Module Diversification: Fault Tolerance and Aging Mitigation for Runtime Reconfigurable Architectures

Hongyan Zhang*, Lars Bauer*, Michael A. Kochte‡, Eric Schneider‡, Claus Braun‡, Michael E. Imhof‡, Hans-Joachim Wunderlich‡ and Jörg Henkel*
*Chair for Embedded Systems, Karlsruhe Institute of Technology, Karlsruhe, Germany
‡Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany

***Abstract*—Runtime reconfigurable architectures based on Field-Programmable Gate Arrays (FPGAs) are attractive for realizing complex applications. However, being manufactured in latest semiconductor process technologies, FPGAs are increasingly prone to aging effects, which reduce the reliability of such systems and must be tackled by aging mitigation and application of fault tolerance techniques.**

This paper presents *module diversification*, a novel design method that creates different configurations for runtime reconfigurable modules. Our method provides fault tolerance by creating the *minimal* number of configurations such that for any faulty Configurable Logic Block (CLB) there is at least one configuration that does not use that CLB. Additionally, we determine the fraction of time that each configuration should be used to balance the stress and to mitigate the aging process in FPGA-based runtime reconfigurable systems. The generated configurations significantly improve reliability by fault-tolerance and aging mitigation.

***Keywords*-Reliability, online test, fault-tolerance, aging mitigation, partial runtime reconfiguration, FPGA**

I. Introduction

Today's *Field-Programmable Gate Arrays* (FPGAs) allow to reconfigure selected regions of the FPGA's fabric at runtime without affecting operation in other regions by support of *partial runtime reconfiguration*. This provides a high degree of flexibility and efficiency, as the FPGA can be optimized for the requirements of any application at any point in time. This flexibility is for instance used in commercially available high-performance computing systems to provide hardware accelerators for the executing application [1]. Especially area, cost, and power constraint embedded systems benefit from the large potential of runtime reconfiguration. For instance, software defined radios can be realized by reconfiguring the wireless protocol into the FPGA at runtime [2] to support a large number of protocols in a small area footprint. Similarly, different encryption standards can be provided on demand using runtime reconfiguration [3]. In systems with diverse functional requirements such as vision-based robotics or automotive [4], runtime reconfiguration is used to provide hardware acceleration for those tasks that have a high priority at a certain point in time. An overview of further embedded applications benefiting from runtime reconfigurable architectures is provided in [5].

In all these examples, the flexibility of runtime reconfiguration is used to build architectures that are able to instantiate hardware accelerators in the reconfigurable fabric according to runtime application requirements. In the following, a region of the FPGA that can be reconfigured at runtime is called *container* and the design that is reconfigured into a container is called *module*.

Modern FPGAs are often manufactured in latest semiconductor process technologies (e.g. 28 nm for Xilinx Virtex-7 and Altera Stratix V) and thus are increasingly prone to various sources of defects [6]. Aging mechanisms like *Time-Dependent Dielectric Breakdown* (TDDB), *Hot Carrier Injection* (HCI), *Negative Bias Temperature Instability* (NBTI), and *Electromigration* have severe impact on FPGA hardware structures [7]. The rate of degradation due to aging effects depends on different *stress* factors such as temperature, switching activity, or electric field strengths in the devices [8]. Systems must be capable of coping with aging effects, variations, and latent defects in the reconfigurable fabric, which can cause unrecoverable permanent faults. These reliability threats are further aggravated by long operation time and harsh environmental conditions (e.g. temperature). To ensure reliable operation of the reconfigurable fabric and prolong the lifetime of the device, aging mitigation, fault detection, and fault tolerance techniques need to be an integral part of runtime reconfigurable architectures.

This paper presents a novel design method called *module diversification* to 1) tolerate permanent and intermittent faults and 2) mitigate the aging process. In general terms, for each module/container pair, a set of configurations is generated that is diversified in terms of the usage of *Configurable Logic Blocks (CLBs)* such that for every CLB in the container, there always exists a configuration that does not require that CLB.

We consider a CLB as faulty if it is affected by one or multiple permanent or intermittent faults. We do not distinguish the type of faults within a CLB. The proposed module diversification enables the system to tolerate at least any single CLB-fault. The detection and localization of a faulty CLB is not the scope of this paper, but can be performed by structural testing and diagnosis at CLB granularity [9, 10]. If a fault is localized in a container, a diversified configuration of a module can be loaded (i.e. reconfigured to the container at runtime) that does not use the faulty CLB.

