as follows: Section 2 gives an overview of the related work in the fields of reconfigurable architectures, dependable systems, and test. Section 3 explains the proposed PRET and PORT methods. The system integration of the proposed online test schemes and the required scheduling is described in Section 4, followed by an in-depth evaluation in Section 5. Section 6 concludes the paper.

2 Related Work

2.1 Reconfigurable Architectures

Different kinds of reconfigurable architectures evolved in the last years [10, 11]. Most architectures focus on exploiting the potential of runtime reconfiguration to increase the performance of applications. Some architectures reconfigure entire tasks as dedicated hardware implementations [12], whereas other architectures reconfigure application-specific accelerators that are invoked by an application executed on a processor [13–15].

Fig. 1 shows the system architecture of a typical reconfigurable processor that is composed of runtime-reconfigurable regions, so-called *containers*, and a non-reconfigurable core pipeline. The containers, the core pipeline and the system memory hierarchy are connected by an interconnect infrastructure as in Fig. 1 which consists of four bi-directional segmented buses as used in [14]. [14] is also used as baseline architecture for later integration and evaluation.

The core pipeline is extended by *Special Instructions* (SIs), i.e. assembler instructions that perform application-specific computations such as transformations, filters, encryption. One or multiple accelerators need to be reconfigured to containers to implement an SI in hardware. A runtime system decides, which accelerators are reconfigured into which containers as well as the reconfiguration sequence. More details about the baseline architecture and its runtime system are available in [14].

Despite the advantages of runtime-reconfigurable architectures, their management is challenging, especially when it comes to testing. The regions of the reconfigurable fabric

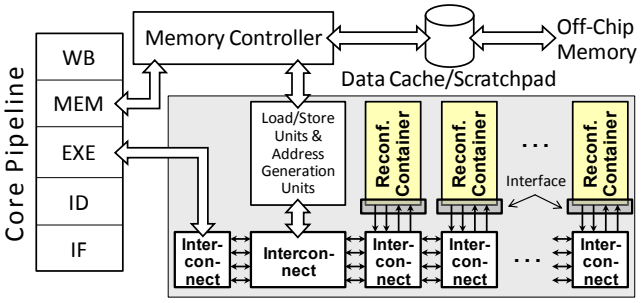

Fig. 1. Overview of a typical reconfigurable processor architecture consisting of reconfigurable containers, the core pipeline, and the memory hierarchy.

that are not intended for runtime reconfiguration can be tested for permanent faults with established approaches such as software-based testing of processors and caches [16], or periodic FPGA test methods which require the interruption of operation [17]. To test for configuration memory bit-flips, periodic scrubbing of configuration memory [18] is very effective.

The focus of this paper lies in regions that are reconfigured at runtime. They need a different testing approach as their configurations are not known statically. Even though the partial bitstreams to configure a container are prepared statically and then retrieved from a repository, the decisions 'which partial bitstream' should be reconfigured into 'which container' at 'which time' are determined at runtime (example given in Section 5.1).

2.2 Dependable Systems

Dependability, among other system properties, comprises both reliability and availability [19]. The focus of this work is to increase reliability by ensuring that the used reconfigurable fabric is fault-free and the reconfiguration process completes without error.

For the test of logic resources of the fabric, the stuck-at fault model is most commonly used. Dedicated fault models exist for interconnects [20]. Test generation for delay faults in the fabric has been introduced in [21]. Apart from these permanent faults, transient events can cause data corruption in memory elements. Especially for SRAM-based FPGAs, soft errors in the configuration memory can alter the configured function. Faults in the reconfiguration logic, e.g. address decoder faults, can result in arbitrary behavior of the configured fabric.

The reliability of a system can be increased by tests targeting faults, and fault-tolerance measures such as time, information, or structural redundancy for concurrent error detection. If executed autonomously and online, these techniques allow a system to detect latent faults and correct errors. Autonomous adaptation to faults as well as graceful degradation becomes possible.

Classical approaches, such as duplication with comparison, triple modular redundancy or principles of self-checking circuits [22] have been optimized for FPGAs [23–25] and extended by FPGA-specific techniques like scrubbing [18].

Sensors in the system allow to monitor the circuit behavior and temperature [26, 27]. The sensor data can be aggregated

and used to predict failures. With failure detection or prediction at hand, systems based on reconfigurable fabric can perform self-repair [28–30] to ensure that mission logic does not use faulty hardware blocks of the fabric. A framework that dynamically adapts its fault tolerance level to varying soft error rates is proposed in [31].

Dependable systems based on runtime reconfiguration require that both the reconfigurable fabric is fault-free and the reconfiguration process completes without error. Runtime online test is an efficient way to check whether the fabric is fault-free and the reconfiguration process completed correctly. Test complements concurrent error detection techniques allowing to prevent fault accumulation in the system and thus, is a requirement in any highly reliable architecture.

2.3 Test Methods for Reconfigurable Architectures

The thorough test of the FPGA fabric requires detailed structural knowledge. The test generation is typically tailored to a specific FPGA architecture. Application-dependent tests [32, 33] target the faults relevant for a known, fixed application of the FPGA. Such test schemes have to be adapted for each application.

In contrast, application-independent tests target the whole fault universe not limited to a specific use of the reconfigurable resources. Such a test consists of multiple test sessions, each comprised of a *test configuration* (TC) and a set of stimuli. A TC configures the FPGA in a way that a set of the structures is controllable and observable so that appropriate test stimuli can be applied to test for faults. The number of required test configurations may range from a few up to a few hundreds if programmable interconnect structures are completely included in the tests [34].

Different test strategies are used for the logic and sequential parts of CLBs, interconnects, I/O cells and specialized cores like RAM or DSPs. For memories, high-coverage functional March tests [35] are used, for arithmetic structures like multipliers or DSPs, functional tests with high structural fault coverage are possible [36].

For an online in-field test of FPGAs, external equipment for *test pattern generation* (TPG) and *output response analysis* (ORA) [37, 38] is not available. Internal testing approaches based on *built-in self-test* (BIST) principles include TPGs and ORAs in the chip.

The highly regular nature of FPGAs allows to configure the structures under test into an iterative logic array [39] such that the test time is constant and independent of the array size (C-testability) [40]. FPGAs with support for memory readback allow a test strategy similar to scan design where the response is captured in sequential elements, read back, and finally analyzed [17, 41]. The readback increases the test time.

While the programmable interconnect structures are to some extent already tested during the test of other structures, dedicated testing based on multiple test configurations has been proposed [42–44]. Due to the complexity of the interconnect configuration circuitry, a high number of TCs is required. This number can be reduced by testing only the interconnect resources used by the target application [45, 46].

Using partial dynamic reconfiguration of FPGAs, the test reconfiguration can be conducted by an external [47] or embedded processor at runtime [48–50]. The access to an internal high-bandwidth configuration port significantly reduces the reconfiguration time

In addition to testing, the homogeneous structure of an FPGA allows the efficient diagnosis of faulty components. High resolution is achieved by additional TCs to distinguish faults with equal signatures [51].

Both test and diagnosis can be executed offline, requiring idle periods of the unit under test, or non-concurrently online, allowing the parts of the array which are currently not under test to continue functional operations. The Roving STARs (self testing areas) method [52] partitions the FPGA into multiple regions which can be either used functionally or tested by an online BIST scheme controlled by an external module.

Scrubbing [18], i.e. continuous reading and overwriting the configuration data, detects and repairs errors in the configuration memory, but fails to detect structural faults in the CLBs It needs to access the configuration port frequently, which conflicts with functional runtime reconfigurations and thus greatly degrades the performance of the reconfigurable architecture.

In this work the structural test of the reconfigurable fabric is tightly integrated into the functional system scheduling. The existing reconfiguration features in the system are reused as much as possible.

2.4 Online Test Scheduling

Test scheduling strategies for core-based systems have been proposed targeting the optimization of test time, test power, and test access bandwidth [53]. For uniform multi-core systems, online test schemes with dedicated test controllers have been introduced [54], which isolate target cores, i.e. stop the execution and save the critical state before the application of stored test patterns. Such scheduling approaches have been extended to the operating system level [55] and to the utilization of virtualization techniques [56] for task migration from cores under test. With respect to runtime-reconfigurable systems with challenging resource and runtime constraints, and heterogeneous accelerators, these approaches are of limited use.

3 Online Testing for Reconfigurable Architectures

The correct operation of mission logic, instantiated into a reconfigurable container, mandates that the underlying reconfigurable fabric is free of defects, and the reconfiguration process is conducted without error.

This section details the design of structural pre-configuration online tests (PRET) and post-configuration online tests (PORT). Reconfiguration in the target system is based on containers of a known size, that can be isolated from the system for testing, and that provide an access for test stimuli (Fig. 1). All tests are executed at full system speed.

