OTERA: Online Test Strategies for Reliable Reconfigurable Architectures

— Invited Paper for the AHS-2012 Special Session "Dependability by reconfigurable hardware" —

Lars Bauer[†], Claus Braun[*], Michael E. Imhof[*], Michael A. Kochte[*]
Hongyan Zhang[†], Hans-Joachim Wunderlich[*] and Jörg Henkel[†]

[*]*ITI, University of Stuttgart, Pfaffenwaldring 47, D-70569, Stuttgart, Germany*
Email: {braun, imhof, kochte}@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de
[†]*Karlsruhe Institute of Technology, Haid-und-Neu-Str. 7, D-76131, Karlsruhe, Germany*
Email: {lars.bauer, hongyan.zhang, henkel}@kit.edu

***Abstract*—FPGA-based reconfigurable systems allow the online adaptation to dynamically changing runtime requirements. However, the reliability of FPGAs, which are manufactured in latest technologies, is threatened not only by soft errors, but also by aging effects and latent defects. To ensure reliable reconfiguration, it is mandatory to guarantee the correct operation of the underlying reconfigurable fabric. This can be achieved by periodic or on-demand online testing.**

The *OTERA* project develops and evaluates components and strategies for reconfigurable systems that feature reliable reconfiguration. The research focus ranges from structural online tests for the FPGA infrastructure and functional online tests for the configured functionality up to the resource management and test scheduling. This paper gives an overview of the project tasks and presents first results.

I. Introduction

Reconfigurable architectures are continuously gaining importance, since they are particularly interesting for a broad field of applications. Today, especially systems based on *Field-Programmable Gate Arrays* (FPGAs) can be found from high-end HPC 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 with latest generations of 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].

Since these FPGAs are typically manufactured in latest semiconductor process technologies (e.g. 28nm for Xilinx Virtex-7 and Altera Stratix V), they must not only increasingly cope with soft errors, but also with aging effects, variations, and latent defects in the reconfigurable fabric [5–7]. These reliability threats are intensified by long mission times and harsh environmental conditions (e.g. temperature). As a consequence, classic production and burn-in tests are no longer sufficient to guarantee reliable reconfigurable systems throughout the whole product lifecycle. Thus, for example, the reliable operation of the FPGA's reconfigurable fabric has to be constantly monitored and verified by thorough online testing. This is especially challenging for runtime reconfigurable systems, because they change their hardware configuration dynamically as part of their normal operation.

The required components and strategies for such reliable runtime reconfigurable systems are explored and developed within the scope of the *OTERA* project (Online Test Strategies for Reliable Reconfigurable Architectures). A *structural pre-configuration online test* (*PRET*) is the first basic key component, designed to independently test the hardware structure of specific regions of the reconfigurable fabric, so called *containers*, periodically or on-demand. The PRET consists of tests for all major FPGA structures, such as the CLBs or the interconnects. In addition to the PRET, the instantiated *accelerators* (reconfigured to containers) have to be functionally tested after each successful reconfiguration. This *functional post-reconfiguration online test* (*PORT*) is the second basic key component, since it ensures that the configured accelerators correctly deliver the expected functionality and timing.

Although of fundamental importance, these two components only form the basis for a reliable reconfigurable system. They have to be supplemented by dedicated components for online monitoring, prediction, and control to allow the system a pro-active adaption to changing environmental conditions. Besides the selection, evaluation, and optimization of suitable solutions for each of these components, the overall system integration is a very challenging task. In contrast to previous approaches, these components may not be considered and designed isolated from each other. They must be inherent parts of the system-level design process to allow a seamless integration into a runtime reconfigurable system with minimum impact on the system's performance.

The paper is structured as follows: Section II gives an overview of the related work in the fields of reconfigurable architectures, dependable systems and test. Section III details the structure of a system with reliable reconfiguration through online testing. A corresponding case study is presented in section IV, followed by an outlook in section V and the conclusion.

II. Related Work

A. Reconfigurable Architectures

Different kinds of reconfigurable architectures evolved in the last years [8]. 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 [9], whereas other architectures reconfigure application-specific accelerators that are invoked by the application that executes on a processor [10, 11]. Despite these advantages of runtime reconfigurable architectures their management is challenging, especially when it comes to testing.

