Aging Resilience and Fault Tolerance in Runtime Reconfigurable Architectures

Zhang, Hongyan; Bauer, Lars; Kochte, Michael A.; Schneider, Eric; Wunderlich, Hans-Joachim; Henkel, Jörg

IEEE Transactions on Computers Vol. 66(6) 1 June 2017

doi: http://dx.doi.org/10.1109/TC.2016.2616405

Abstract: Runtime reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) allow area- and power-efficient acceleration of complex applications. However, being manufactured in latest semiconductor process technologies, FPGAs are increasingly prone to aging effects, which reduce the reliability and lifetime of such systems. Aging mitigation and fault tolerance techniques for the reconfigurable fabric become essential to realize dependable reconfigurable architectures. This article presents an accelerator diversification method that creates multiple configurations for runtime reconfigurable accelerators that are diversified in their usage of Configurable Logic Blocks (CLBs). In particular, it creates a minimal number of configurations such that all single-CLB and some multi-CLB faults can be tolerated. For each fault we ensure that there is at least one configuration that does not use that CLB. Secondly, a novel runtime accelerator placement algorithm is presented that exploits the diversity in resource usage of these configurations to balance the stress imposed by executions of the accelerators on the reconfigurable fabric. By tracking the stress due to accelerator usage at runtime, the stress is balanced both within a reconfigurable region as well as over all reconfigurable regions of the system. The accelerator placement algorithm also considers faulty CLBs in the regions and selects the appropriate configuration such that the system maintains a high performance in presence of multiple permanent faults. Experimental results demonstrate that our methods deliver up to 3.7x higher performance in presence of faults at marginal runtime costs and 1.6x higher MTTF than state-of-the-art aging mitigation methods.

Preprint

Aging Resilience and Fault Tolerance in Runtime Reconfigurable Architectures

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

Abstract—Runtime reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) allow area- and power-efficient acceleration of complex applications. However, being manufactured in latest semiconductor process technologies, FPGAs are increasingly prone to aging effects, which reduce the reliability and lifetime of such systems. Aging mitigation and fault tolerance techniques for the reconfigurable fabric become essential to realize dependable reconfigurable architectures.

This article presents an accelerator diversification method that creates multiple configurations for runtime reconfigurable accelerators that are diversified in their usage of Configurable Logic Blocks (CLBs). In particular, it creates a minimal number of configurations such that all single-CLB and some multi-CLB faults can be tolerated. For each fault we ensure that there is at least one configuration that does not use that CLB.

Secondly, a novel runtime accelerator placement algorithm is presented that exploits the diversity in resource usage of these configurations to balance the stress imposed by executions of the accelerators on the reconfigurable fabric. By tracking the stress due to accelerator usage at runtime, the stress is balanced both within a reconfigurable region as well as over all reconfigurable regions of the system. The accelerator placement algorithm also considers faulty CLBs in the regions and selects the appropriate configuration such that the system maintains a high performance in presence of multiple permanent faults.

Experimental results demonstrate that our methods deliver up to 3.7× higher performance in presence of faults at marginal runtime costs and 1.6× higher MTTF than state-of-the-art aging mitigation methods.

Index Terms—Runtime reconfiguration, aging mitigation, fault-tolerance, resilience, graceful degradation, FPGA

1 Introduction

Runtime reconfigurable architectures enable dynamic adaptation to changing workloads, which allows to optimize area, performance and power [1]. They find widespread use in various fields from acceleration of web-page ranking in data centers [2] to payload processing in space [3]. Typical reconfigurable architectures such as Xilinx' Zynq UltraScale+ MPSoC [4] and Intel Xeon+FPGA platforms [5] consist of a general purpose processor and a reconfigurable fabric implemented on an SRAM-based FPGA. For runtime reconfiguration, the fabric is partitioned into multiple rectangular regions into which hardware accelerators can be reconfigured to improve application's performance and efficiency.

The FPGA-based reconfigurable fabric, manufactured in latest technology nodes, suffers from degradation due to aging [6, 7]. The resilience of the reconfigurable fabric is essential to the dependability of reconfigurable architectures, as most of the application's computations are offloaded to the reconfigurable fabric. The manifestations of aging can range from increased transistor switching delay up to permanent faults that cause a transistor or interconnect wire to fail entirely. Different aging mechanisms have been reported for the current generation of CMOS designs, e.g. negative/positive biased temperature instability (NBTI/PBTI), time-dependent dielectric breakdown (TDDB), hot carrier injection (HCI), or electromigration (EM) [8].

The main causes of these effects are environmental and electrical *stress*. Stress can be induced in different ways, e.g. through the presence of strong electrical fields or high current density [6, 9]. Due to the increasing susceptibility of ever-shrinking nano-CMOS devices, these effects cannot be ignored anymore [10]. Their consideration has become essential for dependable and resilient designs [11], where aging mitigation, detection of permanent faults and fault tolerance techniques for the reconfigurable fabric need to be an integral part of runtime reconfigurable architectures.

1.1 System overview

In this article we target a general reconfigurable processor architecture together with applications that use application-specific accelerators. Figure 1 shows the hardware and software view of the system. The *reconfigurable fabric* consists of multiple *reconfigurable regions* that can be reconfigured at runtime to contain *accelerators*. A reconfigurable region is a two dimensional array of Configurable Logic Blocks (CLBs). Each accelerator may use any subset of these CLBs for its implementation. All reconfigurable regions are of identical size and shape, which allows any accelerator to be configured into any region. Accelerator relocation techniques allow to use only one implementation (i.e. partial bitstream) per accelerator, regardless of the region into which the accelerator shall be reconfigured at runtime [12, 13]. While it would be possible to use the reconfigurable fabric in a more flexible manner (e.g.

- *H. Zhang, L. Bauer, and J. Henkel are with the Chair for Embedded Systems, Karlsruhe Institute of Technology (KIT), Germany. E-mail: {hongyan.zhang, lars.bauer, henkel}@kit.edu*
- *M. A. Kochte, E. Schneider, and H.-J. Wunderlich are with the Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany. E-mail: {kochte, schneiec}@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de*

Manuscript received April 2016; revised ...

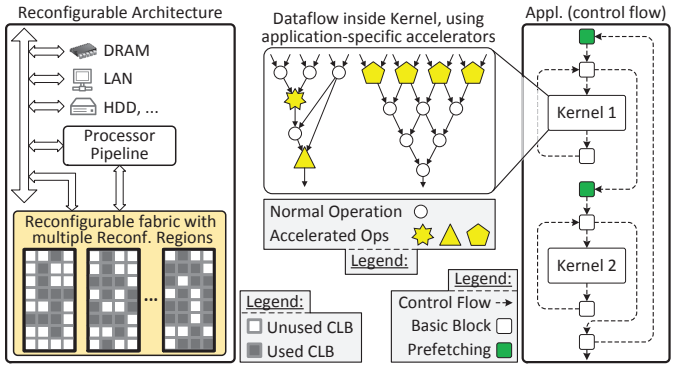

Fig. 1: System Overview showing how applications use accelerators that are reconfigured into reconfigurable regions and how these regions are coupled to the bus and processor pipeline

variable-sized regions), it would come with significant drawbacks such as the demand to create different implementations per accelerator (optimized for different region types), complex management of available resources [14] and the difficult aspect of establishing communication between variable-sized regions [15]. Providing the required infrastructure and overhead may pay off for systems that implement entire applications as accelerators. For instance, an H.264 video encoder application that is implemented as one big reconfigurable hardware block has significantly different requirements than an AES encryption application (needs much less resources), and thus they might benefit from a more flexible fabric management. However, we target applications that are mainly implemented as software (due to the ease of developing and deploying complex control as software), where only the computationally intensive parts are accelerated by hardware.

The software view for the targeted system is shown in the right half of Fig. 1. An application consists of one or multiple computationally intensive kernels (loops) that may contain computations that are worth being implemented by hardware accelerators. The application informs about the upcoming accelerators before entering the kernel (prefetching), which triggers their reconfiguration into the reconfigurable regions. An example for the computation inside a kernel is shown in the middle of Fig. 1. It consists of a mixture of normal operations (executed on the processor pipeline) and operations that correspond to the accelerators. A required accelerator may not be available (i.e. reconfigured) on the reconfigurable fabric, e.g. because its reconfiguration did not finish yet or because it could not be reconfigured due to too many permanent faults in the reconfigurable fabric. In such a case, the execution of an accelerated operation is emulated in software on the processor pipeline by issuing an 'unimplemented instruction' trap [16]. This ensures that the application can be executed as long as the processor pipeline is functional. We assume that a hardened processor pipeline is used that may also be implemented using more conservative cell libraries, as most of the computation is offloaded to the reconfigurable fabric anyway. Therefore, our article focuses on the aging resilience and fault tolerance of the reconfigurable fabric.

1.2 Contributions of this Article

In this article we aim at improving the aging resilience and fault tolerance in runtime reconfigurable architectures. At first, we propose a design method called *Accelerator Diversification* (see Fig. 2) that enables us to tolerate permanent faults in the reconfigurable regions. For each accelerator, a set of configurations is generated that is diversified in terms of their CLB usages, such that for every CLB in a region, at least one configuration of an accelerator does not require that CLB. This article presents a generic algorithm to generate the minimal set of configurations to tolerate arbitrary single-CLB faults and to generate additional configurations to tolerate multi-CLB faults in a reconfigurable region.

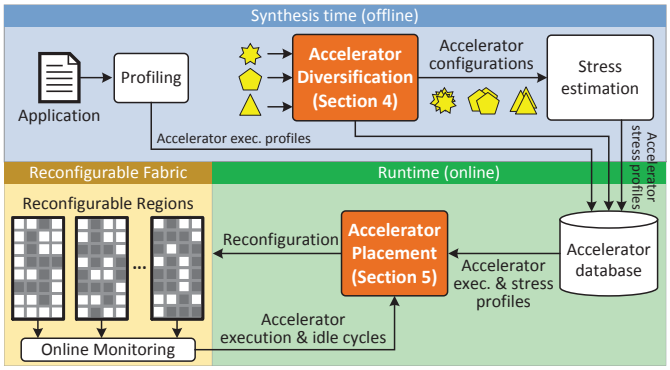

Fig. 2: Overview of the proposed methods

At runtime, the *Accelerator Placement* (see Fig. 2) uses the diversified configurations to tolerate faults. For a given set of accelerators that the application requested and in presence of multiple permanent faults in one or multiple regions, the accelerator placement aims at finding a combination of accelerator configurations and regions (into which the configurations can be reconfigured), such that the application performance remains relatively high in comparison to the system without faults, i.e. many accelerators can still be placed.