This paper provides a general algorithm to generate the minimal set of configurations for tolerating single CLB-faults and additional configurations for multi-fold CLB-fault tolerance. The relationship between the required number of configurations, amount of spare resources, and reliability is investigated in this paper. In addition, since the number of these configurations shall be as small as possible to reduce storage overhead, these alternative configurations are

978-1-4799-0859-2/13/$31.00

inherently highly diversified, i.e. the number of common CLBs of two different configurations is as small as possible.

Loading new configurations on demand is readily supported by architectures that use runtime reconfiguration as part of their regular operation. They provide means to tolerate temporary unavailability of a module. For instance, runtime reconfigurable processors that use the reconfigurable fabric to implement so-called *Special Instructions* typically use an alternative software implementation of the Special Instruction when the hardware module is not reconfigured [11, 12].

In addition to tolerating faults, module diversification can be used to mitigate the aging process. The aging process of a CLB can be slowed down by reducing its stress duty cycle. The inherent diversity of the generated configurations for fault tolerance allows to *balance the stress on all CLB resources* and thus mitigate the aging process in the reconfigurable fabric by loading these configurations in an alternating manner. The aging-related stress values of CLBs in each configuration are estimated after synthesis and place-and-route. Whenever a hardware module is loaded into a container, an appropriate diversified configuration for that module is selected, so that stress is distributed to all CLBs in the container and not concentrated on a single or a few CLBs. Therefore, stress balancing is fully transparent to the normal operation of the reconfigurable architecture.

This paper formulates and optimally solves the problem of balancing the stress on CLBs using diversified configurations as a linear programming problem. Thus, the *optimal scheduling policy of configurations* is determined, i.e. how long a configuration shall be used relative to other configurations.

The proposed module diversification design method focuses on permanent and intermittent CLB-faults as well as aging mitigation and is orthogonal to established methods for soft error mitigation.

Contribution of this paper:

1) The *module diversification* design method.
2) An algorithm that generates the minimal set of configurations to tolerate all single CLB-faults.
3) An optimal schedule of diversified configurations for aging mitigation by stress balancing.

Paper structure: Section II gives an overview of the related work in the field of reconfigurable architectures and fault tolerant systems. Section III describes the module diversification design method and the reliability analysis of generated configurations. Section IV presents the stress balancing method using the diversified configurations. The overall implementation flow of module diversification is shown in Section V. Section VI presents the experimental results on benchmark modules and Section VII concludes the paper.

II. Related Work

Recovery schemes for permanent faults in reconfigurable systems are based on resource remapping. The FPGA is partially reconfigured to an alternative configuration using spare resources after the faults are detected and located so that the faulty resources are no longer used. The reconfiguration is performed either offline or during runtime.

A. Online Test and Diagnosis of Reconfigurable Systems

Online test and diagnosis methods are a prerequisite for fault recovery in reconfigurable systems. The thorough test of the reconfigurable FPGA fabric has been in the focus of research for over 20 years. Application independent testing targets the whole fault universe of the fabric and is not limited to a specific use of the fabric. They typically consist of multiple special test configurations and corresponding test stimuli [13]. In contrast, application dependent tests target only the subset of programmable resources of the FPGA fabric relevant for a particular target application [14].

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 [15].

With the use of partial dynamic reconfiguration in FPGAs, the reconfigurations during test application can be performed by an external or embedded processor at runtime [16–19]. In [18, 19], the Roving STARs (Self Testing AReas) method for online test is introduced which partitions the FPGA into rows and columns which can be either used functionally or tested by an online BIST approach.

The transparent integration of online test methods into runtime reconfigurable architectures was presented in [10, 20]. 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 test configurations to distinguish and localize faults [9, 21, 22].

Similar to the integration of online tests into runtime reconfigurable architectures [20], diagnosis techniques based on diagnostic configurations, stimuli and response evaluation can be integrated and controlled by an embedded processor [17].

B. Fault Tolerance in Reconfigurable Systems

Once a fault is detected and localized, different recovery 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) [23, 24]. In [23] each tile consists of multiple CLBs with one spare CLB. If a CLB in a certain tile is detected to be faulty, an alternative configuration for that tile is loaded which implements the same logic function but utilizes the spare rather than the faulty CLB in order to avoid the fault. In [24] the circuit in the faulty tile is entirely remapped to another spare tile. Column-based approaches apply similar concepts to CLB columns [25, 26], where the reconfigurable fabric is partitioned into a 1-dimensional array of CLB columns. Each column can implement a functional module and in order to provide fault tolerance, intentionally unused columns are introduced as spares. In response to a fault,

a precompiled configuration is loaded where the functional module that resides in the faulty column has been remapped by shifting the functional modules starting from the faulty column towards the next unused spare. Both tile- and column-based approaches need complex customized routing techniques. Tile-based approaches require fixed interfaces between adjacent tiles so that each tile can be reconfigured independently of others. Column-based approaches require online routing after module remapping as the location of the functional modules change and the communication in-between has to be re-established. They also do not maximize the inherent diversity in alternative configurations or exploit it to balance the stress on the reconfigurable fabric.