3.1 PRET: Pre-Configuration Online Test

To check that the reconfigurable fabric of a container is free of defects, it is necessary to exercise the fabric such that effects of

defects become observable. This is achieved by the structural pre-configuration online test [8] which targets the following faults in the fabric.

3.1.1 Fault Model

The *stuck-at fault model* is widely adopted in the literature for FPGA testing [40]. For complex FPGA sub-components such as lookup tables (LUTs) and flip-flops, typically only a functional description is available from the vendor and structural implementation details are missing. This results in a weak modeling of defects. Here, the stuck-at fault model is used for components in which the structural information is sufficient for fault derivation and for the interconnects. For the remaining components, structural and cell faults are targeted during test generation resulting in a hybrid fault model. The tests are generated under the single fault assumption.

- *LUT in function mode:* The LUT in function mode is treated as a combinational function of n inputs and m outputs, and the cell fault model [57, 58] is applied. Cell faults describe any mismatch at the outputs of a unit under test for the possible inputs. The number of cell faults equals the number of possible inputs multiplied by the number of faulty outputs $2^m(2^n - 1)$.
- *LUT in RAM mode:* If the LUT is operated in RAM mode, the following memory faults [35] are targeted: stuck-at faults, address decoder faults, transition faults, coupling faults, and data retention faults.
- *Sequential elements:* CLBs contain separate sequential elements such as flip-flops, latches, or LUTs in shift register mode. For these elements, we consider the commonly used fault models in hardware testing, i.e. stuck-at and transition faults [53].

3.1.2 Test Configuration Generation for CLB Elements

Due to the complexity of a CLB, it requires multiple TCs to exercise all of its components. In each TC, the fabric under test is configured such that a subset of the components in the CLBs is activated and tested. The test configurations are designed to guarantee full fault coverage. Testing is applied per container. Each TC consists of two main parts:

1) Container setup: CLBs are configured in a specific way to ensure the test of targeted faults.
2) Test stimuli: Applied to exercise the configured components in the CLBs.

In addition, test infrastructure is required to generate the stimuli with a *test pattern generator* (TPG) and evaluate the responses with an *output response analyzer* (ORA). The TPG and ORA may differ between TCs. They are external to the container under test to cover the container's bus interface.

The regular structure of the reconfigurable fabric allows the efficient and scalable test of large containers by connecting the CLBs into C-testable arrays as exemplified in Fig. 2. The CLBs in the array are configured and interconnected such that each CLB receives all required test patterns at its inputs from the TPG or via its predecessor. The test responses of the CLB are propagated to the ORA via its successors. This ensures equally high controllability and observability of all CLBs in the array during testing. The resulting test time for the array is very low and independent of the array size [40].

A TPG feeds the test patterns to the array and the responses are aggregated and evaluated using an ORA. For the TPG, a counter is used to generate the exhaustive test set. Responses are evaluated by mutual comparison, as indicated in Fig. 2. To avoid a slow test clock, the TCs are pipelined by utilizing the sequential elements in CLBs.

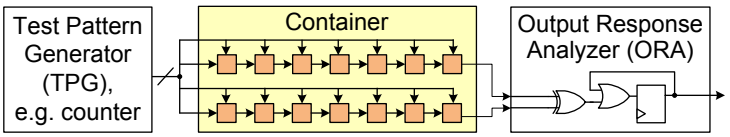

Fig. 2. Container configured into a C-testable array with external TPG and ORA.

For the LUT in function mode, all cell faults are targeted. This is achieved by two TCs [40], and by applying the exhaustive set of test patterns to the inputs. For the shift register mode of the LUT, stuck-at and transition faults are tested by applying standard scan chain test patterns [59]. The LUTs in shift register mode are connected into multiple shift chains. The outputs of these chains are compared mutually for response evaluation. Individual flip-flops in CLBs can be simultaneously tested in the same TC by including them into the chain. An n-input LUT can also implement a 2^n-bit RAM which is tested by the MATS++ [35] algorithm to ensure coverage of all stuck-at faults, address decoder faults, and transition faults. Test patterns are generated by the global TPG.

The multiplexers in the CLBs are tested by applying all possible configurations to exercise all combinations of the select signals. The data path is tested for stuck-at faults by applying the 0 and 1 stimuli. Multiplexer testing is typically included in other tests since they are on the sensitized path used for testing other components.

Many FPGA architectures contain dedicated carry chains consisting of multiplexers and XOR cells. To test for all the stuck-at faults in the carry chain efficiently, the elements are configured into pipelined C-testable arrays.

Testing flip-flops in CLBs is identical to testing the LUT in shift register mode. If there are level sensitive latches, a separate TC is required which creates a scan chain of latches. To guarantee proper latch function, two non-overlapping clocks are used. A detailed discussion is given in [8].

For a container consisting of multiple CLBs, the TCs are generated according to Fig. 3. In the first step, the targeted CLBs in the container are selected, depending on its size and location. Then, the required TPG and ORA for the different tests are generated. In the last step, the configuration of the container is created by instantiating TC templates for the selected CLBs. The resulting configuration data is stored in an XDL (Xilinx Design Language, [60]) file.

3.1.3 TCs for Application-Dependent Interconnect Test

During system operation the set of reconfigured hardware accelerators uses only a fraction of the interconnect resources, such as wires and programmable interconnect points (PIPs). Instead of testing the complete interconnect configuration circuitry, only the interconnect resources used by the target application are tested in an application-dependent pre-

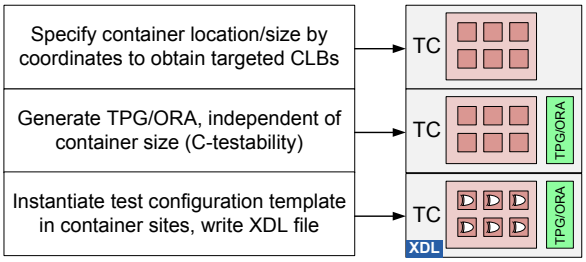

Fig. 3. XDL file generation flow for CLB TCs.

configuration test. Thereby the number of required test configurations is much lower than in an exhaustive test of all configuration possibilities of the PIPs.

The required TCs for this interconnect test are derived from the accelerator configurations (ACs) such that a high controllability and observability of the used interconnects is achieved. For each accelerator, the interconnect structure under test is given by its AC and taken over into the corresponding TC. Since the logic functions in the ACs may reduce controllability and observability, the functions are modified in the TC. All logic functions implemented in CLBs in the AC are changed to the XOR function. All faults at interconnect lines are observable at the outputs unless all propagation paths to outputs are reconverging multipaths with even branch multiplicity (fault coverage is analyzed in Section 5.2).

Each test configuration is now transformed to an equivalent simulation model containing FPGA primitives such as wires, multiplexers or LUTs. The behavioral description of these primitives is provided by Xilinx in the SIMPRIM library. A commercial automatic test pattern generation (ATPG) tool is used to generate the interconnect test patterns. The list of targeted faults consists of stuck-at faults assigned to all in- and output ports of the contained primitives such that all faults at signal fanout stems and their branches are tested.

The precomputed test set is applied once the TC is configured into a container. The container under test computes a signature over all test responses which is then compared with the stored expected signature.

3.2 PORT: Post-Configuration Online Test

The fabric's structural integrity is assessed by executing PRET. However, this does not guarantee the correct configuration and integration of mission logic in a container. Errors may occur during the configuration process:

- due to faults in the configuration logic, and/or
- due to corruption of the bitstream in system memory or during transmission of the bitstream caused by transient events such as crosstalk or radiation.

In consequence, the configured function of the targeted container may be wrong. Even worse, the configuration of other containers may be adversely altered in case of address decoder faults or errors in the configuration address in the bitstream.

The *post-configuration online test* (PORT) tests the function of the reconfigured hardware accelerators. Conducting PORT directly after reconfiguration assesses whether the reconfiguration process completed without errors and that the expected functionality is delivered. A periodic PORT execution during operation checks for malfunctions occurring during phases where no reconfiguration occurs, for example the corruption of configuration bits due to soft errors.

The PORT is conducted by deterministically generated patterns exercising structures with low controllability or observability. PORT does only test the parts of the container fabric which are used by the mission logic. Both built-in self-test as well as software-based self-test principles for test generation and response evaluation can be exploited. In contrast to PRET, PORT does not reconfigure the tested structures and thus exhibits a much lower performance impact on the application.

3.2.1 Test Set Generation

Post-configuration online test patterns are generated for each accelerator. A deterministic pattern set exercising the given accelerator is derived similar to the steps in Section 3.1.3. The configuration bits of the contained primitives define their behavior and are specified as model parameters called init-strings (Fig. 4-a). The models of the primitives are transformed such that deterministic test patterns can be generated using a commercial automatic test pattern generation (ATPG) tool.