The regions of the reconfigurable fabric that are never reconfigured during runtime can be tested with established functional approaches. However, the regions that are reconfigured during runtime need a different testing approach (focus of this paper) as their configurations are not know statically. State-of-the-art reconfigurable systems focus on increasing the performance [9–12] or on saving energy [4]. However, they neither consider testing the runtime reconfigurable fabric as a goal or constraint, nor do they propose testing methods and their integration into a runtime reconfigurable system, as done in this work.

Jacobs et al. [13] proposed a reconfigurable fault tolerance framework that allows dynamically reconfiguring between three different reliability/performance trade-offs. However, they do not consider structural faults of the reconfigurable fabric, adapting the test strategy automatically, or provide graceful degradation methods for reconfigurable architectures.

B. Dependable Systems

Dependability, among other system properties, comprises both reliability and availability [14]. The focus of the OTERA project, is to increase reliability by ensuring that the used reconfigurable fabric is fault-free, and to increase availability by fast online repair and use of partially faulty resources.

For the test of logic resources of the fabric, the stuck-at fault model is most commonly used. Dedicated fault models exist for interconnects [15]. Test generation for delay faults for the fabric has been introduced in [16]. 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 the permanent 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 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 [17] have been optimized for FPGAs [18] and extended by FPGA specific techniques like scrubbing [19].

Sensors in the system allow to monitor the circuit behavior and temperature [20, 21]. 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 [22–25] to ensure that mission logic does not use faulty hardware blocks of the fabric.

First approaches [26] bring together system self-awareness by monitoring application behavior and system adaption to optimize runtime performance. While the paradigm of online monitoring and self-adaptation is indeed embodied in this recent work, it is limited to the application performance. With the system-wide integration of monitors, tests and repair, the OTERA project targets dependable systems which autonomously apply fault-tolerance techniques as required while imposing as little impact on application performance as possible. The tight integration of fault tolerance techniques into reconfigurable systems transparent to the application is a novel challenge.

C. Test Methods for Reconfigurable Architectures

The thorough test of the FPGA fabric requires fundamental structural knowledge and the test generation is typically tailored to a specific FPGA architecture. Application dependent tests [27] target the faults relevant for a known fixed application configuration of the FPGA. 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 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 test configurations may range from a few up to a few hundreds if programmable interconnect structures are completely included in the tests [28].

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 [29] are used, for arithmetic structures like multipliers or DSPs, functional tests with high structural fault coverage are possible [30].

For an online in-field test of FPGAs, external equipment for test pattern generation (TPG) and output response analysis (ORA) [31, 32] is not available. Internal testing approaches based on built-in self-test (BIST) principles include TPGs and ORAs in the unit under test.

The highly regular nature of FPGAs allows to configure the structures under test into an iterative logic array (ILA [33]) such that the resulting test time is constant and independent of the array size (C-testability [34]). 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 [35, 36]. 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 [37–39]. Due to the complexity of the interconnect configuration circuitry, a very high number of TCs is required.

Using partial dynamic reconfiguration of FPGAs, the test reconfiguration can be conducted by an external [40] or embedded processor during runtime [41–43]. The access to an internal high-bandwidth configuration port significantly reduces the time required for reconfiguration.

In addition to testing, the homogeneous structure of an FPGA allows the efficient diagnosis of faulty components. High resolution is achieved by failure data analysis and additional dedicated TCs to distinguish faults with equal signatures [44].

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 [45] partitions the FPGA into multiple regions which can be either used functionally or tested by an online BIST scheme controlled by an external module.

In the OTERA project, the structural PRET is implemented by combining the state-of-the-art in FPGA testing to achieve high fault coverage in the reconfigurable fabric. The reconfiguration process for PRET is tightly integrated into the functional system scheduling. The existing reconfiguration features in the system are reused for PRET.

III. Reliable Reconfiguration by Online Testing

The correct operation of mission logic, instantiated into a reconfigurable container, mandates that both the underlying reconfigurable fabric is free of defects, and the reconfiguration process is conducted without error. To achieve this we consider systems where containers have a static size, a static position, can be isolated from the system for testing, and provide an access for test stimuli. After presenting the system architecture that we use for evaluation, this section details how a structural pre-configuration online test (PRET) and a post-reconfiguration online test (PORT) are integrated into the system to achieve this goal.

A. System Architecture

Fig. 1 shows the system architecture of a typical reconfigurable processor that connects runtime reconfigurable containers (right side) to a non-reconfigurable core processor (left side). Even though the presented PRET and PORT approaches are conceptually orthogonal to the particular system architecture, we describe them with respect to this system architecture as it is also used for evaluation.