The Roving STARs based fault tolerance method [19] combines distributed CLB spares and online compilation of configurations to replace faulty CLBs with spares. For complex designs, this online compilation or synthesis may result in unpredictable timing behavior.

Instead, the method proposed here works on CLB-granularity and does not need explicit tile/column-wise partitioning or online synthesis. The CLB placement and routing of the alternative configurations is prepared offline by vendor place-and-route tools.

The authors in [8] introduce a scheme to alleviate aging mechanisms through remapping of a design. This approach is coarse-grained and uses only two different configurations. The configurations are swapped only once after a half-life period of the first failing component and it does not guarantee tolerance to CLB-faults.

A similar approach was introduced in [27], where the authors presented three strategies 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.

The authors of [28] propose the idea of using alternative configurations for reconfigurable modules, each of which uses different CLBs such that any possible single defective CLB is 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.

In this work, we present the method of diversified configurations for single CLB-fault tolerance. It can be implemented using standard vendor tools. The potential of diversified configurations to further tolerate multi-fold CLB faults and mitigate aging for increased reliability is also exploited and evaluated.

III. Module Diversification

A. Design Method

A *module* defines the logic functions to be implemented in a *container* which consists of CLBs that are arranged regularly in a 2-dimensional array in the FPGA. The *configuration* determines which CLBs in the container are used to implement the module. A natural way to describe the CLB usage of a configuration is to use a Boolean matrix. The size of the matrix matches the size of the container: A rectangular container with m CLBs in height and n CLBs in width requires an $m \times n$ matrix. If a CLB is used, the corresponding matrix element is 1, otherwise 0. We call this Boolean matrix a *configuration matrix*. For example, a module configuration using 5 CLBs implemented in a 3×3 container can be represented in a configuration matrix $\mathbf{A}$:

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

The module diversification design method generates a set of configurations, each of which implements the same module function, but uses different CLB resources, such that by loading diversified module configurations at runtime: 1) any single faulty CLB can be tolerated and 2) the stress on CLBs can be balanced in order to mitigate aging.

Formally, we search a set of configurations C for a module implemented in an $m \times n$ container.

$$C = \{\mathbf{A}_1, \cdots, \mathbf{A}_w\},\ \mathbf{A}_k : m \times n \text{ Boolean matrix} \tag{2}$$

Assume that all of these configurations utilize the same amount of CLBs U and there is at least one free CLB, i.e.

$$\forall \mathbf{A}_k \in C : \sum_{i,j} [\mathbf{A}_k]_{i,j} = U < m \cdot n \tag{3}$$

To be able to tolerate any single faulty CLB, this set of configurations must satisfy the *completeness condition*:

$$\begin{gathered} \forall i, j, 1 \le i \le m, 1 \le j \le n : \\ \exists \mathbf{A}_k \in C \text{ such that } [\mathbf{A}_k]_{i,j} = 0 \end{gathered} \tag{4}$$

The completeness condition guarantees that if any CLB is detected to contain faults, there always exists a diversified configuration $\mathbf{A}_k$ that does not require the faulty CLB. This condition can be easily verified by checking whether:

$$\sum_{i,j} \prod_{k=1}^{w} [\mathbf{A}_k]_{i,j} = 0 \tag{5}$$

Eq. 5 states that there is no CLB that is used by all configurations. For an $m \times n$ matrix with $m \cdot n - U$ spares, at least w_{min} configurations are required for the completeness condition:

$$w_{min} = \lceil \tfrac{m \cdot n}{m \cdot n - U} \rceil \tag{6}$$

In each configuration, exactly $m \cdot n - U$ CLBs are spare. For a configuration $\mathbf{A}_k$, at most $m \cdot n - U$ CLBs that were not spare in any of the configurations $\mathbf{A}_l, l < k$ can be spare in $\mathbf{A}_k$, which directly results in this lower bound.