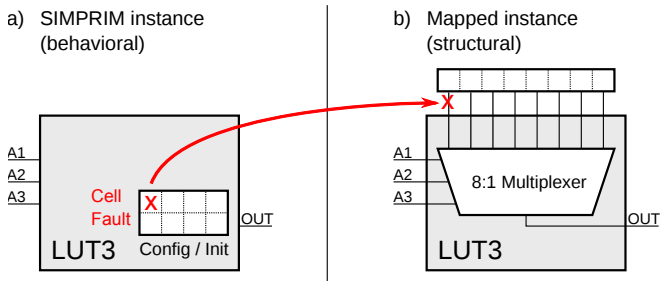

Fig. 4. Mapping of configuration bits for a 3-input LUT.

This transformation requires a detailed structural modeling of the configurable primitives. We map the behavioral simulation models to a structural description based on multiplexers such that the configuration bits are represented as separate signals. E.g. each n-input LUT containing 2^n configuration bits is mapped to a tree of $2^n - 1$ 2:1 multiplexers. This mapped model is processed by the commercial automatic test pattern generation tool and yields a test pattern set targeting stuck-at faults on internal signals as well as in configuration bits, i.e. cell faults in the used LUTs (Fig. 4-b).

3.2.2 Test Execution

The post-configuration online test execution is possible by two strategies: Each configured accelerator is tested with its accelerator specific pattern set (Section 3.2.1). Alternatively, a unified test set which consists of the union of all generated accelerator specific test sets is used for all containers. This test set is effective for all considered accelerators and is efficiently applied to all containers in parallel using a broadcast on the bus structure (similar to multi-site testing). Each container calculates a test signature over all test responses of the applied pattern set using a 32-bit multiple input signature register (MISR, [53]). By comparing the signature of each container with the expected signature of the configured accelerator the test result is calculated.

4 System Integration and Scheduling

4.1 Integration Overview

This section explains how the developed test manager that contains the *test pattern generator* (TPG) and the *output response analyzer* (ORA) for PRET and PORT is integrated into a runtime-reconfigurable system. Fig. 5 shows an excerpt of a reconfigurable fabric with three containers and the components involved in testing them. In the first step (Fig. 5a), the runtime system decides that an accelerator shall be reconfigured into a particular container to implement a *Special Instruction* (SI), which triggers the demand to test the container first (a so-called *on-demand PRET*). Details on those decisions of the runtime system (available in [14, 61]) are beyond the scope of this paper and are orthogonal to the presented approach. As performing an exhaustive PRET would delay the accelerator reconfiguration significantly, PRET is executed incrementally. This means that not all *Test Configurations* (TCs) are applied to the container at once, but only some of them. The runtime system decides how many TCs should be applied (potentially none) before reconfiguring the accelerator, depending on the test history. The runtime system tracks which TCs were applied to a container in the past and how much time passed since the last exhaustive PRET of the container. Depending on this test history, it activates PRET, reconfigures the selected TCs into the container, and uses TPG and ORA to exercise the reconfigurable fabric (Fig. 5b). In addition to the on-demand PRETs, the runtime system also schedules *periodic PRETs* to ensure that also seldomly reconfigured containers are tested.

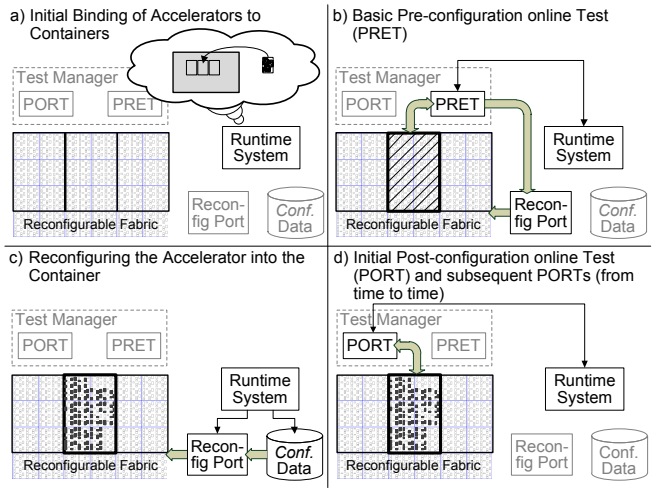

Fig. 5. Typical runtime flow with PRET and PORT.

If no structural defect is found by PRET, the runtime system reconfigures the desired accelerator into the container (Fig. 5c). Before the accelerator is used in an SI, the runtime system triggers an *on-demand PORT* (Fig. 5d) to test whether the reconfiguration process completed without error, i.e. whether the accelerator is operational. Additionally, accelerators instantiated in other containers are tested as well to check that they were not affected by the reconfiguration. As PORT does not require any reconfiguration, it operates significantly faster than PRET and can also be applied during normal operation. The runtime system schedules subsequent *periodic PORTs*.

4.2 Integration Details

The presented PRET and PORT are conceptually orthogonal to a particular system architecture. This section describes their integration into the reconfigurable processor architecture that is shown in Fig. 1 (the baseline architecture). Fig. 6 shows the resulting architecture after the test manager is integrated with the reconfigurable containers.

To execute PRET or PORT, the test manager needs to communicate with the container under test. This communication is established by the same interconnect infrastructure that is already available to establish inter-container communication for SI executions. In order to utilize this infrastructure, both PRET and PORT are implemented as dedicated test-SIs. In the baseline architecture, all SIs contain an implicit control word which is not part of the SI assembly instruction, but stored in an on-chip memory for each executable SI. This control word configures the interconnect infrastructure according to the SI requirements, i.e. depending on which containers are configured with the required accelerators. For the test-SIs, the same mechanism is used to establish the connection between the test manager and the container under test. Additional details can be found in [61, 62], but are not required for understanding the test schemes described in this paper.

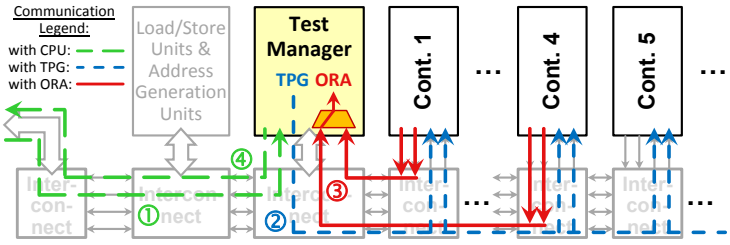

Fig. 6. Test manager integration with TPG and ORA.

For PRET, the test configuration needs to be reconfigured into the container under test before the test-SI can be executed. Sending the test patterns to the container under test and analyzing the responses is similar for PRET and PORT and is performed by the test-SIs. When the processor executes a test-SI, the SI parameters are sent from the register file to the test manager, as shown in Fig. 6 step ①. The parameters determine which container is tested and which test patterns are applied. The test manager sends the test patterns to all containers (Fig. 6 step ②). The patterns are generated in the test manager, or stored along with the expected output signatures in an on-chip memory (see evaluation in Section 5). The memory is attached as a slave to the system bus and initialized when the system starts.

For PRET targeting subcomponents of CLBs, a test pattern and its corresponding response fit into one 32-bit word each. For each test pattern, the responses of 4 containers are sent back to the test manager (Fig. 6 step ③). The limitation to 4 responses per cycle is due to the availability of 4 buses for the interconnect infrastructure. The example in Fig. 6 step ③ shows how the responses of containers 1–4 are sent back via the buses. The test manager then selects the response of the container under test with an internal multiplexer. To be able to perform PRET on containers 5–8, another test-SI is used that configures the interconnect infrastructure such that the results of these containers are sent back.

During PORT, the responses of the accelerators are compacted in space and time as explained in Section 3.2. After the application of the test patterns, a single 32-bit signature per container has been computed and is stored locally. To ensure that the outputs and the bus interface of a container are also tested, the hardware that computes the signature is integrated into the interconnect infrastructure. PORT only needs one test-SI that tests all containers at the same time. After applying all test patterns, the locally stored signatures are sent to the test manager in multiple cycles (four signatures per cycle). The test manager then compares the signatures with the expected signatures that are specific for each accelerator. The information which accelerator is reconfigured to which container is available in the hardware architecture and is updated before and after each reconfiguration.

For PRET targeting interconnects, an interconnect test configuration is reconfigured into the container under test similar to PRET for CLB subcomponents. Then, stored test patterns are applied similar to PORT, and a test signature is computed and sent back to the test manager.

At the end of the tests (PRET or PORT), the final test result (passed or failed) is written back to the register file (Fig. 6 step ④). As the PORT test-SI tests all containers at the same time, the result contains a 1-bit information (passed or failed) for each container.

4.3 Test Scheduling

4.3.1 PRET Scheduling

The cost of PRET execution is high as time-consuming reconfigurations have to be performed. Thus, we aim at reducing the application impact by executing PRET at times when the system needs to be reconfigured anyway. An on-demand PRET *Test Configuration* (TC) is scheduled after a certain number of *Accelerator Configurations* (ACs). For instance, one TC is scheduled before every AC or before every 2nd AC, 3rd AC, etc. This allows to distribute the tests over space (number of containers) and time (test frequency of a particular container). The tests are initiated by the runtime system that is responsible for scheduling the accelerator reconfigurations. After a reconfiguration completes, the runtime system is informed by an interrupt. If that reconfiguration was a test configuration, the runtime system executes a PRET SI for the corresponding container and evaluates its result before triggering the next reconfiguration.