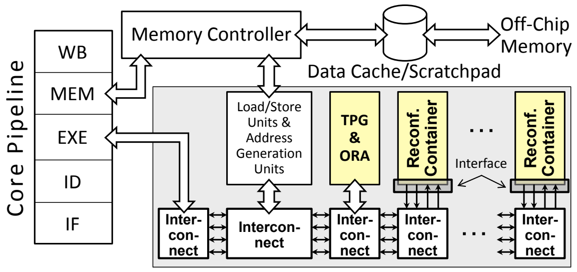

Figure 1. Overview of the reconfigurable processor architecture consisting of the reconfigurable containers, the core pipeline, and the memory hierarchy [46]

The reconfigurable containers are connected to each other using an interconnect infrastructure that consists of four bi-directional segmented buses. This interconnect infrastructure also connects the containers to the core pipeline (providing access to the register file) and to the system memory hierarchy. The core pipeline is extended by Special Instructions (SI), i.e. assembler instructions that perform application-specific computations such as transformations, filters, encryption etc. One or multiple accelerators need to be reconfigured to the containers to implement an SI. A runtime system decides, which SIs are reconfigured and when they are reconfigured. More details about this architecture and its runtime system (not required for the content of this paper) are available in [11].

B. PRET: Pre-Configuration Online Test

To ensure 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 which targets a set of faults in the fabric.

1) Fault Model: The stuck-at fault model (SAF) is widely adopted in the literature for FPGA testing [47]. For complex FPGA sub-components such as 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 and interconnects in which the 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.

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 (CFM) [48, 49] is applied. The 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, functional memory faults [29] are targeted, i.e. address decoder faults (AF), stuck-at faults, transition faults (TF), coupling faults (CF), and data retention faults (DRF).

Sequential elements: CLBs contain 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).

2) TC Generation for CLB Elements: The complexity of a CLB requires multiple TCs to exercise all components in a test session. 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 deterministically designed to guarantee full fault coverage. Testing is applied on a container. Each TC targets a specific set of faults in the subcomponents in each CLB and consists of two main parts:

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

In addition, test infrastructure is required to generate the stimuli (TPG) and evaluate the responses (ORA). The TPG and ORA may differ between TCs and they are external to the container under test as shown in Fig. 1.

The regular structure of the reconfigurable fabric allows the efficient and scalable test of large containers by connecting the CLBs into a C-testable array as exemplified in Fig. 2. The resulting test time for the array is very low and independent of the array size.

A TPG feeds the test patterns at the start of each array and the responses are aggregated and evaluated at the end of the array using an ORA. In FPGA testing, typically all possible stimuli are applied to the components under test. For the TPG, a counter or linear feedback shift register is used. 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 included in each CLB.

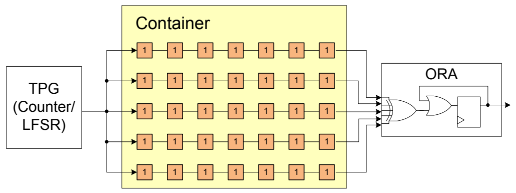

Figure 2. Container configured into a C-testable array with external test pattern generator and output response analysis [46]

For the LUT in function mode, all cell faults are targeted. This is achieved by two TCs computing the XOR and XNOR of the inputs, 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 for by applying standard scan chain test patterns [50]. 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++ [29] algorithm to ensure coverage of all SAFs, AFs and TFs. The test patterns are generated at the global TPG.

The multiplexers in the CLBs are tested by applying all possible configurations to exercise all select combinations. The data path is tested for SAFs 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 SAFs in the carry chain efficiently, the elements are configured into pipelined C-testable arrays. Two TCs are required to fully test the carry chain.

Testing flip-flops in CLBs is identical to testing the LUT in SR 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.

3) TC Generation for Container: 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 two steps, 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, [51]) file.

C. PORT: Post-Reconfiguration Test

The structural integrity of the fabric can be assessed by execution of PRET. This however does not guarantee the correct configuration and integration of mission logic in a container. Errors may occur during loading the configuration bitstream, e.g. due to faults in the configuration logic, or transient events such as crosstalk, radiation or other sources of transient events. As a consequence, the configured function of the targeted container may be wrong, or the configuration of other containers may be adversely altered.