In addition, to avoid the emergence of faults due to aging effects, our accelerator placement uses the diversified configurations to distribute stress. At runtime, it simultaneously considers the induced intra- and inter-region stress distribution for the placement decision to mitigate aging effects by reducing the maximum stress of transistors in the reconfigurable fabric. We distinguish two types of stress: *static stress* (e.g. BTI) or *dynamic stress* (e.g. HCI) and our accelerator placement balances it among all resources in the reconfigurable fabric instead of accumulating it in individual transistors, which leads to aging mitigation and increased system lifetime.

For prototyping purposes, we have integrated the Accelerator Diversification into the Xilinx tool-chain and the fault-tolerant and stress-aware Accelerator Placement method into the runtime system of a reconfigurable architecture. Altogether, **the main novel contributions of this article are:**

- The accelerator diversification design method that allows to tolerate permanent faults and to mitigate the aging process.
- An algorithm that generates the minimal set of accelerator configurations for tolerating any single-CLB fault in one region and that generates additional configurations for regions with multi-CLB faults (i.e. multiple faulty CLBs).
- A runtime accelerator placement method that is fault-tolerant and stress-aware:

– Minimizes application performance degradation by placing *critical* accelerators first, i.e. those with the fewest placement options due to faults in reconfigurable regions.
– Considers the intra- and inter-region distribution of stress induced by accelerators to reduce the maximum stress and mitigate aging. Achieves efficient search space pruning to reduce the runtime overhead by calculating guaranteed bounds of the achievable stress distribution.

Paper structure: Section 2 discusses related work and state-of-the-art. Section 3 presents the background on aging and how we represent it in our methods. Our novel accelerator diversification and placement methods are presented in Sections 4 and 5. Section 6 shows the technical details of our implementation flow. We evaluate our methods and compare with state-of-the-art in Section 7 and conclude in Section 8.

2 Related Work

This section discusses related work for aging mitigation in FPGA-based runtime reconfigurable architectures based on diversified resource usage. If excessive stress is imposed on the resources in the reconfigurable fabric, permanent faults may emerge. Such faults can be detected and localized by online test methods for FPGAs. Once located, fault tolerance and recovery methods can be applied to avoid using faulty resources. These methods are typically based on resource remapping to spare resources by partial reconfiguration.

2.1 Aging Mitigation in Reconfigurable Architectures

Aging mitigation by wear-leveling in runtime reconfigurable architectures can be achieved by using alternative logic mappings in CLBs, using spare resources in the fabric, or changing placements of accelerators. The coarse-grained approach in [9] uses only two different configurations, which are swapped only once after a half-life period of the first failing component. A similar idea is used in [17], where three strategies are discussed for FPGA wear-leveling based on signal state inversion, use of spare resources for timing critical functions, and alternative placement. Since only two different configurations are used, the effectiveness is limited. In [18], a combined process variation and NBTI-aware placement algorithm is proposed. While the authors suggest that the logic placement and configuration bitstream generation could be recomputed during runtime, for most embedded systems such a computation would cause too much overhead. In [19], aging in LUTs is mitigated by manipulating the configuration bits of LUTs. This method targets static systems in which the logic function of LUTs does not change during runtime and not runtime reconfigurable architectures.

Since typically not all CLBs in a region are actively used by an accelerator configuration [9], it is possible to prepare alternative placements and to reconfigure between them to distribute stress. The CLBs that are unused in a particular configuration can be configured to minimize stress [17]. This reduces the maximum stress in the resources and increases the system's Mean Time to Failure (MTTF), as demonstrated in [9, 20, 21]. The methods in [9, 20] only target systems without runtime reconfiguration. They create alternative configurations for the entire FPGA, i.e. placing *one* complete design anywhere on the FPGA. In [21, 22], runtime reconfiguration with multiple regions and accelerators is considered. However, they only distribute the stress within a region, i.e. intra-region stress distribution, whereas our method also performs inter-region stress distribution in addition to providing fault tolerance. In contrast to our work, the target system in [22] implements the entire application in one region.

The online placement of accelerators in [23] extends the KAMER placement algorithm [24] by considering the maximum stress in the regions at runtime. The accumulated stress values of the resources in the candidate region are stored in a degradation table and their algorithm performs a local optimization that considers one accelerator after the other. The stress-aware accelerator placement STRAP [25] distributes the stress of accelerators among all reconfigurable regions. The methods of [23] and [25] only use one configuration per accelerator, whereas we use multiple diversified configurations per accelerator, which allows for a higher stress-balancing potential. Additionally, our method explicitly supports fault tolerance and also aging mitigation by stress balancing in presence of permanent faults, which is not targeted by [23, 25]. With regard to aging mitigation in reconfigurable architectures, the methods of [23, 25] represent the current state-of-the-art approaches and we will compare with both in the evaluation to show the advantages of our methods.

2.2 Online Test and Diagnosis of Reconfigurable Systems

Online test and diagnosis methods are a prerequisite for handling faults in reconfigurable systems. Application dependent and independent test methods for the reconfigurable FPGA fabric can be distinguished. Application independent testing targets the whole fault universe of the fabric and is not limited to a specific use of the fabric. It typically employs multiple special test configurations and corresponding test stimuli [26]. In contrast, application dependent tests target only the subset of programmable resources of the FPGA fabric relevant for a particular target application [27].

For an online test in the field, external equipment or circuitry for test pattern generation or output response analysis is not available. Internal testing approaches based on built-in self-test (BIST) principles include test pattern generation and output response analysis in the circuit under test [28].

In FPGAs with partial dynamic reconfiguration, the reconfigurations during test application can be performed by an external or embedded processor at runtime [29–32]. The Roving STARs (Self Testing AReas) method for online test partitions the FPGA into rows and columns, which can be either used functionally or tested by an online BIST approach [31, 32]. The transparent integration of online tests into runtime reconfigurable architectures was presented in [33, 34]. It was shown that online testing concurrent to system operation causes a negligible performance impact of less than 1%.

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 configurations to distinguish and localize faults [35–37]. In [38], multiple faults are diagnosed and can be tolerated using multiple diversified configurations with disjunct resource usage. The number of required configurations quickly rises with the number of faults to be detected and localized. Diagnosis techniques based on special configurations, stimuli, and response evaluation can also be integrated

into runtime reconfigurable architectures and controlled by an embedded processor [30].

2.3 Fault Tolerance in Reconfigurable Systems

Once a fault is detected and localized, different methods can be applied to ensure continued system operation despite of the fault. Tile-based fault tolerance techniques partition the reconfigurable fabric into a 2-dimensional array of rectangular regions (tiles) [39, 40]. In [39], a tile consists of multiple CLBs with one spare CLB. If a CLB in a tile is detected to be faulty, an alternative configuration for that tile is loaded to implement the same logic function but using the spare rather than the faulty CLB. In [40] the circuit in the faulty tile is entirely remapped to a spare tile. Column-based approaches apply similar concepts to CLB columns [41, 42], where the fabric is partitioned into a 1-dimensional array of CLB columns, each of them can implement an accelerator. In response to a fault, spare columns are used. Both tile- and column-based approaches need complex customized routing techniques, requiring fixed interfaces between adjacent accelerators or online routing after accelerator remapping as the accelerator locations change and the communication in-between has to be re-established. The methods also do not maximize the diversity in resource usage or exploit it to balance the stress in the fabric.

The Roving STARs method [32] combines distributed CLB spares and online compilation of configurations to replace faulty CLBs with spares. For complex designs, this online compilation or synthesis may cause unpredictable timing behavior. The Roving STARS also do not balance the stress.

The authors of [43] propose to use alternative configurations for accelerators, each of which uses different CLBs such that any single defective CLB can be tolerated. However, they do not provide a method to automatically generate these configurations. They neither investigate the possibility of tolerating multiple CLB faults in general nor do they consider mitigation of aging effects within the reconfigurable fabric. Instead, the diversified configurations in [21] are systematically generated using standard tools so that all single-CLB faults and some multi-CLB faults are tolerated. The approach in [38] also generates diversified configurations similar to [21], but it also tolerates all multi-CLB faults where up to k CLBs in a region can be faulty. But none of these approaches use their diversified configurations to distribute stress at runtime across all reconfigurable regions, as we propose in this work.

3 Background

Before describing the details of our methods in Sections 4 and 5, we clarify basics about aging, explain the assumptions we make for our method, and how we represent stress. A more detailed study of stress and its relation to aging and mean time to failure (MTTF) can be found in [25].

3.1 Basic Stress Properties

Aside from material constants, aging mainly depends on three non-material factors: supply voltage, temperature, and transistor activities. In the proposed method we use a simplification that focuses on the aging effects induced by transistor activities. That is a reasonable approximation as reconfigurable accelerators are typically operated in a static voltage domain (i.e. no dynamic voltage scaling) and do not show high temperature variation because they are often optimized for data-level parallelism and thus run at rather low frequency [44] with correspondingly low power density [45, 46]. However, for the evaluation we use a more accurate model that also considers the influence of temperature.

The term *stress* is defined as the condition under which a transistor is experiencing electrical and physical degradations. An example for such a degradation is the threshold voltage shift ΔV_{th}, which may eventually cause a failure of the circuit.

In the following, we distinguish two types of stress in nano-scale CMOS circuits: static and dynamic stress. A transistor is under *static stress* when an electric field is exerted across its gate oxide to induce a conducting channel. It is under *dynamic stress* when current flows between its source and drain. Static stress is characterized by the *stress duty cycle*, i.e. the fraction of operation time that a transistor is conducting. Dynamic stress is characterized by the toggling rate, i.e. the ratio of number of toggles and total operating time. Reducing the stress time $stress_{stat}$ reduces the stress duty cycle, while reducing the number of toggles $stress_{dyn}$ reduces the toggling rate. Both reduce the transistor degradation, i.e. threshold voltage shift. Static stress leads to aging effects like BTI, while dynamic stress leads to aging effects like HCI.

Different models have been proposed for these aging effects, e.g. [6, 8, 9, 47, 48]. They all indicate that in the long term the transistor degradation *monotonically* increases with $stress_{stat}$ or $stress_{dyn}$ for static or dynamic stress, respectively. For instance, $\Delta V_{th}(stress_{stat1}) > \Delta V_{th}(stress_{stat2})$ when $stress_{stat1} > stress_{stat2}$ under the same supply voltage and temperature [47, 48]. In other words, the aging effects are reduced when $stress_{stat}$ or $stress_{dyn}$ is reduced.