In addition to on-demand PRETs before accelerator configurations, a timer-based periodic PRET is realized. This is required to limit the test latency for containers which are only rarely reconfigured by the running application. The periodic PRET is implemented by a handler (see Algorithm 1) consisting of two phases: i) triggering the reconfiguration of a TC for a particular container (lines 3–14) and ii) executing the PRET SI after the TC is reconfigured (lines 15–29).

The first phase of the handler scans all containers for their last test time (maintained by a data structure of the runtime system) and identifies the least recently tested container (lines 4–6 in Algorithm 1). If the time since the last test is larger than a configurable threshold (we use 500 ms for evaluation in Section 5), a periodic PRET is triggered for this container.

The second phase of the handler is activated when the reconfiguration of the test configuration triggered by the first phase is completed. It then executes the PRET SI, informs the runtime system about the health state of the container under test and updates the data structures for the next PRET.

Algorithm 1 Interrupt Handler for periodic PRET using the 'Least Recently Tested Strategy'.

Require: Trigger by $timer\_event$ and $reconf\_complete\_event$
Require: $cont[i]$: runtime system information about containers
1: static $pret\_cont$:= NULL; // which container
2: static $pret\_tc$:= NULL; // which test configuration
3: **if** (triggered by $timer\_event$) **then**
4: // determine least recently tested container
5: $lrt\_cont := MIN_{\forall Container\ i}\ \{cont[i].last\_test\_time\}$;
6: $lrt\_time := cont[lrt\_cont].last\_test\_time$;
7: **if** ($current\_time - lrt\_time >$ Threshold) **then**
8: $pret\_cont = lrt\_cont$;
9: $pret\_tc = cont[lrt\_cont].next\_tc$;
10: // Trigger the reconfiguration of the test config.
11: $reconfiguration\_queue.push(pret\_cont, pret\_tc)$;
12: **return**
13: **end if**
14: **end if**
15: **if** (triggered by $reconf\_complete\_event$ **and** $pret\_cont \neq$ NULL **and** $cont[pret\_cont].accelerator = pret\_tc$) **then**
16: **switch** ($pret\_cont$)
17: **case** $0-3$**:**
18: $result = pret\_si\_cont0\_3(pret\_cont)$;
// this calls the test-SI for containers 0–3
// parameter: which of these 4 containers to test
19: **break**
20: **case** $4-7$**:**
21: $result = pret\_si\_cont4\_7(pret\_cont - 4)$;
22: **break**
23: **end switch**
24: $cont[pret\_cont].health\_state := result$;
25: $cont[pret\_cont].last\_test\_time := current\_time$;
26: $cont[pret\_cont].next\_tc := (cont[pret\_cont].next\_tc + 1)$ **mod** $number\_of\_tcs$;
27: $pret\_cont$:= NULL;
28: $pret\_tc$:= NULL;
29: **end if**

4.3.2 PORT Scheduling

The on-demand PORT is conducted directly after reconfiguration to assure that the reconfiguration process has correctly completed without error and that the configured accelerator delivers the expected functionality. As PORT tests all containers in parallel (see sections 3.2 and 4.2), errors in the other accelerators, e.g. due to address decoder faults in the configuration logic or errors in the configuration address, are detected as well.

In addition, periodic PORTs are also scheduled over runtime to check the accelerators for malfunctions, caused e.g. by emergent faults in CLBs or soft errors in configuration bits. Periodic PORT is realized by an interrupt handler similar to the periodic PRET handler (Algorithm 1), but without the need to trigger reconfigurations.

5 Evaluation

This section investigates the overhead and test effectiveness of PRET and PORT, integrated into the reconfigurable architecture described in Section 4.

5.1 Experimental Setup

The reconfigurable architecture introduced in Section 4 forms the platform for the experimental evaluation. A Leon processor [63] is used as core pipeline with a configurable number of attached containers (see Fig. 1). The actual hardware prototyping is performed on an XUPV5 FPGA board with a Xilinx Virtex-5 LX110T. Our prototype has 5 runtime-reconfigurable containers with 4x20 CLBs per container. We have integrated the test manager into the prototype (as shown in Fig. 6) and implemented test-SIs to perform tests. A SystemC-based simulator (parameterized by the hardware prototype) is used for evaluating different system parameters like the number of containers. The system operates at a clock frequency of 100 MHz and a reconfiguration bandwidth of 50 MB/s (limited by off-chip system DRAM that is also used to store partial bitstreams).

A sophisticated H.264 video encoder is chosen as target application. The encoder is a challenging application since it frequently reconfigures accelerators in the containers. The H.264 encoder consists of three different functional blocks that are executed in sequence for each video frame: Motion Estimation (ME), Encoding Engine (EE), and in-loop deblocking filter. Each functional block requires different Special Instructions (SIs, implemented by multiple different accelerators) that are reconfigured when the block executes. For instance, when EE processes a frame then the SIs for EE are reconfigured which replaces the SIs for ME that finished processing this frame before EE started. The SI requirements for a particular computational block may vary over time. For instance, EE uses different encoding techniques (accelerated by different SIs) depending on the input data, e.g. slow moving objects vs. hectically changing structures.

In total, 9 SIs are implemented for the H.264 encoder by using combinations of 9 different types of accelerators (see Table 1). More details about the developed SIs and accelerators for H.264 are available in [64]. The implementation of the H.264 encoder on the reconfigurable system leads to a speedup of more than 20× in comparison to the Leon processor without SIs.

TABLE 1
Short description of accelerators implemented for H.264.

Accelerator	Description
Clip3	clipping to a configurable min/max interval
CollapseAdd	summing up the 4 bytes inside a 32-bit word
LF_BS4	4-pixel edge filter for in-loop deblocking
LF_Cond	filtering condition for in-loop deblocking
PointFilter	six-tap filter for sub-pixel motion estimation and compensation
QuadSub	4 parallel byte subtractions
SADrow_4	sum of absolute differences of two 4-pixel rows
SAV	sum of absolutes of four 16-bit values
Transform	(inverse) DCT or (inverse) Hadamard transform.

5.2 PRET Results

The PRET overhead and test effectiveness are discussed in the following. Additional hardware blocks for TPG and ORA as well as memory for the generated test patterns are required. This introduces a small area overhead discussed in Section 5.2.1 but has no effect on the system clock frequency. Tests are scheduled periodically or on-demand to containers like functional workloads. PRET execution may delay the configuration of accelerators or consume communication resources. The resulting impact on the performance as well as the test effectiveness are presented in Section 5.2.2.

5.2.1 Test Configurations

A full test session consists of multiple test configurations (TCs), each of which tests a subset of the components in the CLBs of a container or a set of interconnects used in one of the accelerator configurations as explained in Section 3.1. Altogether nine TCs are required to test all subcomponents in the CLBs, and another nine TCs are required to test the interconnects of the nine accelerators of Table 1. Partial bitstreams for these TCs are generated and stored in memory.

Table 2 provides an overview of the 18 TCs, the 9 CLB TCs labeled TC 1-9 and the 9 interconnect TCs labeled 10-18. Column one shows the configuration number. Column two shortly describes the parts of fabric under tested. Columns three and four list the PRET overhead in CLBs used for the TPG and ORA and the size of the generated partial bitstream. The total area overhead introduced by PRET for all TCs is 17 CLBs. The test time for a container consists of two parts: the container configuration time and the test pattern application time (see 'Test length' in Table 2). Typically, the latter ranges from a few cycles up to a few hundred cycles. For instance, applying all test patterns for TC 9 (the TC with the largest number of patterns) lasts 3.2 μs at 100 MHz system frequency. The container configuration time dominates the test time with tens of thousands of cycles and is directly proportional to the size of the configuration data (partial bitstreams) and the reconfiguration bandwidth. As shown in Table 2, the bitstream size of each TC varies from 22.3 KB to 29.6 KB, which corresponds to a configuration time between 0.45 ms and 0.59 ms at 50 MB/s configuration bandwidth, i.e. between 45 and 59 thousand cycles at 100 MHz system frequency.

The PRET overhead for the interconnect TCs is not applicable as the deterministic patterns are not generated by a TPG but stored similar to PORT patterns. For response compaction we reuse the response compaction unit of the PORT. In total 3780 bytes are required to store the test patterns of all interconnect TCs together with their signatures. One of the nine accelerators (and its corresponding TC) requires two clock cycles for execution (78.8 MHz). All others require only a single clock cycle and have a frequency higher than 100 MHz. All interconnect TCs reach a fault coverage of 100%, except for SADrow_4 with a fault coverage of 98.28%.