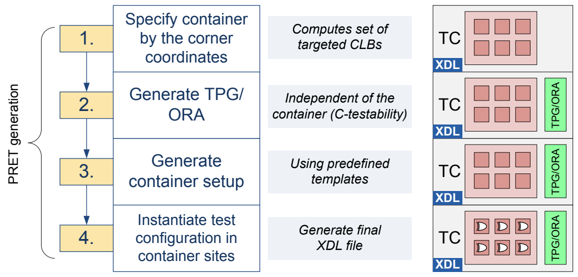
PRET generation
1.
Specify container by the corner coordinates
Computes set of targeted CLBs
2.
Generate TPG/ ORA
Independent of the container (C-testability)
3.
Generate container setup
Using predefined templates
4.
Instantiate test configuration in container sites
Generate final XDL file
TC
XDL
TPG/ORA

Figure 3. XDL file generation flow for CLB TCs [46]

A functional test of the configured logic is able to check whether the reconfiguration process completed without errors and the mission logic interfaces correctly to the surrounding system. This shall be implemented by PORT which is a functional test tailored to the mission logic of the containers. PORT is executed after reconfiguration of a container, and periodically during operation to uncover corruption of configuration bits due to soft errors.

A PORT can be conducted by functional patterns, random patterns and 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, POST does not reconfigure the tested structures and thus exhibits a much lower performance impact on the application.

D. System Integration and Scheduling

We now explain how the developed PRET and PORT tests are integrated into the system. Fig. 4 shows an excerpt of the reconfigurable fabric with three containers and the components involved in testing them. In the first step (Fig. 4a), the runtime system decides that an accelerator shall be reconfigured into a particular container (to implement a Special Instruction), which triggers the demand to test the container first. As performing an exhaustive PRET test would delay the accelerator reconfiguration significantly, only an incremental PRET is performed. This means that not all test configurations (TCs) are applied to the container, but only some of them. The runtime system decides how many TCs should be applied before reconfiguring the accelerator (potentially none), depending on the test history. For each container, the runtime system keeps track of which TCs were applied to this container in the past and how much time passed by since the last exhaustive PRET test of the container. Depending on this test history, it activates the PRET component that reconfigures the selected TCs into the container and uses the test pattern generator and the output response analyzer to exercises the reconfigurable

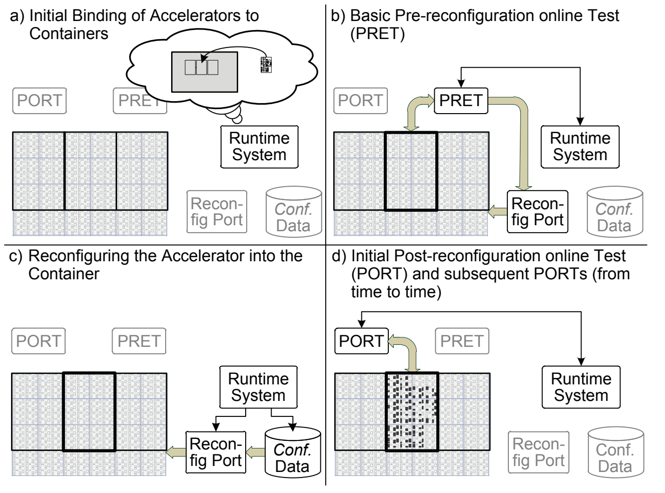
a) Initial Binding of Accelerators to Containers
PORT
PRET
Runtime System
Recon-fig Port
Conf. Data
b) Basic Pre-reconfiguration online Test (PRET)
PORT
PRET
Runtime System
Recon-fig Port
Conf. Data
c) Reconfiguring the Accelerator into the Container
PORT
PRET
Runtime System
Recon-fig Port
Conf. Data
d) Initial Post-reconfiguration online Test (PORT) and subsequent PORTs (from time to time)
PORT
PRET
Runtime System
Recon-fig Port
Conf. Data

Figure 4. Example for typical runtime flow with PRET and PORT

fabric (Fig. 4b).

If no structural defect is found, the runtime system reconfigures the desired accelerator into the container (Fig. 4c). Before the accelerator can be used to implement a Special Instruction (SI), the runtime system triggers PORT to test whether the reconfigured accelerator operates as expected. Additionally, accelerators in other containers that are used to implement the SI are tested as well to ensure that they were not affected by the reconfiguration. As PORT operates significantly faster than PRET, it can also be applied during normal operation (i.e. not only after reconfiguring a container) and the runtime system schedules subsequent PORTs regularly.