In order to minimize the number of diversified configurations for satisfying the completeness condition and to improve the effect of aging mitigation, we require that the generated

set of configurations also satisfies the *max diversification condition*:

$$\forall k, 1 \leq k \leq w : \exists \mathbf{A}_l \in C, l \neq k \text{ such that}$$
$$\sum_{i,j}([\mathbf{A}_k]_{i,j} \cdot [\mathbf{A}_l]_{i,j}) = \begin{cases} 2U - m \cdot n & \text{if } U > \frac{1}{2}m \cdot n \\ 0 & \text{else} \end{cases} \quad (7)$$

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 module requires at most half of the available CLB resources, or $2U - m \cdot n$, whenever all unused CLBs ($m \cdot n - U$) in one configuration are used in the other configuration. In the latter case, the number of common CLBs is $U - (m \cdot n - U)$. The max diversification condition states that for every configuration $\mathbf{A}_k \in C$ there is at least one other configuration $\mathbf{A}_l$ which differs from $\mathbf{A}_k$ as much as possible w.r.t. the used CLB resources. The *max diversification condition* also ensures that the stress on each CLB can be minimized by loading diversified configurations in an alternating manner.

For example, consider a module requiring 5 CLBs to be implemented in a 3×3 container. The following set of configurations satisfies the completeness condition but does not satisfy the *max diversification condition*:

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{A}_3 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (8)$$

When 5 out of 3×3 CLBs are used, the minimal possible number of common CLBs between two configurations is 1 CLB (see Eq. 7). Yet, in the above 3 configurations, all pairs of configurations have at least 2 CLBs in common. One possible set of configurations that satisfies both conditions is as follows:

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{A}_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (9)$$

As we will see later in Section IV, the first set of configurations causes worse balanced stress than the second one.

To find a maximally diversified set of configurations, enumeration is computationally intractable. For example, if a module requires 80 CLBs in a container with 100 CLBs, then there are $\binom{100}{80} \approx 5.36 \times 10^{20}$ possible configurations. Algorithm 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 $\mathbf{S}$, which has the same size of 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, $\mathbf{S}$ is initialized to $\mathbf{A}_1$, the only element in C at the moment. In line 3, the next new configuration matrix $\mathbf{A}_{new}$ is initialized to the initial configuration matrix. In the inner loop (line 8 to 16), it is further modified by swapping zero- and one-elements. The inner loop iterates through all

Algorithm 1 Generation of diversified configurations C

1. C := $\{\mathbf{A}_1\}$ // $\mathbf{A}_1$ is the initial configuration
2. $\mathbf{S}$:= $\mathbf{A}_1$ // Score matrix $\mathbf{S}$ stores swapping priority of CLBs
3. $\mathbf{A}_{new}$:= $\mathbf{A}_1$
4. **loop**
5. $zero\_elem\_list$:= $\{(i,j) \mid [\mathbf{A}_{new}]_{i,j} = 0\}$ // unused CLBs
6. $candidates\_list$:= $\{(i,j) \mid [\mathbf{A}_{new}]_{i,j} = 1\}$
7. sort $candidates\_list$ according to the value of $\mathbf{S}_{i,j}$ in descending order // first element has the highest score
8. **for all** (i,j) in $zero\_elem\_list$ **do**
9. $swap\_candidates$:= $\{(x,y) \mid (x,y) \in candidates\_list$ and $\mathbf{S}_{x,y} = \mathbf{S}_{candidates\_list[0]}\}$ // all CLBs with the highest score
10. $farthest\_swap\_candidate$:= $(x,y) \in swap\_candidates$ with max. Manhattan distance between (x,y) and (i,j) // farthest elements are swapped first so that CLBs are located near each other and better timing is achieved
11. swap($[\mathbf{A}_{new}]_{i,j}$, $[\mathbf{A}_{new}]_{farthest\_swap\_candidate}$)
12. $candidates\_list$.remove($farthest\_swap\_candidate$)
13. **if** $candidates\_list = \emptyset$ **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. $\mathbf{S} := \mathbf{S} + \mathbf{A}_{new}$ // update CLB score
21. $C := C \cup \{\mathbf{A}_{new}\}$
22. **if** $|C|$ = desired number of configurations $\vee |C| = \binom{m \cdot n}{U}$ **then**
23. break
24. **end if**
25. **end loop**

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 many times in 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 (line 9 to 11) so that in the resulting configuration used CLBs are located near each other.

The first generated $\lceil \frac{m \cdot n}{m \cdot n - U} \rceil$ configurations is 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. In the generation process of the minimal set, the CLBs that are used most by already generated configurations are not included as resources in the next generated configuration $\mathbf{A}_{new}$. The number of common CLBs in $\mathbf{A}_{new}$ and the last generated configuration is always $2U - m \cdot n$ and therefore the *max diversification condition* is guaranteed.

If the user requires more configurations for higher reliability (i.e. tolerate more multi-fold CLB-faults), 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 line 17 to line 19, where

random changes are made to $\mathbf{A}_{new}$ to generate a new unique configuration matrix.