In addition, $stress_{dyn}$ is generally considered as *additive*. For instance, the dynamic stress of two different workloads corresponds to the number of toggles that these workloads impose on a transistor. Intuitively, the combined dynamic stress is the sum of these toggles, which is proportional to the amount of charge transported between drain and source [48, 49]. In general, the total stress experienced by a transistor under different workloads ($stress_{dyn}(work_1 + work_2)$) is the sum of stress experienced under the individual workloads ($stress_{dyn}(work_1) + stress_{dyn}(work_2)$). In the long term, this argument also holds for $stress_{stat}$. Actually, BTI aging may experience a *recovery effect*, but that requires complex conditions or long relaxation periods [7] and will thus hardly affect the additive property.

The monotonic and additive properties of $stress_{stat}$ and $stress_{dyn}$ allow a simplified consideration of CLB stress during accelerator placement (see Section 5) rather than evaluating complex aging models at runtime. The proposed stress-aware accelerator placement applies to both types of stress and we will refer to "*stress*" when we do not need to explicitly differentiate between static and dynamic stress. Note that it optimizes either for dynamic or for static stress.

3.2 Stress Representation

The transistors of a reconfigurable region are stressed by the reconfigured accelerator in a way that is determined by its logic functionality and input signal patterns. As the number of transistors in a region may be huge, we combine the stress

experienced by individual transistors to CLB granularity for our accelerator placement method. We define *CLB stress* as the sum of the stress experienced by all its transistors. With this definition, CLB stress preserves the monotonic and additive property of transistor stress, i.e. the total stress a CLB experienced from different accelerators is the sum of the induced stress from individual accelerators.

With the established stress properties, we can describe the stress in the reconfigurable fabric in a formal way. The stress state of a reconfigurable region is denoted as matrix $\mathbf{S}$, where each entry represents the stress experienced by the corresponding CLB in the region. The stress that a particular accelerator configuration induces per clock cycle is obtained from offline stress estimation and called *unit stress*, denoted by a matrix of the same size as $\mathbf{S}$. In general, the induced stress due to the work done by an accelerator configuration is shown in Eq. (1), where scalars τ_{exec} and τ_{idle} are the number of clock cycles when the accelerator is in execution or idle, while matrices $\mathbf{s^{unit}_{exec}}$ and $\mathbf{s^{unit}_{idle}}$ denote the unit stress induced by the accelerator during execution or idle time.

$$\mathbf{s} := \tau_{exec}\mathbf{s^{unit}_{exec}} + \tau_{idle}\mathbf{s^{unit}_{idle}} \tag{1}$$

During idle time, we assume all inputs to the accelerator are hold at constant values. In this case, the accelerator exhibits a different stress pattern from when it performs an execution. The stress estimation flow to obtain the unit stress of an accelerator is described in Section 6. During synthesis time, the values for τ_{exec} and τ_{idle} are obtained from application profiling to construct the stress matrices (Eq. (1)) for every accelerator. The runtime system uses them to determine how much stress an accelerator would induce to a region *before* actually placing it. It also uses online monitoring information (see Fig. 2) that provides the actual number of accelerator executions and idle times for each region *after* a computational kernel finished execution. This allows to keep track of the actual stress that a region experienced, which is the starting point for the next placement decision.

Note that the stress matrix $\mathbf{S}$ can also be used to incorporate process variation. Instead of initializing $\mathbf{S}$ to the zero matrix, its initial values can be used to represent process variation. They can be determined by measuring the delay of each individual LUT after manufacturing, as proposed in [6].

4 Accelerator Diversification for Fault Tolerance and Aging Mitigation

An accelerator defines the logic functions to be implemented in a region, while a *configuration* of the accelerator determines which CLBs in the region are used to implement the functionality. We use a Boolean matrix, called *configuration matrix*, with the same dimension as a region to describe the CLB usage of a configuration. If a CLB is used, the corresponding matrix element is 1, otherwise 0. For example, an accelerator configuration using five CLBs implemented in a 3×3 region can be represented in a configuration matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

The accelerator diversification method generates a set of configurations, each of which implements the same accelerator, but uses different CLB resources, such that any single faulty CLB in a region can be tolerated by one of the diversified configurations. Formally, we search a set of configurations C for an accelerator implemented in an $X \times Y$ region:

$$C = \{\mathbf{A_1}, \cdots, \mathbf{A_w}\},\ \mathbf{A_i} : X \times Y \text{ Boolean matrix} \tag{3}$$

Assume that all of these configurations utilize the same amount of CLBs U and there is at least one free CLB, i.e. $\forall \mathbf{A_i} \in C : \sum_{x,y} [\mathbf{A_i}]_{xy} = U < XY$. To be able to tolerate any single faulty CLB, this set of configurations must satisfy the *completeness condition*:

$$\begin{gathered} \forall x, y, 1 \le x \le X, 1 \le y \le Y : \\ \exists \mathbf{A_i} \in C \text{ such that } [\mathbf{A_i}]_{x,y} = 0 \end{gathered} \tag{4}$$

The completeness condition guarantees that if any CLB is detected to contain faults, there always exists a diversified configuration $\mathbf{A_i}$ that does not require the faulty CLB. For an accelerator requiring U CLBs to be implemented in an $X \times Y$ region, at least W_{min} configurations are required for the completeness condition:

$$w_{min} = \lceil \tfrac{XY}{XY-U} \rceil \tag{5}$$

In each configuration, exactly $XY - U$ CLBs are spare. For a configuration $\mathbf{A_i}$, at most $XY - U$ CLBs that were not spare in any of the configurations $\mathbf{A_j}, j < i$ can be spare in $\mathbf{A_i}$, which directly results in this lower bound.

In order to minimize the number of diversified configurations for satisfying the completeness condition, we require that the generated set of configurations also satisfies the *max diversification condition*:

$$\begin{gathered} \forall i, 1 \le i \le w : \exists \mathbf{A_j} \in C, j \ne i \text{ such that} \\ \sum_{x,y} \Big([\mathbf{A_i}]_{xy} \cdot [\mathbf{A_j}]_{xy} \Big) = \begin{cases} 2U - XY & \text{if } U > \frac{1}{2}XY \\ 0 & \text{else} \end{cases} \end{gathered} \tag{6}$$

We define that two configurations are maximally diversified if the difference between them is maximized. The minimum number of common CLBs between two configurations is either 0, if the accelerator requires at most half of the available CLB resources, or $2U - XY$, whenever all unused CLBs $(XY - U)$ in one configuration are used in the other configuration. In the latter case, the number of common CLBs is $U - (XY - U)$. The max diversification condition states that for every configuration $\mathbf{A_i} \in C$ there is at least one other configuration $\mathbf{A_j}$ which differs from $\mathbf{A_i}$ as much as possible w.r.t. the used CLB resources.

Enumerating all possible configurations to find a maximally diversified set of configurations is computationally intractable. For instance, if an accelerator requires 50 CLBs in a region with 80 CLBs, then there are $\binom{80}{50} \approx 9 \times 10^{21}$ possible configurations. Alg. 1 presents the generation of a given number of configurations that satisfy the completeness condition and maximizes their diversity. It incrementally generates diversified configurations from an initial configuration $\mathbf{A_1}$.

In Line 1, the set of diversified configurations C is initialized with the initial configuration. The score matrix $\mathbf{G}$, which has the same dimension as the configuration matrix, stores for each CLB the number of diversified configurations which use that CLB. The score matrix is simply the sum of all configuration matrices in C. In Line 2, $\mathbf{G}$ is initialized to $\mathbf{A_1}$, the only element in C at the moment. In Line 3, the next new

Algorithm 1 Generation of diversified configurations C

1. $C := \{\mathbf{A_1}\}$ // $\mathbf{A_1}$ is the initial configuration
2. $\mathbf{G} := \mathbf{A_1}$ // Score matrix $\mathbf{G}$ stores swapping priority of CLBs
3. $\mathbf{A_{new}} := \mathbf{A_1}$
4. **loop**
5. `zero_elem_list` $:= \{(x,y) \mid [\mathbf{A_{new}}]_{xy} = 0\}$ // unused CLBs
6. `candidate_list` $:= \{(x,y) \mid [\mathbf{A_{new}}]_{xy} = 1\}$
7. sort `candidate_list` according to the value of $\mathbf{G}_{xy}$ in descending order // first element has the highest score
8. **for all** (x,y) in `zero_elem_list` **do**
9. `swap_candidates` $:= \{(p,q) \mid (p,q) \in$ `candidate_list` and $\mathbf{G}_{pq} = \mathbf{G}_{\texttt{candidate\_list[0]}}\}$ // all CLBs with the highest score
10. `farthest_swap_candidate` $:= (p,q) \in$ `swap_candidates` with max. Manhattan distance between (x,y) and (p,q) // farthest elements are swapped first so that CLBs are located near each other and better timing is achieved
11. `swap`$([\mathbf{A_{new}}]_{xy}, [\mathbf{A_{new}}]_{\texttt{farthest\_swap\_candidate}})$
12. `candidate_list.pop(farthest_swap_candidate)`
13. **if** `candidate_list` $= \emptyset$ **then**
14. **break**
15. **end if**
16. **end for**
17. **while** $\mathbf{A_{new}} \in C$ **do**
18. swap a random zero- with random one-element in $\mathbf{A_{new}}$
19. **end while**
20. $\mathbf{G} := \mathbf{G} + \mathbf{A_{new}}$ // update CLB score
21. $C := C \cup \{\mathbf{A_{new}}\}$
22. **if** $|C| =$ desired number of config. $\vee\ |C| = \binom{XY}{U}$ **then**
23. **break**
24. **end if**
25. **end loop**

configuration matrix $\mathbf{A_{new}}$ is initialized to the initial configuration matrix. In the inner loop (Lines 8 to 16), it is further modified by swapping zero- and one-elements. The inner loop iterates through all zero-elements in $\mathbf{A_{new}}$ and swaps zero-elements with one-elements in $\mathbf{A_{new}}$ in an order determined by the score matrix (Line 7). If a CLB has a higher score (i.e. it is used more often in the diversified configurations), its corresponding one-element in $\mathbf{A_{new}}$ will be first swapped. If there are several CLBs with the same score, the farthest one from the current zero-element is swapped first (Lines 9 to 11) so that in the resulting configuration, the used CLBs are located near each other. The first generated $\lceil \frac{X \cdot Y}{X \cdot Y - U} \rceil$ configurations correspond to the *minimal* set of configurations that satisfies both the completeness and max diversification condition. It is guaranteed that the random swapping (Line 18) does not occur while generating the minimal set.