IV. Case Study

To demonstrate the idea of PRET presented in section III-B, we integrated the PRET scheme into the reconfigurable architecture introduced in section III-A and investigated the PRET overhead and the test effectiveness of it.

A. Experimental Setup

The reconfigurable architecture introduced in section III-A forms the hardware platform for the experimental evaluation. A Leon-3 processor is used as core pipeline with 10 attached containers (see Fig. 1). In our prototype, each container consists of 4x20 CLBs (other sizes are also possible). The actual hardware prototyping is performed on a 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 a reconfiguration bandwidth of 66 MB/s. Although the internal configuration access port (ICAP) of the Xilinx FPGA can operate with up to 400 MB/s, the actual configuration bandwidth depends on the provisioning speed of the partial bitstream, which is stored in off-chip DRAM.

For the purpose of case study, a sophisticated H.264 video encoder encoding 500 frames is chosen as application. The encoder is a challenging application since it frequently reconfigures 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 (SIs) and each SI demands multiple accelerators, hence the system reconfigures the containers accordingly. In total, 9 SIs are implemented for the encoder by using combinations of 10 different types of accelerators. The implementation of the H.264 encoder on the reconfigurable system leads to a speedup of more than 20x in comparison to the Leon-3 without SIs.

For the integration of the proposed PRET scheme, the TPG and the ORA have been attached to the containers as shown in Fig. 1. The runtime system, which controls the reconfigurations, was extended to schedule test configurations in regular intervals. The reconfiguration process for the test configurations is identical to the reconfiguration of the accelerators.

B. Test Overhead and Results

Realizing the benefits of PRET leads to overhead in space and time that are discussed in the following. Additional hardware blocks for TPG and ORA are required. They only introduce minimal area overhead as shown in section IV-B1 and have no affect on the system clock frequency. Tests are regularly scheduled to containers like functional workloads. The test execution may delay the configuration of accelerators or consume certain communication resources, and thus have a negative impact on the performance of the application as presented in section IV-B2.

1) Test Configurations: A full test session consists of multiple test configurations (TCs), each of which activates and tests a subset of the components in the CLBs of a container as described in section III-B. Altogether nine TCs are required to test all subcomponents in the CLBs. Partial bitstreams for these TCs are generated and stored in memory so that the runtime system of the reconfigurable architecture can fetch and configure them into the FPGA to test a container.

Table I provides an overview on the nine TCs. Column one shows the configuration number. Column two shortly describes the portion of each CLB being 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 (‘test length’ in Table I). Typically, the latter ranges from a few cycles up to a few hundred cycles while the former dominates the test time with tens of thousands of cycles. The configuration time is directly proportional to the size of the configuration data (partial bitstreams) and the reconfiguration bandwidth. As shown in Table I, the bitstream size of each TC varies from 22.3 KB to 28.6 KB, which corresponds to a configuration time between 0.33 ms and 0.42 ms at 66 MB/s configuration bandwidth, i.e. between 33 and 42 thousand cycles at 100 MHz system frequency.

TC	Tested CLB subcomponents	PRET overh. [CLBs]	Bitstr. size [KB]	Test length [Cycles]
1	LUT configured as XOR, connected to FF	2	24.0	64
2	LUT configured as XNOR, connected to FF	2	24.0	64
3	Carry MUX, interleaved with MUX and latch	1	28.6	6
4	Carry MUX, interleaved with MUX and latch	1	26.1	6
5	Carry XOR, interleaved with MUX and FF	1	28.0	6
6	Carry XOR, interleaved with MUX and FF	1	28.2	6
7	Carry-in/-out tested with multiplexed scan chain	1	27.1	6
8	LUT configured as SR with slice MUX	1	22.9	6
9	LUT configured as RAM with slice output	7	22.3	320

Table I
CLB TEST CONFIGURATIONS: OVERHEAD, BITSTREAM SIZE AND LENGTH [46]

2) *Concurrent Test Scheduling:* When a container is not being used and the configuration port is available, TCs can be configured into that container and tests can be performed on it without affecting the system performance. In the general case, tests are scheduled like functional workload and compete with accelerator reconfigurations (configuration port) and SI executions (interconnect buses), which may have a negative impact on the performance of the system.