B. Reliability Analysis

The reliability of an entity is the probability that it operates without failure for at least the specified time period t. Let $R_{CLB}(t)$ be the reliability of a CLB at time t. Without any fault-tolerance techniques applied, the reliability of a module using U out of $m \times n$ CLBs is

$$R_{\text{No FT}}(t) = R_{CLB}(t)^U, \tag{10}$$

i.e. all U CLBs are required to be operational to allow the module to operate without failure. Using the proposed module diversification design method, the reliability of the module can be increased: In case of CLB failures, the module can be reconfigured with a diversified configuration such that only operational CLBs are used by the configured module. In this case, the reliability of the module becomes:

$$R_{Div}(t) = R_{CLB}(t)^{m \cdot n} + \sum_{f=1}^{m \cdot n} \underbrace{C_f \alpha_f \binom{m \cdot n}{f} (1 - R_{CLB}(t))^f R_{CLB}(t)^{(m \cdot n - f)}}_{\text{Probability that } f\text{-fold CLB failures can be tolerated}} \tag{11}$$

The first term states the probability that all CLBs are fault free. The second term aggregates all the scenarios where only a single CLB is faulty, two CLBs are faulty, three CLBs are faulty, ..., all CLBs are faulty.

Fault coverage C_f, $0 \leq C_f \leq 1$, is the fraction of f-fold CLB faults which are detected by an online test or concurrent error detection scheme (c.f. Section II-A) such that reconfiguration with a diversified configuration allows to continue module operation.

The number α_f, $0 \leq \alpha_f \leq 1$ denotes the fraction of f-fold CLB faults which can be tolerated with the set of configurations generated by the module diversification design method. For example, $\alpha_2 = 0.5$ means 50% of 2-CLB faults can be tolerated by loading a diversified configuration. The completeness condition of Eq. 4 guarantees that any single CLB fault can be tolerated. Therefore, for every generated set of configurations we have $\alpha_1 = 1$. The values of α_f for $f \geq 2$ depend on the placement details of each configuration and need to be calculated from the generated set of configurations.

C. Module Diversification for Interconnect Resources

The proposed module 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.

IV. Stress Balancing

The reliability degradation of nano-CMOS circuits due to aging and wearout depends on operational and functional factors as well as on process parameters. These include stress conditions such as temperature, time period of operation, or electrical field strengths. Balancing the CLB stress by using different configurations allows to slow down the aging process and to increase the reliability of individual CLBs. Balancing can be done by using the diversified configurations in an alternating way by reconfiguring them at runtime. In doing so, the lifetime of a reconfigurable system is prolonged.

In Section IV-A, we present a general stress balancing method using the diversified configurations to reduce the maximum accumulated stress in the CLBs of a container. Section IV-B explains the application of the general method to two different aging mechanisms.

A. Scheduling for Minimum Stress

By scheduling the diversified configurations in an alternating manner, the maximum stress accumulated in the individual CLBs of a container can be reduced. Here, the stress of a CLB may represent different stress conditions for aging mechanisms, such as switching activity or voltage potential of the transistors. CLBs which are not used in a configuration can be set into an unstressed or relaxing mode [8]. The particular mode depends on the aging mechanism.

For each configuration $\mathbf{A}$, a *stress matrix* $\mathbf{A^S}$ is constructed which contains the stress for each CLB imposed by configuration $\mathbf{A}$. Positive values in $[\mathbf{A^S}]_{i,j}$ imply that the CLB (i,j) degrades under the particular configuration. Negative values in $[\mathbf{A^S}]_{i,j}$ can be used to express recovery effects as found, for instance, in NBTI aging [29]. The stress matrix for the configuration in Eq. 1 could be:

$$\mathbf{A^S} = \begin{bmatrix} 21.6 & 4.1 & 36.0 \\ 15.9 & 11.3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{12}$$

Based on the stress matrices for the configurations, we search for the fraction of time each configuration should be active (loaded into a container and used for computation) such that the accumulated stress of the CLBs in the container over the lifetime is minimal. Given a set of w configurations and corresponding stress matrices, the *Minimum Stress Problem* is to compute the optimal active period of each diversified configuration such that the stress on *all* CLBs is minimized. This is a multi-objective optimization problem with the stress on CLBs as the objective functions. To find the Pareto-optimal solution, we focus on the most stressed CLB (i.e. we want the first CLB failure to occur as late as possible) and thus formulate the problem as a minimax problem:

$$\begin{array}{ll} \text{Minimize} & \max_{i,j} \sum_{k=1}^{w} x_k \cdot [\mathbf{A_k^S}]_{i,j} \\ \text{subject to} & 0 \leq x_k \leq 1 \\ & \sum_{k=1}^{w} x_k = 1 \end{array} \tag{13}$$

where x_k is the percentage of the active period of configuration $\mathbf{A}_k$ during the lifetime of the system.