If the user requires more configurations for higher reliability (i.e. tolerate more multi-CLB faults) or to have more alternatives for aging mitigation by stress balancing, further possible configurations can be generated (this might use the random swapping in Line 18 at some time). The algorithm terminates when either the desired number of configurations or all possible configurations have been generated. In both cases, the generated set of configurations always satisfies the completeness condition but may violate the max diversification condition due to the while loop from Lines 17 to 19, where random changes are made to $\mathbf{A_{new}}$ to generate a new unique configuration matrix.

The proposed accelerator diversification design method is in principle applicable for all regularly distributed resources of the fabric. In Xilinx FPGAs, the routing resources are regularly distributed: One programmable switching matrix is attached to each CLB. Thus, the resource usage patterns for target configurations computed by the proposed method can also diversify the use of programmable routing resources.

5 Stress-Aware and Fault-Tolerant Runtime Accelerator Placement

The reconfigurable fabric consists of N equally sized rectangular regions. During runtime, the application requests to configure M ($M \leq N$) accelerators to speed up its computational kernels. The runtime system has to decide to which regions the M accelerators shall be configured and which diversified configurations shall be used. When faulty CLBs are detected in a reconfigurable region, then only those accelerator configurations can be placed into the region that do not require them. The application performance will experience degradation when not all requested accelerators can be placed.

We propose a placement algorithm that follows these rules: 1) place as many as possible accelerators to minimize the application performance degradation when faults are detected in the reconfigurable regions, 2) distribute stress evenly among different CLBs within the region (intra-region distribution) by maximally utilizing under-stressed CLBs within the region, and 3) avoid placing high-stress accelerators into highly stressed regions, i.e. the stress shall be evenly distributed among different regions (inter-region distribution). In the following, regions are indexed with letter k, accelerators with j and diversified configurations of an accelerator with w.

5.1 Placement Algorithm

We show the pseudo-code of our fault-tolerant and stress-aware placement algorithm (Alg. 2) and give a brief explanation of it. Algorithmic details are presented in Sections 5.2 to 5.4. Our algorithm determines the diversified configuration and region to be configured for all requested accelerators, one after the other. The placeability of an accelerator configuration, i.e. into which regions the accelerator configuration can be reconfigured, is determined after a new fault has been detected and localized. It is then stored in a lookup table and used in Alg. 2 (see Line 14). The actual reconfigurations are just started after all accelerator placements are decided, because under the presence of permanent faults, an initially considered placement may be changed at a later phase of the algorithm, if that allows to place more accelerators.

Before the placement starts, the required accelerators are sorted according to their *placement freedom* in ascending order (see Line 2). The placement freedom of an accelerator corresponds to the number of regions for which the accelerator has at least one diversified configuration that can be placed into that region (i.e. that tolerates the permanent faults in that region). Such a region is called a *compatible* region. The runtime accelerator placement then iterates over all accelerators j that shall be placed (Lines 3 to 46). In each iteration, it first determines those regions where accelerator j is placeable (Line 7), i.e. where the faulty CLBs can be avoided by one of its diversified configurations. Then, it prunes out

Algorithm 2 Runtime accelerator placement

Input: List of accelerators `Acc` and list of regions `Reg` that shall be reconfigured

```
1. occupied := array of length len(Reg) initialized to zeros
2. Acc.sort(key=#compatible regions, order=ascending)
3. for j := 1 to len(Acc) do
4.   max_profit := −∞
5.   sel_reg := null // Selected region
6.   sel_conf := null // Selected configuration
7.   Reg_tmp := regions in Reg compatible with Acc[j]
8.   Reg_tmp := Reg_tmp pruned based on bounds of placement profits (see Section 5.4)
9.   for k := 1 to len(Reg_tmp) do
10.    if occupied[k] == 1 then
11.      continue
12.    end if
13.    for w := 1 to len(Acc[j]) do
14.      if Acc[j][w] not placeable in Reg_tmp[k] then
15.        continue
16.      end if
17.      profit := CalcProfit(Acc[j][w], Reg_tmp[k]) based on Eq. (9)
18.      if profit > max_profit then
19.        max_profit := profit
20.        sel_reg := k
21.        sel_conf := w
22.      end if
23.    end for
24.  end for
25.  if sel_reg != null then
26.    Place Acc[j] with sel_conf into region sel_reg
27.    occupied[sel_reg] := 1
28.    continue // Successful placement
29.  end if
30.  is_placed := false
31.  for i := 1 to j-1 do
32.    if Acc[j] placeable in Acc[i].reg then
33.      for k := 1 to len(Reg) do
34.        if occupied[k] == 0 and
           Acc[i] placeable in Reg[k] then
35.          Place Acc[j] into Acc[i].reg
36.          Place Acc[i] into Reg[k]
37.          is_placed := true
38.          break
39.        end if
40.      end for
41.    end if
42.    if is_placed then
43.      break
44.    end if
45.  end for
46. end for
```

those regions that are guaranteed to result in worse stress distribution than other regions after placing accelerator j into them (Line 8, details in Section 5.4). Afterwards, it calculates for all remaining regions k and for all diversified configurations w that are placeable into k the profit (details in Section 5.3) of placing w into k (Lines 9 to 24) and places the accelerator configuration into the region that provides the highest profit (Line 26). If no configuration is placeable in any of the remaining regions, then the algorithm tries to exchange some already placed accelerators (Lines 31 to 45; details in Section 5.2). The complexity of this algorithm is $\mathcal{O}(M^2WXY)$, where M denotes the number of required accelerators, W the maximum number of diversified configurations of the accelerators, and XY the number of CLBs (width×height) of a region.

If the application keeps using an accelerator for a longer time, then its region is not reconfigured and thus our placement algorithm cannot evenly distribute the stress to all regions. The region where this accelerator resides would be constantly stressed by one accelerator without stress redistribution. As a solution, our runtime accelerator placement forces that region to be reconfigured after a user-defined time period. This time period should not be too short to prevent increased reconfiguration overhead, but also not too long to avoid stress accumulation. For instance, a time period of 20 million cycles (0.2 s at 100 MHz) is short enough to avoid aging accumulation and the induced application performance degradation is only 0.51% (see results).

5.2 Fault-Tolerant Placement

When faults are detected in the reconfigurable fabric, the placement freedom of accelerators is reduced. An accelerator j can only be placed into a region, when at least one of its diversified configurations does not require the CLBs that are faulty in that region. If the available regions (i.e. those into which no accelerators are placed by the placement algorithm so far) have rather many permanent faults, it can happen that no configuration of accelerator j can be placed into any of them. If an accelerator cannot be placed, then its hardware functionality has to be emulated in software on the processor pipeline (see Section 1.1). This actually reduces the stress for the regions, as they are not used to execute the accelerator, however, it comes at the cost of significantly degraded performance (i.e. less acceleration for that kernel).

Alg. 2 tries to avoid such situations by placing accelerators one after the other in ascending order of their number of compatible regions (Line 2). If it still comes to the situation that some accelerator j cannot be placed into the available regions, then the algorithm re-evaluates some of its previous placement decisions (note that the actual reconfigurations are just started after all placements are finally decided). It tries whether it can *swap* one of the already placed accelerators into one of the still available regions such that accelerator j can be placed into the region that became free due to swapping (Lines 31 to 45).

Permanent faults also reduces the freedom of selecting diversified configurations for stress distribution. While our algorithm aims at achieving a uniform stress distribution, its highest priority is to place as many accelerators as possible to minimize the performance degradation due to faults.

5.3 Stress-Aware Placement Profits

Each region contains $X \times Y$ CLBs with an (x, y) coordinate relative to the top-leftmost CLB in the region. The stress experienced so far by the CLBs in region k is denoted as $[\mathbf{S_k}]_{xy}$ (with $1 \leq k \leq N,\ 1 \leq x \leq X,\ 1 \leq y \leq Y$). The stress that will be induced by a configuration w of accelerator j ($1 \leq j \leq M$) is denoted as $[\mathbf{s_{jw}}]_{xy}$ (see Eq. (1)). It depends on how often the accelerator will be executed, as determined by offline profiling (see Section 1.1). If an accelerator configuration jw is placed into region k, then the accelerator executions increase the stress state of the region to $\mathbf{S'_k} = \mathbf{S_k} + \mathbf{s_{jw}}$. Diversified configurations of an accelerator use the CLBs in a region in different ways. The total amount of induced stress is equal

for all diversified configurations since they perform the same computation. In the following, when we need to calculate the total stress induced by accelerator j, we omit the index w that is used to indicate a particular diversified configuration:

$$\forall w : \sum_{x,y} [\mathbf{s_{jw}}]_{xy} = \sum_{x,y} [\mathbf{s_j}]_{xy} \tag{7}$$

The goal of stress-aware placement is to place each accelerator to a region, such that upon completion of the application kernel the maximum CLB stress over the N regions is minimized, i.e. $\max_{k,x,y}[\mathbf{S'_k}]_{xy}$ is minimized. It can be easily seen that the strict lower bound of the maximum CLB stress is:

$$\frac{1}{NXY}\left(\sum_{k}^{N}\sum_{x,y}[\mathbf{S_k}]_{xy} + \sum_{j}^{M}\sum_{x,y}[\mathbf{s_j}]_{xy}\right) \tag{8}$$

It is reached if and only if the stress is uniformly distributed over all CLBs. Therefore, to minimize the maximum CLB stress in the reconfigurable fabric, the CLB stress from the accelerators that are to be placed needs to be distributed evenly. We define the profit function of placing accelerator j with configuration w into region k as

$$\text{Profit}_{jkw} = \text{Profit}^{intra}_{jkw} + \text{Profit}^{inter}_{jk} \tag{9}$$

where $\text{Profit}^{intra}_{jkw}$ and $\text{Profit}^{inter}_{jk}$ represent the profit from the stress distribution within one region (intra-region) and across all regions (inter-region), respectively:

$$\begin{aligned} \text{Profit}^{intra}_{jkw} &= \sum_{x,y}\left|[\mathbf{S_k}]_{xy} - \lambda_k\right| - \sum_{x,y}\left|[\mathbf{S_k}+\mathbf{s_{jw}}]_{xy} - \lambda'_{k,j}\right| \\ \lambda_k &= \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}]_{xy} \qquad \lambda'_{k,j} = \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}+\mathbf{s_j}]_{xy} \end{aligned} \tag{10}$$

$$\begin{aligned} \text{Profit}^{inter}_{jk} &= \left|\sum_{x,y}[\mathbf{S_k}]_{xy} - \Lambda\right| - \left|\sum_{x,y}[\mathbf{S_k}+\mathbf{s_j}]_{xy} - \Lambda'\right| \\ \Lambda &= \frac{1}{N}\sum_{k,x,y}[\mathbf{S_k}]_{xy} \qquad \Lambda' = \frac{1}{N}\left(\sum_{k,x,y}[\mathbf{S_k}]_{xy} + \sum_{j,x,y}[\mathbf{s_j}]_{xy}\right) \end{aligned} \tag{11}$$

The two summation operations in the intra-region profit function (Eq. (10)) express the sum of the CLB stress deviation from the average stress value before and after placing accelerator configuration j into region k, respectively. A larger sum of deviation implies that more CLBs are *over-* or *under-*stressed. This profit function thus describes the improvement of stress distribution within region k after placing accelerator configuration j into it. In a similar manner, the inter-region profit function in Eq. (11) describes the deviation from perfect even stress distribution evaluated at the level of reconfigurable regions, i.e. the deviation of the total stress in a region from the average total stress per region. It is independent of which diversified configurations are used for the accelerators, since only the total stress of an accelerator is concerned in Eq. (11).