5.2.2 PRET Scheduling

Fig. 7 shows the simulation results for the performance loss under different test frequencies, depending on the number of reconfigurable containers. The test frequencies vary from one test configuration before every accelerator configuration (1 TC/AC) to one test configuration before every 4^{th} accelerator configuration (1 TC/4 ACs). Using a lower test frequency (e.g. 1 TC/4 ACs) reduces the overhead. The PRET handler is triggered every 1 ms and performs PRET if a container has not been tested for 500 ms. For reference, in a system with 10 containers and without PRET/PORT, the time between two

TABLE 2
Test configurations for CLBs and interconnects: Overhead, size, frequency and length.

TC	Tested components	PRET overh. [CLBs]	Bitstr. size [KB]	Freq. [MHz]	Test length [Patterns]
1	LUT conf. as XOR, connected to FF	2	24.0	207	64
2	LUT conf. as XNOR, connected to FF	2	24.0	207	64
3	Carry MUX, interleaved with MUX and latch	1	28.6	168	6
4	Carry MUX, interleaved with MUX and latch	1	26.1	154	6
5	Carry XOR, interleaved with MUX and FF	1	28.0	168	6
6	Carry XOR, interleaved with MUX and FF	1	28.2	154	6
7	Carry-in/-out with multiplexed scan chain	1	27.1	183	6
8	LUT conf. as SR with slice MUX	1	22.9	157	6
9	LUT conf. as RAM with slice output	7	22.3	225	320
10-18	Interconnect and PIPs of the nine accelerators	n.a.	29.6	78.8-191.9	13-123

consecutive accelerator configurations in a container ranges from 13.2 ms to 1240 ms (average: 200 ms).

Reconfigurable architectures with more containers have a lower overhead as more containers are still operational during the test application. For instance, in an architecture with 5 containers, only 4 containers can be used for SIs during the PRET reconfiguration and pattern application period, whereas in an architecture with 14 containers, still 13 containers can be used for acceleration.

Fig. 8 shows the average test latency. For example, for an architecture with 10 containers and a test frequency of 1 TC/3 ACs, each container is completely tested every 7.1 seconds while the performance loss introduced by PRET is only 0.5%. The observed test latencies (3.8 s to 8.1 s) show that emerging faults do not remain undetected in the system for longer than 1.9 s to 4.05 s in average.

As shown in Fig. 9, with decreasing on-demand PRET frequencies, the number of on-demand tests (solid lines) decreases while the number of periodic tests (dashed lines) increases. A low on-demand PRET frequency increases the chance that a periodic test will be triggered because the possibility that a container remains untested for a time that

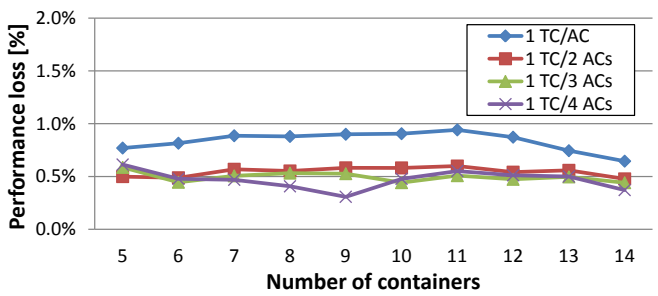

Fig. 7. Performance loss of the video encoder application under different on-demand PRET frequencies and number of containers.

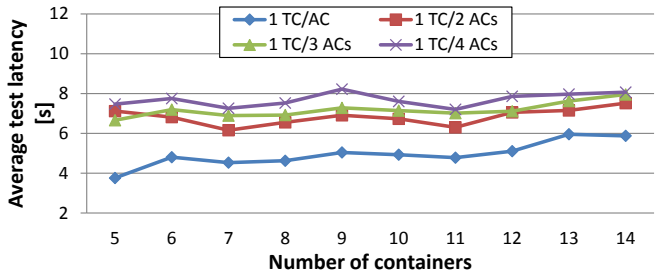

Fig. 8. Average test latency under different PRET frequencies and number of containers.

exceeds the threshold (in our experiment 500 ms) is higher. For reconfigurable systems with a large number of containers, all accelerators may fit into the containers at the same time. Hence, less reconfigurations are required leading to fewer on-demand tests and a higher number of periodic tests.

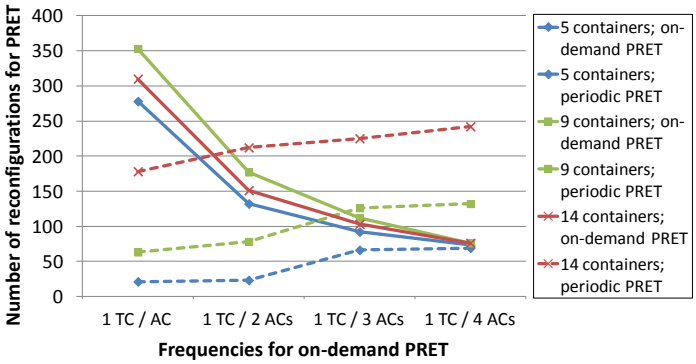

Fig. 9. Comparison of the number of on-demand and periodic tests for different on-demand PRET frequencies and number of containers.

5.3 PORT Results

This section discusses the achieved fault coverage and test overhead of PORT.

5.3.1 Fault Coverage

The fault coverage of the test patterns applied during PORT is evaluated by fault simulation. The targeted fault set depends on the CLB usage of the accelerator and is computed according to the hybrid fault model (see Section 3.1.1). It is the union of the stuck-at faults in the used CLBs and interconnects, as well as the functional cell faults for all used LUTs.

The fault simulation is based on the synthesized, mapped and routed accelerators, for which simulation models based on Xilinx SIMPRIM instances are generated (see Section 3.2.1). E.g. Clip3 contains 87 LUT2, 78 LUT3, 128 LUT4, 27 LUT5 and 37 LUT6 instances containing 6252 LUT configuration bits which are mapped to 5895 2:1 multiplexers. In these instances, a number of address inputs to LUTs have been assigned constant values during synthesis, effectively masking out a part of the memory elements in the LUT. These blocked cell faults are functionally irrelevant and untestable under the single fault assumption. Hence, these cell faults are excluded from the evaluation of the fault coverage (resulting in 6162 LUT configuration bit faults for Clip3).

Table 3 shows for each accelerator i the amount of essential bits (the bits associated with the design, a true subset of the

configuration bits), the number of deterministic test patterns P_i, as well as the numbers of LUT and stuck-at faults (F_i) according to the underlying fault model, and the achieved fault coverage (FC_i) using the union of the individual test pattern sets. For accelerators Clip3 and SADrow_4, the fault coverage of cell faults in the LUTs is low. These faults are proven untestable by the test generation tool, i.e. it is not possible to select the value in the LUT by an input pattern and propagate the value to an observable output.

TABLE 3
Accelerator dimensions and achieved fault coverage of PORT for the hybrid fault model.

Accelerator	Essential config. size *[bits]*	P_i *[count]*	LUT config. bit faults F_i *[count]*	LUT config. bit faults FC_i *[%]*	Stuck-at faults F_i *[count]*	Stuck-at faults FC_i *[%]*
Clip3	65,608	549	6,162	55.84	1,644	92.63
CollapseAdd	22,836	176	1,012	100.00	728	90.24
LF_BS4	48,653	241	2,184	98.12	1,528	98.03
LF_Cond	27,681	1038	2,704	95.67	784	89.28
PointFilter	86,866	244	6,502	89.26	3,230	96.56
QuadSub	18,984	16	128	100.00	622	95.81
SADrow_4	39,380	715	4,060	65.96	1,164	95.70
SAV	34,242	393	2,170	96.26	1,102	92.83
Transform	48,762	218	4,136	100.00	1,356	98.52

Altogether, PORT sends 3,590 test patterns to the containers which takes 35.9 μs at 100 MHz system frequency. The test patterns and expected signatures for specific accelerators are stored in an on-chip memory. For the accelerators in Table 3, altogether 28,756 bytes are required for the test patterns and signatures, which is roughly the size of a test configuration bitstream for PRET (see Table 2). The required memory occupies only 7 Block RAMs, approx. 4% of on-chip memory resources in the Virtex-5 LX110T FPGA.

5.3.2 Estimation of PORT Effectiveness

We estimate the effectiveness of PORT-based periodic accelerator testing by a stochastic model of the average time period T_{CRIT} where a fault is critical, i.e. it remains undetected in the system. During this period T_{CRIT}, a fault might cause data corruption or system failure.

Since transient faults occur at a much higher rate than aging induced faults, we focus on soft error induced corruption of the configuration bits causing structural or functional changes in the accelerator. A fault is thus either detected by PORT or removed by a regular container reconfiguration (configuring a new accelerator).