The Minimum Stress Problem can be converted to a linear optimization problem by introducing a slack variable s:

$$\begin{array}{ll} \text{Minimize} & s \\ \text{subject to} & 0 \leq x_k \leq 1 \\ & \sum_{k=1}^{w} x_k = 1 \\ & \sum_{k=1}^{w} x_k \cdot \mathbf{A}_k^{\mathbf{S}} \leq s \cdot \mathbf{1}_{m \times n} \\ & s \geq 0, \end{array} \tag{14}$$

where $\mathbf{1}_{m \times n}$ is a m×n matrix, whose elements are all 1s. The solution to this problem specifies the fraction of time each configuration should be used. This time allocation can be enforced by a scheduler which ensures that during a time period t, configuration k is at most used for the period $t \cdot x_k$.

For the two sets of configurations shown in Eq. 8 and Eq. 9 (in the following called Config. (8) and (9)), assume that all used CLBs are equally stressed: $\mathbf{A}_k^{\mathbf{S}} = \mathbf{A}_k$. The solution of the Minimum Stress Problem for both sets are the same: $x_1 = x_2 = x_3 = 1/3$, i.e. the three configurations are equally weighted. However, the accumulated CLB stress values $\mathbf{\Sigma}_{(8)}^{\mathbf{S}}$ for Config. (8) and $\mathbf{\Sigma}_{(9)}^{\mathbf{S}}$ for Config. (9) differ:

$$\mathbf{\Sigma}_{(8)}^{\mathbf{S}} = \begin{bmatrix} 2/3 & 2/3 & 2/3 \\ 2/3 & 2/3 & 2/3 \\ 1/3 & 2/3 & 0 \end{bmatrix}, \; \mathbf{\Sigma}_{(9)}^{\mathbf{S}} = \begin{bmatrix} 2/3 & 2/3 & 2/3 \\ 2/3 & 1/3 & 2/3 \\ 2/3 & 1/3 & 1/3 \end{bmatrix} \tag{15}$$

Config. (8), which does not satisfy the max diversification condition, leaves one CLB always unused and thus imposes this stress on another CLB, which raises the failure probability of that CLB. Config. (9), which does satisfy the *max diversification condition*, distributes the stress more uniformly to all CLBs and hence reduces the failure probability of the individual CLBs.

The Minimum Stress Problem can be extended to consider multiple aging or stress mechanisms by combining the separate stress matrices for each configuration and degradation mechanism. The resulting multi-objective Minimum Stress Problem allows to find Pareto-optimal schedules of the configurations such that the maximum stress for all mechanisms is minimized at the same time.

B. Application to HCI and NBTI Degradation

This section describes the application of the stress balancing method to two different aging mechanisms: Hot Carrier Injection (HCI) and Negative Bias Temperature Instability (NBTI). Depending on the considered aging mechanism, stress and wearout have different causes. For HCI based degradation, the switching process causes high energy carriers which progressively degrade the gate oxide [29]. For NBTI degradation, stress in PMOS transistors is caused by a negative potential at the gate, i.e. in the ON-state of the transistor [30].

While we perform stress balancing for HCI and NBTI as explained below, the method can also be applied to other degradation mechanisms, such as Positive Bias Temperature Instability (PBTI), by extracting the appropriate stress data.

1) Hot Carrier Injection: Hot carrier injection (HCI) is a transistor degradation mechanism which causes a shift of the threshold voltage due to generation of interface traps and charges in the oxide by high energy (hot) carriers [29]. Hot carriers emerge when a source-drain current flows. The number of generated interface traps ΔN depends on the stress time t_s in a power-law relation: $\Delta N \propto t_s^n$. The exponent n is reported to range between 1.0 initially and 0.5 later during the lifetime [31].

In CMOS technology, a shortcut current or charge/discharge current along source-drain flows when a transistor switches and there is a conducting path from VDD to ground. Thus, the stress time t_s is proportional to the switching activity Sw of the transistor $t_s \propto Sw$. Since the shift of threshold voltage V_{th} is proportional to the number of generated interface traps ΔN, $\Delta V_{th} \propto Sw^n$.