5.4 Bounds of Placement Profits

The exact computation of $\text{Profit}^{intra}_{jkw}$ requires the summation of stress deviation of XY CLBs (second term of Eq. (10)), which is unique for every combination of j, k, and w. The XY CLB-level computations have to be performed for each configuration-region combination and thus represents the most compute-intensive part during the placement. Instead of an exhaustive profit computation of all configuration-region combinations, we propose a two-step maximum profit search algorithm. For a given accelerator j, we first compute the *ranges* of Profit_{jkw}, which is independent of w as shown later, for $1 \le k \le N$ without computing the exact values of intra-region profit. If a region has an *upper* bound of profit that is less than the *lower* bound of profit of any other region, placing the accelerator into this region would not result in maximum profit. This region is therefore excluded in the second step, where the exact profits of other regions are computed and compared to find the region with the maximum profit. In this way, the runtime overhead of the algorithm is reduced by early pruning of unnecessary CLB-level computations.

The range of $\text{Profit}^{intra}_{jkw}$ is determined as follows. By substituting $\lambda'_{k,j}$ into the equation, the second term of Eq. (10), i.e. the sum of CLB stress deviation after placement, can be rewritten as

$$\sum_{x,y}\left|\underbrace{[\mathbf{S_k}]_{xy} - \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}]_{xy}}_{D_{kxy}} + \underbrace{[\mathbf{s_{jw}}]_{xy} - \frac{1}{XY}\sum_{x,y}[\mathbf{s_j}]_{xy}}_{d_{jwxy}}\right| \tag{12}$$

where D_{kxy} denotes the stress deviation of the CLB at location (x, y) in *region* k from the average value, while d_{jwxy} denotes the stress deviation of the CLB at location (x, y) of *accelerator* j with configuration w from the average value. The sum of stress deviation $\sum_{x,y}|d_{jwxy}|$ for different diversified configurations of accelerator j is the same, since diversified configurations have the same average stress and differ only in the used CLB locations. Similar to Eq. (7), we omit the w index in the following by defining $\forall w : \sum_{x,y}|d_{jwxy}| = \sum_{x,y}|d_{jxy}|$. With triangle inequalities, it can be shown that

$$\sum_{x,y}|D_{kxy} + d_{jxy}| \ge \left|\sum_{x,y}|D_{kxy}| - \sum_{x,y}|d_{jxy}|\right| \tag{13}$$

$$\sum_{x,y}|D_{kxy} + d_{jxy}| \le \sum_{x,y}|D_{kxy}| + \sum_{x,y}|d_{jxy}| \tag{14}$$

Based on Eq. (10) and (12) to (14), we obtain the lower and upper bound of the intra-region profit of placing accelerator j into region k:

$$\begin{aligned} -\sum_{x,y}|d_{jxy}| &\le \text{Profit}^{intra}_{jk} \\ &\le \sum_{x,y}|D_{kxy}| - \left|\sum_{x,y}|D_{kxy}| - \sum_{x,y}|d_{jxy}|\right| \end{aligned} \tag{15}$$

Depending on the stress distribution of region k and accelerator j, following scenarios may occur. When the stress in region k is already uniformly distributed, i.e. very small $\sum_{x,y}|D_{kxy}|$, placing any accelerator into it will only bring marginal (when $\sum_{x,y}|d_{jxy}| \approx \sum_{x,y}|D_{kxy}|$) or even negative (when $\sum_{x,y}|d_{jxy}| \gg \sum_{x,y}|D_{kxy}|$) intra-region profit. When the stress of accelerator j is already uniformly distributed, i.e. very small $\sum_{x,y}|d_{jxy}|$, placing this accelerator into any region can bring at most marginal intra-region profit. A large intra-region profit can only be realized when both region k and accelerator j have very nonuniform stress distribution, i.e. $\sum_{x,y}|D_{kxy}| \gg 0$ and $\sum_{x,y}|d_{jxy}| \gg 0$. A positive intra-region profit is produced when the high stress and low stress CLB locations in the region overlap with the low stress and

high stress CLB locations in the accelerator, respectively. In the best case, the non-uniformity of stress in the region is completely canceled by placing the accelerator when $\sum_{x,y} |D_{kxy}| = \sum_{x,y} |d_{jxy}|$, which leads to a intra-region profit of $\sum_{x,y} |D_{kxy}|$. However, when the high stress and low stress CLBs of the region and the accelerator overlap at the same locations, the nonuniform stress distribution in the region would be further worsened, which leads to $-\sum_{x,y} |d_{jxy}|$ intra-region profit in the worst case.

The range of total profit is obtained by adding the exact value of inter-region profit to the range of intra-region profit. Figure 3 shows an example of profit ranges of an accelerator j to be placed in five different regions. The upper bound of Profit_{j1} and Profit_{j3} is less than the lower bounds of Profit_{j2} and Profit_{j4}. It is therefore not necessary to compute to the exact values of Profit_{j1} and Profit_{j3} to determine the maximum profit. In contrast, the profit ranges of Profit_{j2} and Profit_{j4} overlap and their exact values for different diversified configurations of accelerator j need to be computed to determine which configuration-region combination leads to a more uniform stress distribution.

Fig. 3: Ranges of placement profit of an accelerator j in five different regions

6 Implementation Flow

This section explains the overall flow of the generation of diversified configurations, tool integration and computation of stress matrices using the Xilinx tool flow. The Xilinx tools support the `PROHIBIT` placement constraint [50], which prevents the place-and-route tool to use specific resources such as CLBs or BlockRAMs at specified locations[1]. In the following, we employ this constraint to implement diversified configurations for accelerators.

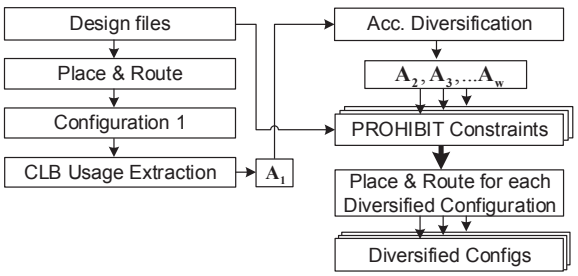

Fig. 4: Generation of diversified configurations using our accelerator diversification method

As shown in Fig. 4, an initial configuration is generated for the accelerator by synthesis and place-and-route of the original design file. From this configuration, the used CLBs are extracted and stored in the matrix $\mathbf{A_1}$. Using Alg. 1, we compute diversified configuration matrices $\mathbf{A_i}$ that specify the diversified CLB usage. They are exported as PROHIBIT placement constraints and then provided to the Xilinx place-and-route tools. The result is the set of diversified configurations for which finally the stress of transistors in CLBs is estimated to construct the stress matrices.

Fig. 5 shows the stress estimation flow for accelerator configurations. To obtain the unit stress of an accelerator configuration, the placed-and-routed configuration and its input signal activities (toggle rate and average duty cycle) are fed to Xilinx XPower that computes the signal activity of every wire in the accelerator. The wires are then matched to the CLB inputs to obtain the input signal activities of every look-up table (LUT) in the CLBs used by the accelerator configuration. Based on the signal activity propagation through a transistor-level LUT model, the toggle rate and stress duty cycle of the LUT transistors are calculated.

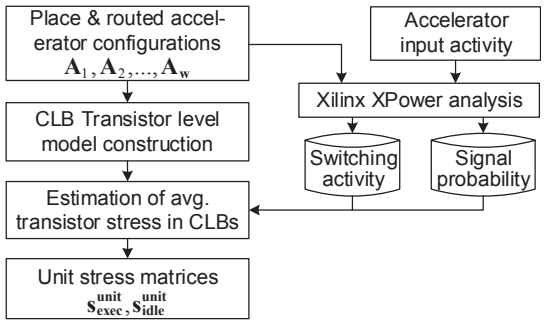

Fig. 5: Stress Estimation Flow

Out of several prospects [9, 51, 52], we model the LUTs in CLBs as 2-input multiplexer (MUX) cells. The configuration SRAM cells are not on the critical path of accelerators during logic operations, because logic transitions in SRAM cells only happen when they are reconfigured. Therefore, aging in configuration SRAM cells is not explicitly targeted here. Our LUT model considers 6-input LUTs that consist of trees of 2-input multiplexers similar to [51]. All SRAM configuration cells of a LUT are connected to MUX data inputs and the LUT inputs are connected to the select signals of MUXes in their respective level of the tree. The evaluation is then performed for each LUT in a two-stage approach by a level-by-level probabilistic analysis given the LUT configuration bit string and the LUT input activities.

For the calculation of the internal signal probabilities, the signal values at the MUX data inputs are weighted according to the duty time of the corresponding select input. Hence, for a MUX with input values v_0, v_1 and select signal sel, the output value is calculated by: $v_{out} := v_0 \cdot P[\overline{sel}] + v_1 \cdot P[sel]$, where $P[sel]$ is the probability that $sel = 1$. Once the output values of all multiplexers (and hence the inputs of each succeeding MUX) are determined, the calculation of the toggle propagation is performed.

In the toggle analysis, we distinguish two types of switching sources as shown in Fig. 6: (a) *propagated* toggles that are fed in through the MUX data inputs, and (b) *generated* toggles that spawn by changing the select signal. In our LUT model, the data inputs of the multiplexers on the first level of the tree are connected to the configuration bits. Thus, upon a select signal switch, toggles can only be generated (if the two configuration bits have different values), but not propagated.