We use a regular approach that schedules one TC after a certain number of Accelerator Configurations (ACs) to demonstrate that the test has negligible effects on the performance while still maintaining high test effectiveness. For instance, one TC is scheduled before every AC or before every 2nd AC, 3rd AC, etc. This corresponds to one AC per TC or 2 ACs per TC etc. Fig. 5 shows the simulation results for the performance loss under different test frequencies (from 1 AC/TC to 4 ACs/TC), depending on the number of reconfigurable containers that are available (i.e. the size of the reconfigurable fabric). Using a lower test frequency (e.g. 4 ACs/TC) reduces the overhead. Additionally, reconfigurable architectures with more containers have a lower overhead as more containers are still operational during the test. For instance, in a reconfigurable architecture with 5 containers, only 4 containers can be used to implement SIs during a PRET test, whereas in an architecture with 14 containers, still 13 containers can be used for acceleration.

Fig. 6 shows the average test period. For example, for a reconfigurable architecture with 10 containers and a test frequency of 3 AC/TC, each container is completely tested every 4.3 seconds while the performance loss introduced by PRET is only 0.33%. To give more insight into the test effectiveness, Fig. 7 shows the number of completed test runs (1 complete test run of a container requires 9 TCs) performed in each container during the application execution time for this architectures.

It is noticeable that in Fig. 7 some containers are tested more than twice as often as other containers. This directly depends on the number of accelerator reconfigurations of these containers. Some accelerators are reconfigured rather seldom, which directly affects the test frequency of the corresponding containers. Generally, reconfigurable architectures with a large number of containers perform less reconfigurations (in the extreme case, the reconfigurable fabric is large enough to provide all required accelerators at the same time). This is the reason why the test period in Fig. 6 increases significantly for systems with more than 11 containers. In such systems, some containers are reconfigured rather seldom, thus it takes longer until the entire system is tested. However, the longest observed test period of 14 seconds is still orders of magnitudes shorter than the process of aging which impacts circuit parameters over the duration of years.

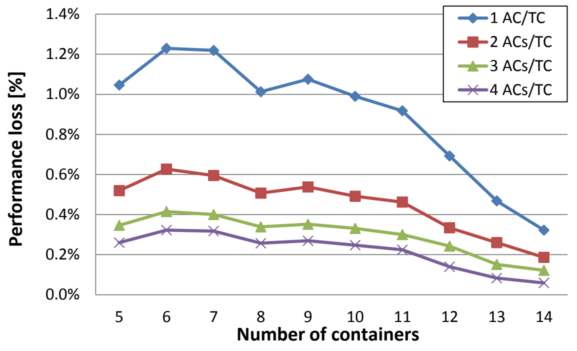

Figure 5. Performance loss of the video encoder application under different test frequencies and number of containers

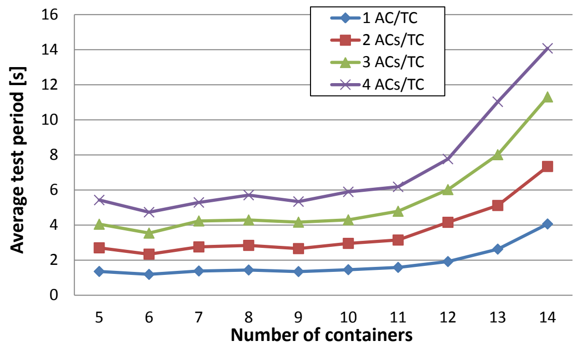

Figure 6. Average test period under different test frequencies and number of containers

V. From Reliable Reconfiguration to Dependable Systems

Reliable Reconfiguration ensures the integrity of the configured function and the underlying fabric. The dependability of a system is affected during runtime by environmental effects such as a changing environment and transient faults. A dependable system therefore needs to be self-aware of its environment in order to adaptively choose appropriate countermeasures (Fig. 8). Concurrent error detection (CED), containment and recovery contribute in maintaining correct operation under transient faults. Diagnosing the affected system parts and classifying the root cause as transient or permanent enables the system to take appropriate actions, such as containment/recovery or graceful degradation, actively due to the monitored effects or pro-actively based on a prediction.