We extract the switching activity for the transistors in the CLBs per configuration to estimate the stress duty cycle t_s per transistor. These values are averaged per CLB and inserted into the stress matrix $\mathbf{A}_i^{\mathbf{S_{HCI}}}$ for configuration i. Transistors in CLBs that are not used in configuration i have no switching activity and thus do not degrade w.r.t. HCI.

2) Negative Bias Temperature Instability: Bias temperature instability degrades the transistor by generation of trapped charges at the interface between the gate oxide and silicon substrate, attributed to the dissociation of silicon-hydrogen bonds and followed by hydrogen migration into the oxide [31]. As consequence, V_{th} increases.

For PMOS transistors the degradation is called negative bias temperature instability (NBTI) and it is more severe compared to NMOS transistors [29]. PMOS transistors are under stress when the gate voltage V_{GS} is negative and in contrast to HCI, transistors are under stress during static state operation.

The increase of V_{th} depends on the duration of stress t_s. In literature, logarithmic, exponential, or power-law time dependence have been published [29]. We follow the widespread power-law relation: $\Delta V_{th} \propto t_s^n$, with n ranging from 0.15 to 0.3.

For the computation of the stress matrix $\mathbf{A}_i^{\mathbf{S_{NBTI}}}$, we extract signal probabilities for the CLBs used in configuration i and derive the stress period for each PMOS transistor in the LUTs, i.e. the fraction of time it has negative V_{GS} (is in the ON-state).

NBTI recovery is a phenomenon that partially revokes the stress induced degradation at the oxide when V_{GS} is zero. NBTI recovery for PMOS transistors can be used to reduce the worst case degradation. CLBs that are not used in a configuration can be set into a relaxing/recovering state as proposed in [8, 32].

V. 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 [33], which prevents the place-and-route tool to use specific resources such as CLBs or Block RAMs at specified locations[1]. In the following we employ this constraint to implement diversified configurations for CLBs.

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

As shown in Figure 1, an initial configuration is generated for the module by synthesis and place-and-route of the original design file. From this configuration, the used CLBs are extracted and stored in the matrix $\mathbf{A}_1$.

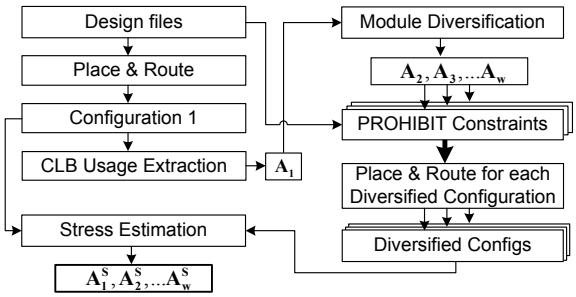

Fig. 1. Generation of diversified configurations using the module diversification design method

Using Algorithm 1, we compute diversified configuration matrices $\mathbf{A}_k$ which specify the diversified CLB usage. They are exported as PROHIBIT placement constraints and then provided to the Xilinx place-and-route tools. An example constraint file for a diversified configuration of module *alu4* implemented in a 20×4 container is as follows:

```
INST "alu4_inst" AREA_GROUP = "pblock_alu4_inst";
AREA_GROUP "pblock_alu4_inst"
  RANGE=SLICE_X48Y80:SLICE_X57Y99;
PIN "alu4_inst.i_0" LOC = SLICE_X56Y80;
PIN "alu4_inst.i_1" LOC = SLICE_X56Y80;
 :       :       :
PIN "alu4_inst.o_0" LOC = SLICE_X57Y80;
PIN "alu4_inst.o_1" LOC = SLICE_X57Y80;
 :       :       :
CONFIG PROHIBIT=SLICE_X56Y84;
CONFIG PROHIBIT=SLICE_X56Y85;
 :       :       :
CONFIG PROHIBIT=SLICE_X53Y80;
CONFIG PROHIBIT=SLICE_X54Y80;
```

The result is the set of diversified configurations for which finally the stress of transistors in CLBs is estimated to construct the stress matrices $\mathbf{A}_i^{\mathbf{S}}$.

Aging analysis of single transistors requires a CLB model at gate level or transistor level. Out of several prospects [8, 34, 35], we chose to model a CLB as LUTs composed of 2-input multiplexers in CMOS, each consisting of 14 transistors. SRAM in LUTs is less susceptible to aging [8] and not explicitly considered here.

As shown in Figure 2, the switching activity and signal probability values of a configuration are obtained from the power analysis tool Xilinx XPower Analyzer which provides both values at CLB granularity. The LUT-internal switching activity and signal probability values for individual transistors are then derived from the values at CLB granularity using the LUT model. Based on these values we compute the average over the stress of all transistors in the LUTs of a CLB. These values are then used to construct the stress matrix.