1. Currently the PROHIBIT constraint is not effective/supported for routing resources.

On succeeding stages, the propagation of generated toggles then takes into account the switching activity at all of the input signals: Again, all sources of the toggles to be propagated from data inputs are weighted according to the signal probability at the MUX select input: $t_{prop} := t_0 \cdot P[\overline{sel}] + t_1 \cdot P[sel]$, where t_0 and t_1 are the toggle counts of the data inputs. As for the generated toggles, we consider the likelihood of spawning a toggle after a select input switch as the XOR of the two data inputs multiplied by the toggle frequency f_{sel} as $t_{gen} := f_{sel} \cdot (v_0 \oplus v_1)$. The total toggle count at each MUX output is the sum of the propagated toggles and the toggles generated in its instance: $t_{tot} := t_{prop} + t_{gen}$.

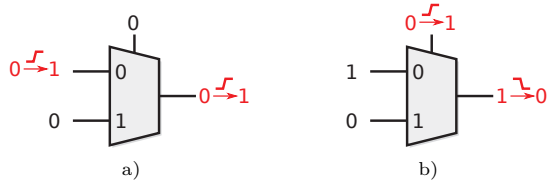

Fig. 6: Toggle propagation (a) and generation (b) in MUXes.

We assume that each MUX is composed of pass transistors, i.e. the input signals can be directly mapped to the respective transistor terminals to obtain static stress [25]. Similarly, the number of switches at each transistor was derived from the toggle activity to obtain the unit stress matrices of the dynamic stress [25].

7 Experimental Evaluation

The presented methods are evaluated in a reconfigurable architecture with eight reconfigurable regions implemented on a Xilinx Virtex-5 LX110T FPGA. Our method performs optimizations on CLB granularity. To evaluate the actual stress for each transistor, we use a transistor-level model of LUTs similar to [6, 53] by representing them as trees of two-input multiplexers (see Section 6). A complex H.264 video encoder (with nine distinct accelerators), an ADPCM audio encoder (one accelerator), an AES encryption (one accelerator) and a JPEG image decoder (three accelerators) were chosen as target applications to represent different computational requirements. Table 1 shows all accelerators along with their logic utilizations (2nd column) and bitstream sizes (3rd column). The total bitstream size of all configurations that need to be stored in the system memory is about 1636 KB. To accommodate the accelerator resource requirement, each reconfigurable region has a size of 4×20 CLBs with eight 6-input LUTs per CLB. The only exceptions are the JPEG accelerators that use 8×20 CLBs for their reconfigurable regions. They would have fit into 4×20 CLB regions when using DSP blocks. But as the transistor-level structure of DSP blocks is not known to us we cannot evaluate stress balancing for DSP blocks and thus we decided to implement JPEG using CLBs only. A minimal set of diversified configurations to tolerate any single-CLB fault is generated for each accelerator (4th column).

For our proposed method we assume a fault model at CLB granularity. A CLB is considered faulty if any faulty behavior in its internal structure makes it unusable for functional operation. We are not limited to a specific fault model (e.g. stuck-at faults), but we assume that the position of faulty CLBs is

TABLE 1: Properties of reconfigurable accelerators and their change in maximum frequency of diversified configurations

Accelerator	CLB utilization [%]	Bitstream [KByte]	#Divers. Config.	Original [MHz]	AccDiv [MHz]	Worstcase Δ [%]
Clip3	66	30	3	133	119–133	10.5
CollapseAdd	23	30	2	158	142–158	10.1
LF_BS4	48	30	2	121	117–121	3.3
LF_Cond	23	30	2	146	139–146	4.8
PointFilter	65	30	3	89	81–89	9.0
QuadSub	9	30	2	257	217–257	15.6
SADrow_4	38	30	4	100	99–103	1.0
SAV	33	30	2	139	122–139	12.2
Transform	45	30	2	167	160–167	4.2
AdpcmEncDec	84	30	7	67	63–67	5.6
AesLutEnc	53	30	3	269	258–269	3.7
JpegTransform	59	52	3	108	98–108	8.8
Jpegidcte	69	52	4	156	135–156	13.1
Jpegidcto	83	52	6	158	149–160	5.5
System freq.				67	63	5.6

detected and diagnosed (as for instance done in [31]) and given as input to our method. To investigate the system behavior under the presence of faulty CLBs, we randomly select 1 up to 80 CLBs (from all CLBs of all regions) to be faulty. To ensure correct operation, any accelerator configuration that requires at least one of the faulty CLBs is not allowed to be configured anymore. For a given number of faulty CLBs, 100 simulation runs are executed with randomly selected faulty CLBs.

A SystemC-based cycle-accurate architectural simulator is used to evaluate the system behavior in the presence of different number of detected permanent faults and using different runtime strategies. It accurately models the hardware implementation of the reconfigurable architecture including the bus arbitration in the reconfigurable fabric, the duration of reconfiguration, and the application behavior including request arbitration for accelerator configuration and software-emulation of unavailable accelerators. The proposed accelerator placement method is integrated into the runtime system and the stress distribution is optimized for dynamic stress. For comparison, a baseline and two state-of-the-art strategies (summarized in Table 2) are evaluated:

- The baseline strategy does not perform fault-tolerant and stress-aware placement and each accelerator has only one configuration, i.e. without diversified configurations. When the algorithm decides to place an accelerator into a region where one of the required CLBs is faulty, then the accelerator will not be configured and its functionality will be emulated in software.
- The stress-aware accelerator placement method STRAP [25] distributes the stress of accelerators uniformly into all reconfigurable resources, where accelerators do not have diversified configurations. We integrated our proposed fault-tolerant placement method (see Section 5.2) into it to evaluate the effectiveness of fault tolerance and stress distribution when no accelerator diversification is employed.

TABLE 2: Compared strategies in the experiments

Strategy	Fault-tolerance	Stress-aware	Acc. Div.
Baseline	No	No	No
STRAP [25]	Yes	Yes	No
Angermeier [23]	No	Yes	No
This work	Yes	Yes	Yes

- Angermeier *et al.* [23] proposed another state-of-the-art stress distribution method which considered the peak stress of regions to place an accelerator (see Section 2), but it is not capable to tolerate faults.

As proposed for our methods, we also extended [23, 25] to replace an accelerator if its reconfigurable region has not been reconfigured for 20 million cycles to provide a fair comparison (see Section 5.1).

7.1 Accelerator Diversification

Since the accelerator diversification method applies additional constraints to prohibit certain CLB location during place-and-route, the maximally achievable frequency of an accelerator may be affected. Table 1 reports the maximal frequency of the diversified configurations of each accelerator (6th column) and of the original configuration (5th column) that is place-and-routed without prohibit constraints. The worst-case frequency impact (7th column) compares the slowest configuration of the accelerator to the original configuration. The place-and-route tool is given a target frequency of 250 MHz as timing constraint to obtain the maximum operating frequency of each accelerator. The frequency of one of the diversified configuration may actually be better than the original configuration, e.g. in the case of SADrow_4 and Jpegidcto, because the additional placement constraints may actually help the place-and-route tool to explore new placement and routing possibilities that are undiscovered during the generation of the original configuration. The maximum system frequency is however limited by the accelerator with the longest critical path, i.e. AdpcmEncDec, which runs at 67.2 MHz with the original configuration and at 63.4 MHz with the slowest configuration, which leads to 5.6% decrease of the system frequency. The original configuration is one of the diversified configurations and thus can be used when full performance is required.

7.2 Aging Resilience and Fault Tolerance

To evaluate the mean-time-to-failure (MTTF) improvement due to stress reduction, we employ a state-of-the-art physics-based HCI aging model. It is adopted from [48, 49] and its details (e.g. aging and temperature parameters) can be found in the Appendix of Ref. [25]. Table 3 reports the dynamic stress reduction for different benchmark applications in a fault-free system and Table 4 reports the resulting MTTF increase. On average, our methods achieve 6.8× higher MTTF than the baseline and 1.6× higher than the closest competitor.

We investigated in detail the impact of faults in the fabric to the application performance and the peak stress. We chose H.264 video encoder as the target application since it stands as a complex application for reconfigurable architectures.

Fig. 7 shows the application performance in the presence of faults in the reconfigurable fabric when different strategies are applied. The box plots (whose values refer to the left Y-axis) show the statistical distribution of the performance degradation w.r.t. a fault-free baseline system, which is measured by

$$\frac{\text{Execution time in the presence of } n \text{ faulty CLBs}}{\text{Execution time in a fault-free baseline system}} \quad (16)$$

It represents how many times *slower* the application runs than it would run in a fault-free system. If only one or two CLBs are detected faulty, the application performance is typically not affected, as these faulty CLBs are not required by the accelerators. When the number of faulty CLBs increases in the baseline system, fewer accelerators can be placed and computationally intensive parts of the application have to be executed in software, which significantly degrades the performance. In extreme cases, e.g. with 80 faulty CLBs, the application is completely executed in software without acceleration, which leads to more than 22× degraded performance.

TABLE 3: Reduction of maximum transistor toggle rate [%]

Strategy	H.264	ADPCM	JPEG	AES	Avg.
Angermeier [23]	49.5	87.3	63.9	49.7	62.6
STRAP [25]	66.0	87.3	73.6	49.7	69.2
This work	78.5	90.8	87.3	73.5	82.5

TABLE 4: MTTF improvement [×] (e.g. 2× improvement means the MTTF is doubled)

Strategy	H.264	ADPCM	JPEG	AES	Avg.
Angermeier [23]	2.0	7.9	2.8	2.0	3.7
STRAP [25]	2.9	7.9	3.8	2.0	4.1
This work	4.7	10.9	7.9	3.8	6.8

With accelerator diversification and fault-tolerant accelerator placement methods, the application experiences less than 8% performance degradation with 1 to 8 faulty CLBs, since these faulty CLBs are avoided by using diversified configurations. As faults accumulate (e.g. more than 25 faulty CLBs), they are not tolerable any more by the diversified configurations, which leads to the increased performance degradation. Without accelerator diversification, STRAP [25] delivers limited capability of fault tolerance. The application performance degrades in a similar rate to the baseline system after the number of faulty CLBs exceeds 7.

The line plots in Fig. 7 (with values referring to the right Y-axis) show the average performance gain of a runtime strategy w.r.t. the baseline system, given the same number of faulty CLBs. It is measured by

$$\frac{\text{Avg. Exec. time in baseline with } n \text{ faulty CLBs}}{\text{Avg. Exec. time in other strategy with } n \text{ faulty CLBs}} \quad (17)$$

where the average is over all simulation runs for a given number of faulty CLBs. This metric measures the ability of a strategy to provide fault-tolerance. With our proposed methods, the application is able to deliver 1.9–3.7× the performance of a baseline system in the presence of 4 to 40 faulty CLBs. As the number of faulty CLBs increases, our methods gracefully degrades the system performance until it converges to the baseline system. Without accelerator diversification, the stress-aware placement method [25] delivers up to 1.5× performance gain for fewer than 10 faulty CLBs. Angermeier *et al.* [23] optimize for stress reduction and do not consider fault-tolerance during accelerator placement. They place an accelerator into a region that results in the lowest peak stress. However, this may prevent the successful placement of other accelerators, which leads to lower application performance.