T_{CRIT} depends on the fault rate in the configuration bits of the accelerators, the frequency of reconfigurations f_C and PORT executions f_P, as well as the fault coverage of PORT. Since our reconfigurable system allows to use multiple accelerators in different containers, we compute T_{CRIT} as the sum of the critical time T^i_{CRIT} of each accelerator i , weighted by U_i, the fraction of time accelerator i is instantiated and used. The soft error rate of an accelerator SER_i depends on the system soft error rate SER and the number of configuration bits in the accelerator: $SER_i = SER \cdot size_i$. Soft errors are assumed to be uniformly distributed in space and time. Finally, with the fault coverage FC_i of PORT for accelerator i and mission time t, we obtain:

$$T_{CRIT}(t) = \sum_{acc.\,i} T^i_{CRIT}(FC_i, f_P, f_C) \cdot U_i \cdot SER_i \cdot t \quad (1)$$

To estimate T^i_{CRIT} of an accelerator in the system, we assume that reconfigurations (either caused by the application or PRET) are performed with fixed frequency f_C. PORT fault coverage, accelerator size and usage are known from fault simulation (Table 3), synthesis results and application profiling.

The computation of T^i_{CRIT} distinguishes the cases that: (a) the fault occurs just before a reconfiguration and is removed before PORT is executed, (b) the fault occurs, PORT is executed and detects the fault such that the system can trigger PRET or a reconfiguration, and (c) the fault occurs, PORT is executed without detecting the fault which remains undetected until the next reconfiguration. Assuming that the PORT frequency is higher than the configuration frequency $f_P \geq f_C > 0$ Hz:

$$T^i_{CRIT}(FC_i, f_P, f_C) = \overbrace{\frac{f_C}{f_P} \cdot \left[\frac{1}{2}\frac{1}{f_P}\right]}^{(a)} + \left(1 - \frac{f_C}{f_P}\right) \cdot \Big[\underbrace{FC_i \cdot \frac{1}{2}\frac{1}{f_P}}_{(b)} + \underbrace{(1 - FC_i) \cdot \frac{1}{2}\left(\frac{1}{f_C} + \frac{1}{f_P}\right)}_{(c)}\Big] \quad (2)$$

$$= FC_i \cdot \left[\frac{1}{2}\frac{1}{f_P}\right] + (1 - FC_i) \cdot \left[\frac{1}{2}\frac{1}{f_C}\right] \quad (3)$$

For $f_P = 0$ Hz, $T^i_{CRIT} = \frac{1}{2}\frac{1}{f_C}$. For $f_C = 0$ Hz and $FC_i = 100\%$, $T^i_{CRIT} = \frac{1}{2}\frac{1}{f_P}$. For $f_C = 0$ Hz and $FC_i < 100\%$, T^i_{CRIT} is not bound.

The graph in Fig. 10 shows T_{CRIT} in seconds for our system with an SER of 1 soft error per day and MBit configuration data and a mission time t of 1 day. T_{CRIT} is given in dependence of the reconfiguration frequency f_C, as well as PORT frequency f_P. At the blue line f_P is equal to f_C. On the right side of the line ($f_P < f_C$), the application of PORT has no additional effect, whereas on the left side ($f_P > f_C$), PORT lowers T_{CRIT}. Fig. 10 shows that T_{CRIT} can be reduced by two orders of magnitude. Even at low frequency, PORT reduces T_{CRIT} significantly and thus increases system availability.

5.3.3 PORT Scheduling

As presented in Section 4.3.2, one PORT is scheduled after each reconfiguration and periodically over runtime to test all containers for functional integrity. Since PORT is implemented as a dedicated test-SI (Section 4.2), the application execution time is affected by PORT. When the total execution time without PORT is t_{base}, then the total execution time with activated PORT t_{PORT} can be expressed as

$$t_{PORT} = t_{base} + t_{PORT} \cdot f_P \cdot d + n_C \cdot d \quad (4)$$

where f_P is the frequency of periodic PORT executions, d is the duration in cycles of one PORT execution, and n_C is the number of reconfigurations. Therefore, the performance loss due to PORT is

$$\frac{t_{PORT} - t_{base}}{t_{base}} = \frac{1 + n_C \cdot d/t_{base}}{1 - f_P \cdot d} - 1 \quad (5)$$

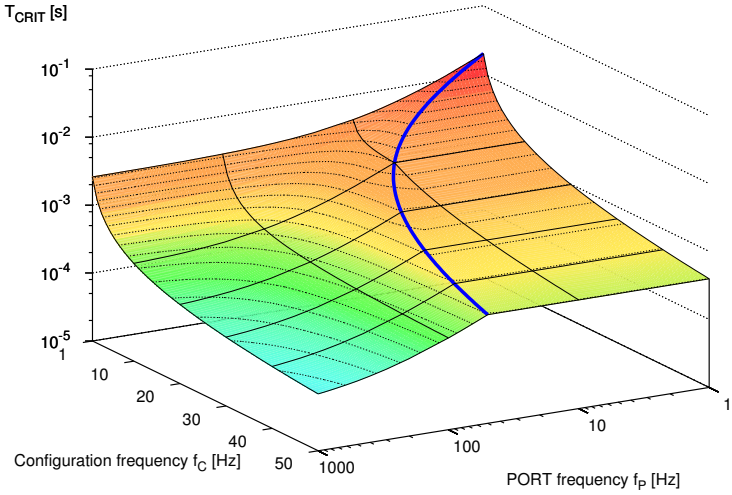

Fig. 10. Fault duration T_{CRIT} until detection/removal depending on configuration frequency f_C and PORT frequency f_P.

Since for the H.264 video encoder the term $n_C \cdot d/t_{base}$ is significantly smaller than 1, the performance loss is dominated by the periodic PORT. We have evaluated this by simulations. The upper part of Table 4 shows the performance loss due to PORT for different PORT frequencies from 143 Hz to 1,000 Hz, i.e. test periods from 1 ms to 7 ms. For each PORT frequency, the table shows the minimum and maximum performance loss of 10 reconfigurable systems with different number of containers (5 to 14). The small difference between the minimum and the maximum values shows that PORT is basically unaffected by an increasing number of containers. This is because one execution of a PORT test-SI tests all containers at the same time (see Section 4.2). Altogether, the performance overhead due to PORT is very low (between 0.51% and 3.73%) and scales well with higher PORT frequencies.

TABLE 4
PORT performance loss and worst case test latency under different PORT frequencies.

	PORT frequency *[Hz]*						
Performance loss	143	167	200	250	333	500	1000
min.* *[%]*	0.51	0.59	0.72	0.89	1.20	1.81	3.68
max.* *[%]*	0.56	0.63	0.75	0.92	1.23	1.85	3.73

	PORT frequency *[Hz]*						
Worst case** test latency	143	167	200	250	333	500	1000
min.* *[ms]*	7.0	6.0	5.0	4.1	3.3	2.3	1.7
max.* *[ms]*	7.8	6.8	5.8	4.8	3.8	2.8	1.8

* Summarizing 10 reconfigurable systems with 5 to 14 containers.
** Corresponds to the longest time period in the whole runtime in which a container remains untested.

In addition to the configured test frequency, the following further situations affect the actual test latency of a container:

(1) If PORT is scheduled while an SI executes in hardware, then PORT must be delayed until the SI execution finishes.
(2) If PORT is scheduled right before or after an accelerator configuration, then all containers are tested twice in a very short period.
(3) During the reconfiguration of a container, no PORT can be executed for that container (no accelerator is available in that container during reconfiguration).

In situations (1) and (3), the test latency of a container is prolonged while in (2) it is shortened. The observed worst case test latency, which corresponds to the longest untested time period of a container is shown in the lower part of Table 4.

5.3.4 Combined PRET and PORT Scheduling

With PRET and PORT both enabled, the reconfigurable system is able to defend the configured accelerators against structural faults induced by aging effects or latent defects and transient events such as crosstalk or radiation. Since both test schemes differ in their test intervals and test methods, they do not interfere with each other. Fig. 11 shows the simulation results for the performance loss of a reconfigurable system with 5 containers when both PRET and PORT are enabled. All combinations of PRET and PORT frequencies used in previous sections are applied.

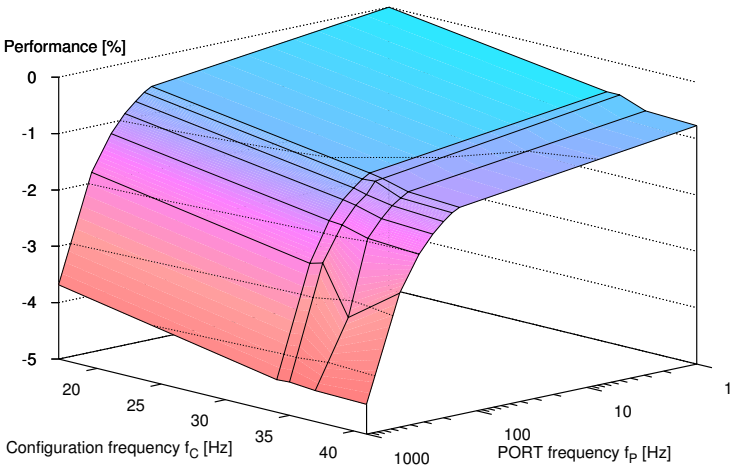

Fig. 11. Performance loss when both PRET and PORT are applied for a reconfigurable system with 5 containers.