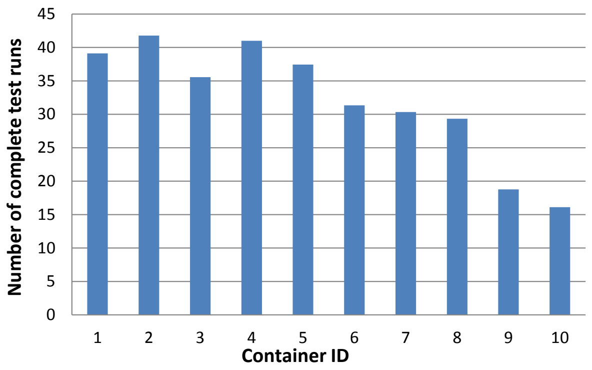

Figure 7. Number of complete test runs in individual containers with a test frequency of 3 ACs/TC

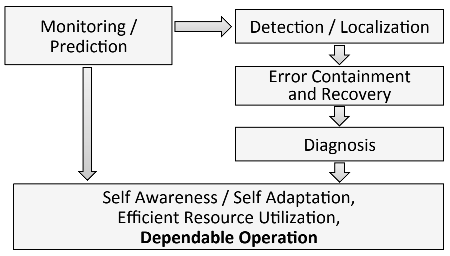
Monitoring / Prediction
Detection / Localization
Error Containment and Recovery
Diagnosis
Self Awareness / Self Adaptation, Efficient Resource Utilization, Dependable Operation

Figure 8. Components for dependable systems

A. Concurrent Error Detection (CED), Containment

The system model is extended towards transient faults which allows the introduction of concurrent test and checking methods. Careful evaluation of the detection capabilities and costs (e.g. area, delay, power) of different solutions allows to select appropriate methods specifically for each container. The runtime system trades-off test intensity, system stress and environmental conditions and selects the appropriate mechanisms (e.g. error correction, checkpoints, recomputation) to handle and contain errors.

Extending reliable reconfiguration with monitoring and preventive system adaption at runtime enables efficient concurrent error detection and containment. Monitors of non-functional observables (e.g. frequency, temperature) get incorporated and placed where required, providing self awareness to the system. The runtime system determines their position to enable preventive system adaptation. A pro-active reaction (based on online monitor prediction) increases the system dependability.

B. Diagnosis

Diagnosing observed errors allows to classify their cause as transient or permanent. In case of transient faults, error containment prevents negative dependability influence. In case of permanent faults, the impact on performance and containment capabilities is significant. If adaption of non-functional parameters such as workload of a container is not possible or not desired, the faulty component needs to be disabled or isolated. In order to minimize the impact on the system (e.g. if only a fraction of a container is faulty), the online diagnosis needs to locate the component in a fine-grained way.

C. Graceful degradation in case of structural faults

When one of the testing mechanisms (e.g. PRET, PORT, CED) detects a fault in a container and the fault turns out to be permanent, then an exhaustive PRET together with a diagnosis is triggered, as shown in Fig. 9a. The diagnosis analyzes the result of PRET to determine the location of the fault. In Fig. 9a this is indicated by crossing out a particular region of the container that was identified as faulty.

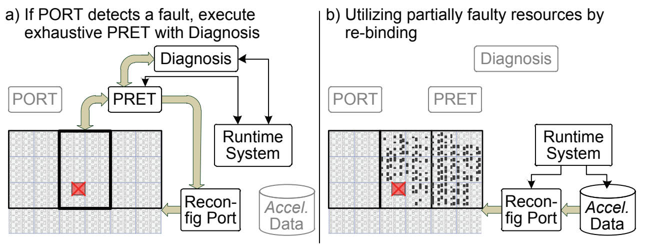
a) If PORT detects a fault, execute exhaustive PRET with Diagnosis
Diagnosis
PORT
PRET
Runtime System
Recon-fig Port
Accel. Data
b) Utilizing partially faulty resources by re-binding
Diagnosis
PORT
PRET
Runtime System
Recon-fig Port
Accel. Data

Figure 9. Using partially faulty resources

In general, a faulty container cannot be used to host accelerators anymore. If the faulty fraction of the container is known, it may be still possible to use the partially faulty resource for some accelerators. The runtime system has a notion of resource requirements of accelerators and can compare that with the identified faulty resource. When binding accelerators to containers, the runtime system has to ensure that the accelerator that shall be reconfigured to a container does not use a partially faulty region of this container (Fig. 9b). In the best case, all demanded accelerators can still be configured to the available containers by re-binding them accordingly, which allows graceful degradation of the system.

VI. Conclusion

In this paper a brief overview of the research goals and activities within the *OTERA* project is provided. A case study is presented that describes the application of the developed *pre-configuration online test* (PRET) for a runtime reconfigurable system. The results show the high flexibility and low overhead for the structural online test of the FPGA infrastructure (0.33% performance loss when exhaustively testing all containers every 4.3 seconds). Moreover, an outlook on the remaining research tasks and required components and strategies for reliable reconfigurable systems is given.