The box plots (left Y-axis) in Fig. 8 shows the statistical distribution of peak dynamic stress in the reconfigurable fabric in the presence of different number of CLB faults. When there are only a few (e.g. fewer than 6) faulty CLBs in the

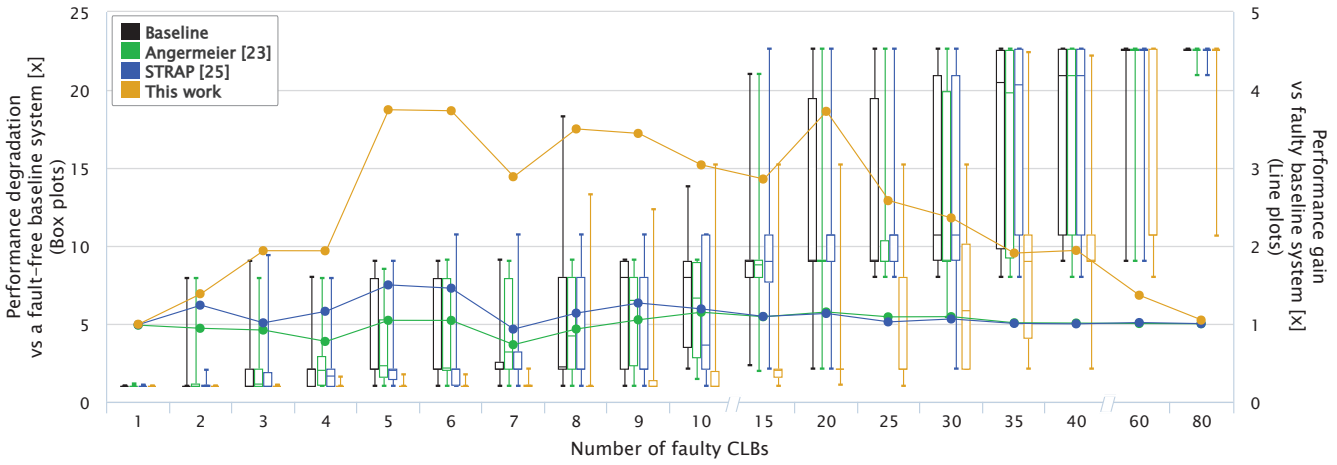

Fig. 7: Application performance in the presence of faults under different strategies. Left Y-axis (box plots): performance degradation w.r.t. a fault-free baseline system. Right Y-axis (line plots): performance gain w.r.t. to the faulty baseline system

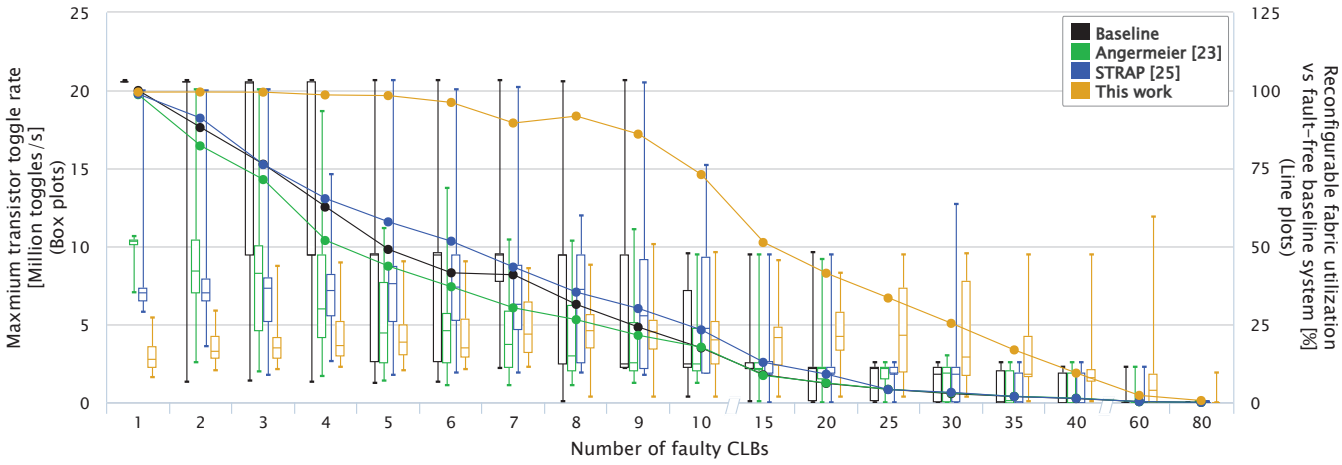

Fig. 8: Peak stress and utilization in the reconfigurable fabric in the presence of faults. Left Y-axis (box plots): maximum transistor toggle rate. Right Y-axis (line plots): utilization of the reconfigurable fabric for acceleration w.r.t. a fault-free baseline system

reconfigurable fabric, accelerators have a higher placement freedom. The runtime system can freely choose into which region an accelerator shall be placed such that the maximum stress is minimized. All stress-aware strategies including [23, 25] and our proposed methods avoid the stress accumulation in individual CLBs and result in a lower peak stress than the baseline system. Angermeier *et al.* [23] performs only inter-region stress distribution and thus produces higher peak stress than other stress-aware strategies in systems with less faults (e.g. 1 to 3 faulty CLBs), i.e. high placement freedom.

Combining accelerator diversification and intra- and inter-region stress distribution, our methods produce the most uniform stress distribution in the fabric, which leads to the lowest peak stress compared to other approaches. The placement freedom diminishes as more CLBs become faulty and the stress distribution becomes ineffective. As can be seen from the resulting peak stress, STRAP [25] does not have clear advantage over the baseline system when the fabric has 6 to 10 faulty CLBs. As the number of faulty CLBs increases further (e.g. 15 to 60 faulty CLBs), the resulting peak stress from our proposed methods is higher than that from other strategies. The reason behind that is, that other strategies are not capable to tolerate these faults and are not able to find feasible placements of accelerators into the faulty regions, i.e. the resources in the reconfigurable fabric are less stressed by the execution of accelerators. This can be seen by looking at the performance degradation in Fig. 7, where computations are more frequently emulated in software instead of being executed on the reconfigurable fabric.

The line plots (right Y-axis) in Fig. 8 show the utilization of the reconfigurable fabric for acceleration w.r.t. a fault-free baseline system. The *fabric utilization* is defined as follows:

$$\frac{\text{Exec. time in the reconfigurable fabric (accelerated)}}{\text{Total execution time}} \tag{18}$$

The values plotted are

$$\frac{\text{Average utilization in the presence of faults}}{\text{Utilization in the fault-free baseline system}} \tag{19}$$

The average is over all simulation runs for a given number of faulty CLBs. The fault-free baseline system has a fabric utilization value of 75.0%. It means that 75% of the total execution time is spent in the execution of accelerators in the reconfigurable fabric while the rest 25% is on the processor pipeline. Since the fault-free baseline system is optimized for performance which fully utilizes the reconfigurable fabric, 75% is the maximum utilization value that can be achieved. In the

presence of faults, requested accelerators may not be placed into the reconfigurable fabric due to faults, which leads to a reduced fabric utilization and less CLB stress. In extreme case (e.g. with 60 or 80 faulty CLBs) the reconfigurable fabric is not used at all as almost all accelerator functions are emulated in software. With our fault tolerance with diversified configurations, nearly full fabric utilization is achieved for less than 7 faulty CLBs. Even at a high amount of faults (e.g. 15 to 35 faulty CLBs), the reconfigurable fabric is still used for acceleration, as can be seen from the induced stress. At around 25% full fabric utilization, the application delivers more than 2× higher performance than other strategies (see Fig. 7).

For the evaluated systems, the worst-case overhead of the accelerator placement algorithm occurs when 8 accelerators need to be placed into 8 regions and the exact values of intra-region profits of all configuration-region combinations are calculated. These calculations only take 1.3 ms on a SPARC V8 LEON3 processor running at 100 MHz.

8 Conclusion

The dependable operation of runtime reconfigurable architectures is threatened by aging. This article presented novel methods to mitigate aging and tolerate emerging faults in the reconfigurable fabric, by combining 1) diversified accelerator configurations so that CLB faults can be tolerated by at least one configuration per accelerators, and 2) an accelerator placement algorithm to balance application-induced stress both within a reconfigurable region as well as across all reconfigurable regions in the system. The runtime placement algorithm takes the faulty resources into account.

With our accelerator diversification and fault-tolerant accelerator placement, the application experiences less than 8% performance degradation with 1 to 8 faulty CLBs, since these faulty CLBs are avoided by using diversified configurations. Using the proposed methods, an H.264 video application delivers from 1.9× up to 3.7× the performance of a baseline system in presence of 4 to 40 faulty CLBs. For a set of benchmark applications in a fault-free system, the reduction in dynamic stress by aging mitigation leads to 6.8× higher MTTF than the baseline system on average and 1.6× higher MTTF than the closest competitor.

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] J. Cardoso and M. Hübner, *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*. Springer, 2011.

[2] A. Putnam, A. Caulfield, E. Chung *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services", in *International Symposium on Computer Architecture*, June 2014.

[3] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond", *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, March 2015.

[4] V. Boppana, S. Ahmad, I. Ganusov *et al.*, "Xilinx 16nm UltraScale+ MPSoC and FPGA families", in *Hot Chips 27: A Symposium on High Performance Chips*, 2015.

[5] P. K. Gupta, "Xeon+FPGA platform for the data center", in *Workshop on the Intersections of Computer Architecture and Reconfigurable Logic*, vol. 119, 2015.

[6] E. A. Stott, J. S. Wong, P. Sedcole *et al.*, "Degradation in FPGAs: Measurement and modelling", in *Int'l Symposium on Field Programmable Gate Arrays (FPGA)*, 2010, pp. 229–238.

[7] X. Guo, W. Burleson, and M. Stan, "Modeling and experimental demonstration of accelerated self-healing techniques", in *Design Automation Conference (DAC)*, 2014.

[8] X. Li, J. Qin, and J. Bernstein, "Compact modeling of MOSFET wearout mechanisms for circuit-reliability simulation", *IEEE Trans. Device and Materials Rel.*, vol. 8, no. 1, pp. 98–121, 2008.