The average configuration frequency f_C is determined by considering all reconfigurations, i.e. accelerator configurations (AC) and test configurations (TC). The lowest configuration frequency of 17 Hz corresponds to the case where on-demand and periodic PRET is disabled, i.e. only accelerator configurations are performed. When enabling PRET, the configuration frequency doubles to 34–41 Hz, but due to the PRET scheduling that distributes the TCs over time, the performance loss remains limited.

The highest configuration frequency of 41 Hz in Fig. 11 is obtained for the highest on-demand PRET frequency of 1 TC/AC. For lower PRET frequencies (1 TC/2 ACs and 1 TC/3 ACs), the configuration frequencies reduce correspondingly (35 Hz and 34 Hz). For an on-demand PRET frequency of 1 TC/4 ACs, the configuration frequency increases again (37 Hz), because more periodic PRETs are executed due to the reduced number of on-demand PRETs. That explains the bend that is visible in Fig. 11 for $f_C = 37$ Hz.

For a PORT frequency f_P of less than 100 Hz the performance loss is dominated by the configuration frequency

f_C. After that point, the PORT frequency dominates the performance loss, but also reduces T_{CRIT} significantly (see Fig. 10). The highest performance loss of 4.4% occurs for a PORT frequency of 1,000 Hz and a configuration frequency of 41 Hz. This setup leads to a significantly reduced T_{CRIT} of $64.7\mu s$.

6 Conclusion

This paper presents efficient online test strategies for reliable runtime-reconfigurable systems and their transparent system integration. The *pre-configuration online tests* (PRET) and *post-configuration online tests* (PORT) check if the reconfigurable fabric is faulty and if the reconfiguration processes complete without errors during runtime. The combination of both test schemes and their non-concurrent execution allow a significant reduction of the time a fault remains undetected in the system.

The tight integration of the test schemes into the runtime system and the system scheduling minimize the performance overhead. During the non-concurrent test execution, system operation is only marginally impacted for a few microseconds.

The experimental results show that the application of the presented strategies to a reconfigurable system results in high fault-coverage and low test latency. The time a fault remains undetected in the system is reduced by up to two orders of magnitude at a performance loss of less than 4.4%. A hardware prototype demonstrates the feasibility of the proposed PRET and PORT based online test strategies.

Acknowledgments

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 – http://spp1500.itec.kit.edu).

References

[1] Convey Comput., "Homepage of Convey Comput." http://www.conveycomputer.com/, accessed at August 15, 2012.
[2] S. Kirsch *et al.*, "An FPGA-based High-Speed, Low-Latency Processing System for High-Energy Physics," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2010, pp. 562–567.
[3] P. Garcia *et al.*, "An Overview of Reconfigurable Hardware in Embedded Systems," *EURASIP J. on Emb. Syst.*, pp. 1–19, 2006.
[4] M. Shafique, L. Bauer, and J. Henkel, "Selective Instruction Set Muting for Energy-Aware Adaptive Processors," in *Proc. Int. Conf. on Comput.-Aided Design (ICCAD)*, 2010, pp. 353–360.
[5] N. Metha and A. DeHon, "Variation and Aging Tolerance in FPGAs," in *Low-Power Variation-Tolerant Design in Nanometer Silicon*. Springer Science+Business, 2011.
[6] J. McPherson, "Reliability Challenges for 45nm and Beyond," in *Proc. 43rd Design Automation Conf. (DAC)*, 2006, pp. 176–181.
[7] S. Bhunia and R. Rao, "Guest Editors' Introduction: Managing Uncertainty through Postfabrication Calibration and Repair," *IEEE Design & Test of Comput. (D&ToC)*, vol. 27, no. 6, pp. 4–5, 2010.
[8] M. S. Abdelfattah *et al.*, "Transparent Structural Online Test for Reconfigurable Systems," in *Proc. 18th IEEE Int. On-Line Testing Symp. (IOLTS)*, 2012, pp. 37–42.
[9] L. Bauer *et al.*, "OTERA: Online Test Strategies for Reliable Reconfigurable Architectures," in *Proc. NASA/ESA Conf. on Adaptive Hardware and Syst. (AHS)*, 2012, pp. 38–45.
[10] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*. Springer, 2007.
[11] H. P. Huynh and T. Mitra, "Runtime Adaptive Extensible Embedded Processors – A Survey," in *Proc. 9th Int. Workshop on Emb. Comput. Syst.: Architectures, Modeling, and Simulation (SAMOS)*, 2009, pp. 215–225.
[12] E. Lübbers and M. Platzner, "ReconOS: Multithreaded programming for reconfigurable computers," *ACM Trans. in Emb. Computing Syst. (TECS)*, vol. 9, no. 1, pp. 8:1–8:33, 2009.
[13] S. Vassiliadis *et al.*, "The MOLEN polymorphic processor," *IEEE Trans. on Comput. (TC)*, vol. 53, no. 11, pp. 1363–1375, 2004.
[14] L. Bauer, M. Shafique, and J. Henkel, "Concepts, Architectures, and Run-time Systems for Efficient and Adaptive Reconfigurable Processors," in *Proc. NASA/ESA Conf. on Adaptive Hardware and Syst. (AHS)*, 2011, pp. 80–87.
[15] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, pp. 659–681, 2006.
[16] M. Psarakis *et al.*, "Microprocessor software-based self-testing," *IEEE Design&Test of Comp. (D&ToC)*, vol. 27, no. 3, pp. 4–19, 2010.
[17] C. Stroud *et al.*, "Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)," in *Proc. 14th IEEE VLSI Test Symp. (VTS)*, 1996, pp. 387–392.
[18] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *Xilinx Application Notes, XAPP216 (v1. 0)*, 2000.
[19] A. Avizienis *et al.*, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. on Dependable and Secure Computing (TDSC)*, vol. 1, no. 1, pp. 11–33, 2004.
[20] C. Stroud *et al.*, "Built-in self-test of FPGA interconnect," in *Proc. IEEE Int. Test Conf. (ITC)*, 1998, pp. 404–411.
[21] E. Chmelar, "FPGA interconnect delay fault testing," in *Proc. IEEE Int. Test Conf. (ITC)*, 2003, pp. 1239–1247.
[22] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Elsevier, 2001.
[23] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerant methods for reliability in FPGAs," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2008, pp. 415–420.
[24] F. L. Kastensmidt and R. Reis, *Fault-Tolerance Techniques for SRAM-Based FPGAs*, ser. Frontiers in Electronic Testing. Springer, 2010.
[25] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (FPGA)*, 2010, pp. 249–258.
[26] S. Velusamy *et al.*, "Monitoring temperature in FPGA based SoCs," in *Proc. 23rd IEEE Int. Conf. on Comput. Design (ICCD)*, 2005, pp. 634–637.
[27] M. Agarwal *et al.*, "Circuit Failure Prediction and Its Application to Transistor Aging," in *Proc. 25th IEEE VLSI Test Symp. (VTS)*, 2007, pp. 277–286.
[28] S. Durand and C. Piguet, "FPGAs with self-repair capabilities," in *Proc. ACM Int. Workshop on Field Programmable Gate Arrays (FPGA)*, 1994, pp. 1–6.
[29] J. Emmert *et al.*, "Dynamic fault tolerance in FPGAs via partial reconfiguration," in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2000, pp. 165–174.
[30] S. Mitra *et al.*, "Reconfigurable architecture for autonomous self-repair," *IEEE Design & Test of Comput. (D&ToC)*, vol. 21, no. 3, pp. 228–240, 2004.
[31] A. Jacobs, A. George, and G. Cieslewski, "Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space," in *Proc. 19th Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2009, pp. 199–204.
[32] A. Krasniewski, "Application-Dependent Testing of FPGA Delay Faults," in *Proc. 25th EUROMICRO Conf.*, vol. 1, 1999, pp. 260–267.
[33] M. Tahoori, "Application-Dependent Testing of FPGAs," *IEEE Trans. on Very Large Scale Integ. (VLSI) Syst.*, vol. 14, no. 9, pp. 1024–1033, 2006.
[34] 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.
[35] A. Van De Goor, "Using march tests to test SRAMs," *IEEE Design & Test of Comput. (D&ToC)*, vol. 10, no. 1, pp. 8–14, 1993.
[36] K. Radecka, J. Rajski, and J. Tyszser, "Arithmetic Built-In Self-Test for DSP Cores," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. (TCAD)*, vol. 16, no. 11, pp. 1358–1369, 1997.
[37] W. K. Huang *et al.*, "Testing configurable LUT-based FPGA's," *IEEE Trans. on Very Large Scale Integ. (VLSI) Syst.*, vol. 6, no. 2, pp. 276–283, 1998.
[38] M. Renovell *et al.*, "Testing the interconnect of RAM-based FPGAs," *IEEE Design & Test of Comput. (D&ToC)*, vol. 15, no. 1, pp. 45–50, 1998.
[39] A. Friedman, "Easily Testable Iterative Systems," *IEEE Trans. on Comput. (TC)*, vol. C-22, no. 12, pp. 1061–1064, 1973.
[40] M. Renovell, "SRAM-based FPGAs: a structural test approach," in *Proc. 11th Symp. on Integr. Circuit Des. (SBCCI)*, 1998, pp. 67–72.
[41] P. Sundararajan, S. Mcmillan, and S. A. Guccione, "Testing FPGA Devices Using JBits," in *Proc. Military and Aerospace Applications of Programmable Devices and Technologies Conf. (MAPLD)*, 2001.
[42] C. Stroud *et al.*, "Built-in self-test of FPGA interconnect," in *Proc. IEEE Int. Test Conf. (ITC)*, 1998, pp. 404–411.
[43] X. Sun *et al.*, "Novel technique for built-in self-test of FPGA interconnects," in *Proc. IEEE Int. Test Conf. (ITC)*, 2000, pp. 795–803.
[44] M. Tahoori and S. Mitra, "Application-independent testing of FPGA interconnects," *IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. (TCAD)*, vol. 24, no. 11, pp. 1774–1783, 2005.
[45] M. B. Tahoori, "Using satisfiability in application-dependent testing of FPGA interconnects," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 678–681.
[46] H. Almurib, T. Kumar, and F. Lombardi, "A single-configuration method for application-dependent testing of SRAM-based FPGA interconnects," in *20th Asian Test Symp. (ATS)*, 2011, pp. 444 –450.
[47] V. Verma, S. Dutt, and V. Suthar, "Efficient On-line Testing of FPGAs with Provable Diagnosabilities," in *Proc. 41th Design Automation Conf. (DAC)*, 2004, pp. 498–503.
[48] D. Milton, S. Dhingra, and C. E. Stroud, "Embedded Processor Based Built-In Self-Test and Diagnosis of Logic and Memory Resources in FPGAs," in *Proc. Int. Conf. on Emb. Syst. and Applications (ESA)*, 2006, pp. 87–93.
[49] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks," *IEEE Trans. on Very Large Scale Integ. (VLSI) Syst.*, vol. 15, no. 2, pp. 216–226, 2007.
[50] B. F. Dutton and C. E. Stroud, "Soft Core Embedded Processor Based Built-In Self-Test of FPGAs," in *Proc. 24th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Syst. (DFT)*, 2009, pp. 29–37.
[51] M. Abramovici, C. E. Stroud, and J. M. Emmert, "Online BIST and BIST-Based Diagnosis of FPGA Logic Blocks," *IEEE Trans. on Very Large Scale Integ. (VLSI) Syst.*, vol. 12, no. 12, pp. 1284–1294, 2004.
[52] M. Abramovici *et al.*, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications," in *Proc. IEEE Int. Test Conf. (ITC)*, 1999, pp. 973–982.
[53] L. Wang, C. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann, 2006.
[54] Y. Li, S. Makar, and S. Mitra, "CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 885–890.
[55] Y. Li, O. Mutlu, and S. Mitra, "Operating System Scheduling for Efficient Online Self-Test in Robust Syst." in *Proc. ACM Int. Conf. on Comput.-Aided Design (ICCAD)*, 2009, pp. 201–208.
[56] H. Inoue, Y. Li, and S. Mitra, "VAST: Virtualization-Assisted Concurrent Autonomous Self-Test," in *Proc. IEEE Int. Test Conf. (ITC)*, 2008, pp. 1–10.
[57] W. H. Kautz, "Testing for faults in combinational cellular logic arrays," in *Proc. 8th Symp. on Switching and Automata Theory (SWAT)*, 1967, pp. 161–174.
[58] M. Psarakis, D. Gizopoulos, and A. Paschalis, "Test Generation and Fault Simulation for Cell Fault Model using Stuck-at Fault Model based Test Tools," *J. of Electronic Testing*, vol. 13, no. 3, pp. 315–319, 1998.
[59] S. Makar and E. McCluskey, "Functional tests for scan chain latches," in *Proc. IEEE Int. Test Conf. (ITC)*, 1995, pp. 606–615.
[60] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases," in *Proc. Int. Workshop on Reconf. Comm.-centric Syst.-on-Chip (ReCoSoC)*, 2011, pp. 1–8.
[61] A. Grudnitsky, L. Bauer, and J. Henkel, "Partial Online-Synthesis for Mixed-Grained Reconfigurable Architectures," in *Proc. Design, Automation and Test in Europe (DATE)*, 2012, pp. 1555–1560.
[62] L. Bauer, M. Shafique, and J. Henkel, "A Computation- and Communication- Infrastructure for Modular Special Instructions in a Dynamically Reconfigurable Processor," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2008, pp. 203–208.
[63] Aeroflex Gaisler, "Homepage of the Leon Processor," http://www.gaisler.com/leonmain.html, accessed at August 15, 2012.
[64] M. Shafique, L. Bauer, and J. Henkel, "Optimizing the H.264/AVC Video Encoder Application Structure for Reconfigurable and Application-Specific Platforms," *J. of Signal Processing Syst. (JSPS)*, vol. 60, no. 2, pp. 183–210, 2009.