VII. Acknowledgment

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] http://www.conveycomputer.com/, Convey Computer Corp.
[2] S. Kirsch *et al.*, "An FPGA-based high-speed, low-latency processing system for high-energy physics", in *Conf. on Field Program. Logic and Applications*, 2010, pp. 562–567.
[3] P. Garcia *et al.*, "An overview of reconfigurable hardware in embedded systems", *EURASIP Journal on Embedded Systems*, pp. 1–19, 2006.
[4] M. Shafique, L. Bauer, and J. Henkel, "Selective instruction set muting for energy-aware adaptive processors", in *Int'l Conf. on Computer-Aided Design*, 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 *Design Autom. 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 Computers*, vol. 27, no. 6, pp. 4–5, 2010.

[8] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*. Springer, 2007.

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

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

[11] L. Bauer, M. Shafique, and J. Henkel, "Concepts, architectures, and run-time systems for efficient and adaptive reconfigurable processors", in *NASA/ESA 6th Conf. on Adaptive Hardware and Systems (AHS)*, 2011, pp. 80–87.

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

[13] A. Jacobs, A. George, and G. Cieslewski, "Reconfigurable fault tolerance: A framework for environmentally adaptive fault mitigation in space", in *Int'l Conf. on Field Programmable Logic and Appl. (FPL)*, 2009, pp. 199–204.

[14] A. Avizienis *et al.*, "Basic concepts and taxonomy of dependable and secure computing", *IEEE Trans. on Dep. and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

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

[16] E. Chmelar, "FPGA interconnect delay fault testing", in *IEEE Int'l Test Conference*, 2003, pp. 1239–1247.

[17] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Elsevier, 2001.

[18] F. L. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Springer, 2006.

[19] C. Hu and S. Zain, "NSEU mitigation in avionics applications", *Xilinx Application Note XAPP1073 v1.0*, 2010.

[20] S. Velusamy *et al.*, "Monitoring temperature in FPGA based SoCs", in *Proc. IEEE International Conference on Computer Design*, 2005, pp. 634–637.

[21] M. Agarwal *et al.*, "Circuit failure prediction and its application to transistor aging", in *VLSI Test Symposium*, 2007, pp. 277–286.

[22] S. Durand and C. Piguet, "FPGAs with self-repair capabilities", in *ACM Int'l Workshop on FPGAs*, 1994, pp. 1–6.

[23] E. Stott, P. Sedcole, and P. Cheung, "Fault tolerant methods for reliability in FPGAs", in *Int'l Conference on Field Programmable Logic and Applications*, 2008, pp. 415–420.

[24] J. Emmert *et al.*, "Dynamic fault tolerance in FPGAs via partial reconfiguration", in *IEEE Symp. on Field-Program. Custom Computing Machines*, 2000, pp. 165–174.

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

[26] F. Sironi *et al.*, "Self-aware adaptation in FPGA-based systems", in *International Conference on Field Programmable Logic and Applications*, 2010, pp. 187–192.

[27] M. Tahoori, "Application-Dependent Testing of FPGAs", *IEEE Trans. on VLSI*, vol. 14, no. 9, pp. 1024–1033, 2006.

[28] 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.

[29] A. Van De Goor, "Using march tests to test SRAMs", *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8–14, 1993.

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

[31] W. K. Huang *et al.*, "Testing configurable LUT-based FPGA's", *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, pp. 276–283, 1998.

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

[33] A. Friedman, "Easily Testable Iterative Systems", *IEEE Trans. on Comp.*, vol. C-22, no. 12, pp. 1061–1064, 1973.

[34] 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.

[35] 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.

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

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

[38] X. Sun *et al.*, "Novel technique for built-in self-test of FPGA interconnects", in *International Test Conference*, 2000, pp. 795–803.

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

[40] 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.

[41] 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 *Int'l Conf. on Embedded Systems and Applications (ESA)*, 2006, pp. 87–93.

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

[43] 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-Tol. in VLSI Systems*, 2009, pp. 29–37.

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

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

[46] M. S. Abdelfattah *et al.*, "Transparent structural online test for reconfigurable systems", in *Proc. IEEE International On-Line Test Symposium (IOLTS)*, 2012.

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

[48] W. H. Kautz, "Testing for faults in combinational cellular logic arrays", in *Symposium on Switching and Automata Theory (SWAT)*, 1967, pp. 161–174.

[49] 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.

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

[51] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases", in *Int'l Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2011, pp. 1–8.