[9] S. Srinivasan, R. Krishnan, P. Mangalagiri *et al.*, "Toward increasing FPGA lifetime", *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 2, pp. 115–127, 2008.

[10] J. Henkel, L. Bauer, N. Dutt *et al.*, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends", in *Design Automation Conference (DAC)*, 2013.

[11] J. Henkel, L. Bauer, J. Becker *et al.*, "Design and architectures for dependable embedded systems", in *Int'l Conf. on HW/SW Codesign and System Synth. (CODES+ISSS)*, 2011, pp. 69–78.

[12] C. Beckhoff, D. Koch, and J. Torresen, "Portable module relocation and bitstream compression for Xilinx FPGAs", in *International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2014, pp. 1–8.

[13] R. Backasch, G. Hempel, S. Werner *et al.*, "Identifying homogenous reconfigurable regions in heterogeneous FPGAs for module relocation", in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2014, pp. 1–6.

[14] A. Ahmadinia, C. Bobda, S. P. Fekete *et al.*, "Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices", *IEEE Transactions on Computers (TC)*, vol. 56, no. 5, pp. 673–680, 2007.

[15] C. Patterson, P. Athanas, M. Shelburne *et al.*, "Slotless module-based reconfiguration of embedded FPGAs", *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 1, pp. 6:1–6:26, Oct. 2009.

[16] L. Bauer, M. Shafique, and J. Henkel, "A computation- and communication- infrastructure for modular special instructions in a dynamically reconfigurable processor", in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2008, pp. 203–208.

[17] E. Stott and P. Cheung, "Improving FPGA reliability with wear-levelling", in *International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2011, pp. 323–328.

[18] A. Bsoul, N. Manjikian, and L. Shang, "Reliability- and process variation-aware placement for FPGAs", in *Design, Automation and Test in Europe Conference (DATE)*, 2010, pp. 1809–1814.

[19] P. M. B. Rao, A. Amouri, S. Kiamehr *et al.*, "Altering LUT configuration for wear-out mitigation of FPGA-mapped designs", in *Int'l Conf. on Field prog. Logic and Applications*, 2013, pp. 1–8.

[20] E. Stott, J. Wong, and P. Cheung, "Degradation analysis and mitigation in FPGAs", in *International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 428–433.

[21] H. Zhang, L. Bauer, M. A. Kochte *et al.*, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures", in *Int'l. Test Conf. (ITC)*, 2013, paper 14.1.

[22] Z. Ghaderi and E. Bozorgzadeh, "Aging-aware high-level physical planning for reconfigurable systems", in *Asia and South Pacific Design Autom. Conf. (ASP-DAC)*, 2016, pp. 631–636.

[23] J. Angermeier, D. Ziener, M. Glass *et al.*, "Stress-aware module placement on reconfigurable devices", in *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2011, pp. 277–281.

[24] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems", *IEEE Design & Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[25] H. Zhang, M. A. Kochte, E. Schneider *et al.*, "STRAP: Stress-aware placement for aging mitigation in runtime reconfigurable architectures", in *International Conference on Computer-Aided Design (ICCAD)*, November 2015, pp. 38–45.

[26] M. Renovell, J. Portal, J. Figueras *et al.*, "Test pattern and test configuration generation methodology for the logic of RAM-based FPGA", in *Asian Test Symp. (ATS)*, 1997, pp. 254–259.

[27] M. Tahoori, "Application-dependent testing of FPGAs", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 14, no. 9, pp. 1024–1033, 2006.

[28] W. K. Huang, F. J. Meyer, X.-T. Chen *et al.*, "Testing configurable LUT-based FPGA's", *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 6, no. 2, pp. 276–283, 1998.

[29] 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.
[30] 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 *International Conference on Embedded Systems and Applications (ESA)*, 2006, pp. 87–93.
[31] M. Abramovici, C. Stroud, C. Hamilton *et al.*, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications", in *Int'l Test Conf.*, 1999, pp. 973–982.
[32] J. Emmert, C. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks", *IEEE Trans. Very Large Scale Integration Systems (TVLSI)*, vol. 15, no. 2, pp. 216–226, 2007.
[33] M. S. Abdelfattah, L. Bauer, C. Braun *et al.*, "Transparent structural online test for reconfigurable systems", in *International On-Line Testing Symposium (IOLTS)*, June 2012, pp. 37–42.
[34] L. Bauer, C. Braun, M. E. Imhof *et al.*, "Test strategies for reliable runtime reconfigurable architectures", *IEEE Transactions on Computers (TC)*, vol. 62, no. 8, pp. 1494–1507, 2013.
[35] C. Stroud, E. Lee, and M. Abramovici, "BIST-based diagnostics of FPGA logic blocks", in *Int'l Test Conf.*, 1997, pp. 539–547.
[36] T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal fault diagnosis for lookup table FPGAs", *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 39–44, 1998.
[37] M. Abramovici, C. E. Stroud, and J. M. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks", *IEEE Trans. Very Large Scale Integration Systems (TVLSI)*, vol. 12, no. 12, pp. 1284–1294, 2004.
[38] A. Alzahrani and R. F. DeMara, "Fast online diagnosis and recovery of reconfigurable logic fabrics using design disjunction", *IEEE Trans. on Computers (TC)*, vol. PP, no. 99, pp. 1–1, 2016.
[39] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems", *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 6, no. 2, pp. 212–221, 1998.
[40] A. Kanamaru, H. Kawai, Y. Yamaguchi *et al.*, "Tile-based fault tolerant approach using partial reconfiguration", in *Int'l Workshop on Reconfigurable Computing (ARC)*, 2009, pp. 293–299.
[41] W.-J. Huang and E. J. McCluskey, "Column-based precompiled configuration techniques for FPGA", in *Symp. on Field-Progr. Custom Computing Machines (FCCM)*, 2001, pp. 137–146.
[42] S. Mitra, W.-J. Huang, N. Saxena *et al.*, "Reconfigurable architecture for autonomous self-repair", *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 228–240, 2004.
[43] M. Psarakis and A. Apostolakis, "Fault tolerant FPGA processor based on runtime reconfigurable modules", in *European Test Symposium (ETS)*, 2012, pp. 1–6.
[44] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 2, pp. 203–215, 2007.
[45] A. N. Nowroz and S. Reda, "Thermal and power characterization of field-programmable gate arrays", in *Int'l Symposium on Field Programmable Gate Arrays (FPGA)*, 2011, pp. 111–114.
[46] S. Velusamy, W. Huang, J. Lach *et al.*, "Monitoring temperature in FPGA based SoCs", in *International Conference on Computer Design (ICCD)*, 2005, pp. 634–637.
[47] Y. Cao, J. Velamala, K. Sutaria *et al.*, "Cross-layer modeling and simulation of circuit reliability", *IEEE Trans. on CAD of ICs and Systems (TCAD)*, vol. 33, no. 1, pp. 8–23, 2014.
[48] H. Amrouch, V. M. van Santen, T. Ebi *et al.*, "Towards interdependencies of aging mechanisms", in *International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 478–485.
[49] C. Ma, H. Mattausch, M. Miyake *et al.*, "Compact reliability model for degradation of advanced p-MOSFETs due to NBTI and hot-carrier effects in the circuit simulation", in *Reliability Physics Symposium (IRPS)*, 2013, pp. 2A.3.1–2A.3.6.
[50] Xilinx, *Constraints Guide (UG625, v. 13.4)*, 2012.
[51] S. Kiamehr, A. Amouri, and M. Tahoori, "Investigation of NBTI and PBTI induced aging in different LUT implementations", in *Int'l Conf. on Field-Progr. Technology (FPT)*, 2011, pp. 1–8.
[52] E. Stott, P. Sedcole, and P. Cheung, "Modelling degradation in FPGA lookup tables", in *International Conference on Field-Programmable Technology (FPT)*, 2009, pp. 443–446.
[53] T. Pi and P. Crotty, "FPGA lookup table with transmission gate structure for reliable low-voltage operation", 2004, US Patent 6,809,552 Xilinx, Inc.

Hongyan Zhang received the M.Sc. degree in electrical engineering and information technologies from the Karlsruhe Institute of Technology in 2011. He joined the Chair for Embedded Systems at the Karlsruhe Institute of Technology in 2011. His research interests include fault tolerant and reliable runtime reconfigurable architectures.

Lars Bauer received the M.Sc. and Ph.D. degrees in computer science from the University of Karlsruhe, Germany, in 2004 and 2009. He is currently a research group leader and lecturer at the Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT). Dr. Bauer received two dissertation awards (EDAA and FZI), two best paper awards (AHS'11 and DATE'08) and several nominations. His research interests include architectures and management for adaptive multi-/many-core systems.

Michael A. Kochte received a diploma in computer science in 2007 and a Dr. rer. nat. (Ph.D.) from the University of Stuttgart, Germany, in 2014. He is currently with the Institute for Computer Architecture and Computer Engineering of the University of Stuttgart and leads the research group for Dependable Hardware. Dr. Kochte received a best dissertation award and two best paper awards (DELTA'08, JETTA'14). His research interests include reconfigurable computing, hardware test and reliability, and hardware security.

Eric Schneider received the diploma degree in computer science (Dipl.-Inf.) from the University of Stuttgart, Germany, in 2012. There he joined the Institute of Computer Architecture and Computer Engineering (ITI), where he is currently working towards his Ph.D. His research interests include accelerated circuit simulation on Graphics Processing Units (GPUs), circuit test and diagnosis.

Hans-Joachim Wunderlich received the diploma degree in mathematics from the University of Freiburg, Germany, in 1981 and the Dr. rer. nat. (Ph.D. degree) 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, Germany. He is editor of various international journals and program committee member of a variety of IEEE conferences on design and test of electronic systems. He has published 11 books and book chapters and more than 250 reviewed scientific papers in journals and conferences. His research interests include test, reliability, and fault tolerance of microelectronic systems. He is a fellow of the IEEE.

Jörg Henkel received his Ph.D. from Braunschweig University with *Summa cum Laude*. He is currently directing the Chair for Embedded Systems (CES). He is the Chairman of the IEEE Computer Society (Germany Section), the Editor-in-Chief of the IEEE Design & Test Magazine, and was the Editor-in-Chief of the ACM Transactions on Embedded Computing Systems (ACM TECS) for two consecutive terms. He holds ten US patent, is a Fellow of the IEEE, has given around ten keynotes at various international conferences, and has received several Best Paper/Poster Awards at DAC, DATE, ICCAD, CODES+ISSS, etc.