Lars Bauer received his M.Sc. and Ph.D. in CS from the University of Karlsruhe, Germany in 2004 and 2009, respectively. He is currently a research assistant, lecturer, and group leader at the Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT). He received the EDAA Outstanding Dissertations Award, the FZI Outstanding Dissertation Award, the AHS'11 best paper award, and the DATE'08 best paper award for his work on adaptive reconfigurable processors. He is a member of the IEEE.

Claus Braun received the Diploma degree in computer science from the University of Tübingen, Germany, in 2008. He joined the Institute for Computer Architecture and Computer Engineering, University of Stuttgart, in 2008. His current research interests include fault tolerance and parallel computer architectures. He is a member of the IEEE.

Michael E. Imhof received the Diploma degree in computer science from the University of Stuttgart, Germany, in 2005. He joined the Institute for Computer Architecture and Computer Engineering, University of Stuttgart, in 2006. His current research interests include fault tolerance and variation-aware test. He is a student member of the IEEE and the IEEE Computer Society.

Michael A. Kochte received the Diploma degree in computer science from the University of Stuttgart, Germany, in 2005. He joined the Institute for Computer Architecture and Computer Engineering, University of Stuttgart, in 2007. His current research interests include test generation, fault simulation, and fault tolerance. He is a student member of the IEEE and the IEEE Computer Society.

Eric Schneider received the Diploma degree in computer science from the University of Stuttgart, Germany, in 2012. He joined the Institute for Computer Architecture and Computer Engineering, University of Stuttgart, in 2012. His current research interests include circuit simulation, test and diagnosis.

Hongyan Zhang received the M.Sc. in Electrical Engineering and Information Technologies from the Karlsruhe Institute of Technology in 2011. He joined the Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT) in 2011. His research interests include fault tolerant and reliable runtime reconfigurable architectures. He is a student member of the IEEE.

Jörg Henkel is with the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, where he is directing the Chair for Embedded Systems (CES). Prior to that, he was with NEC Laboratories, Princeton, NJ. He holds ten US patents. His current research is focused on design and architectures for embedded systems with focus on low power and reliability. Prof. Henkel was the recipient of the 2008 DATE Best Paper Award, the 2009 IEEE/ACM William J. McCalla ICCAD Best Paper Award and the Codes+ISSS 2011 Best paper Award. He is the Chairman of the IEEE Computer Society, Germany Section, and the Editor-in-Chief of the ACM Transactions on Embedded Computing Systems. He is an initiator and the spokesperson of the national priority program "Dependable Embedded Systems" of the German Science Foundation. He is the General Chair of ICCAD 2013. He is a member of the IEEE and the IEEE Computer Society.

Hans-Joachim Wunderlich received a Diploma in Mathematics from the University of Freiburg in 1981 and the Dr. rer. nat. (Ph.D.) from the University of Karlsruhe in 1986. Since 1991 he has been a full Professor and since 2002 he has been the director of the Institute of Computer Architecture and Computer Engineering at the University of Stuttgart. He is editor of various international journals and program committee member of a variety of IEEE conferences on design and test of electronic systems. Hans-Joachim Wunderlich has published 11 books and book chapters and more than 200 reviewed scientific papers in journals and conferences. His research interests include test, reliability and fault tolerance of microelectronic systems. Hans-Joachim Wunderlich is a fellow of IEEE. He is a fellow of the IEEE.