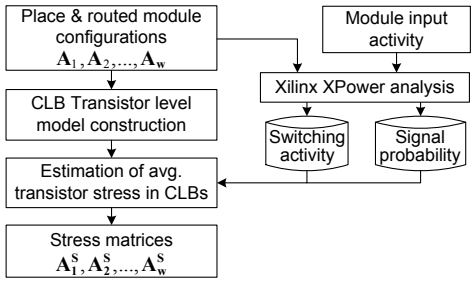

Fig. 2. Stress Estimation Flow

VI. Experimental Evaluation

We applied the proposed module diversification design method to a set of modules from MCNC benchmark suite [36] and OpenCores (http://www.opencores.org/). The target platform was a Xilinx Virtex-5 FPGA with reconfigurable containers that are 20 CLBs in height, or 80 CLBs for large modules from OpenCores, as recommended by Xilinx to align to the clock region boundary. The container width is varied from 3 up to 13 CLB columns to investigate different degrees of redundancy.

A. Module Diversification

For each module/container-size combination, we generated the minimal set of configurations using the proposed module diversification design method (tool-flow overview in Section V). Table I summarizes the container setup and reports the minimal number of configurations and the timing costs of diversified configurations for every module.

TABLE I
Configurations for different container sizes and max. frequency of original (Orig.) and diversified (Div.) modules

Module	Container Width*[CLB]		CLB Redundancy [%]		Minimal #Config.		Frequency Orig. [MHz]	Frequency Div. [MHz]	Frequency Δ[%]
	W_{min}	W_{max}	W_{min}	W_{max}	W_{min}	W_{max}			
pdc	3	5	9.1	64.0	12	3	150.8	145.2	3.7
misex3	4	7	11.1	81.8	10	3	136.5	123.3	9.7
alu4	4	7	3.9	81.8	27	3	130.4	127.5	2.3
apex4	6	9	22.4	111.8	6	2	126.2	114.5	9.3
apex2	6	11	14.3	117.8	8	2	122.4	115.3	6.2
des_perf	7	13	4.9	117.1	22	2	135.3	127.3	6.3
aes_core	3	5	27.7	127.3	5	3	124.7	124.7	0.04

* Container height for large OpenCore modules *des_perf* and *aes_core* is 80 CLBs. Container height for other modules is 20 CLBs.

Column 1 lists the implemented modules and column 2 shows the minimal (W_{min}) and maximal (W_{max}) used container width. The 3rd column shows the degree of CLB redundancy for different container sizes. For example, *apex4* uses 98 CLBs in the 20×6 container (i.e. W_{min}), which corresponds to $(20 \times 6 - 98)/98 \approx 22.4\%$ redundancy. Column 4 lists the minimum number of configurations tolerating all single CLB faults in W_{min} and W_{max}. For larger containers with higher redundancy, fewer configurations are required.

Since the module diversification design method applies additional constraints to prohibit certain CLB placements, the

maximally achievable frequency of a module may be affected. The last column in Table I reports the maximal frequency (over all container widths) of the original unconstrained module (Orig.) in comparison to the diversified modules (Div.). For the diversified modules, the reported maximal frequency always corresponds to the slowest configuration of the module. The timing cost is under 9.7%, which is a promising result for an approach that obtains aging mitigation and fault tolerance at no additional area overhead. Note that the original module implementation is one of the diversified configurations and thus can be used when full performance is required. If the system frequency is lower than the maximal frequency of the diversified modules, there are no timing costs at all.

B. Reliability Analysis

In this section, we investigate the reliability improvement of the proposed module diversification design method for different degrees of CLB redundancy, CLB reliabilities, and number of configurations.

Figure 3 shows the module reliability according to Eq. 11 of Section III-B of module *apex4* for a CLB reliability $R_{CLB}(t) = 0.999$, i.e. the probability of any single CLB to be operational throughout a given time period t is 0.999. We assume $C_f = 1.0$. The figure displays the reliability increase for different numbers of diversified configurations (from the minimum number of configurations up to 20) and for container sizes from 20×6 to 20×9 CLBs, which corresponds to CLB redundancies from 22.4% to 111.8%. Without any diversified configurations, the module reliability is very low at approximately 0.91. Using diversified configurations, the module reliability increases drastically due to higher single and multi-fold CLB-fault tolerance from extra configurations. For example, three configurations are sufficient to tolerate all single-CLB faults for *apex4* implemented in a 20×8 container. In addition, 47% of double and 22% of triple CLB-faults can be tolerated with these three configurations as well. An extra set of 17 diversified configurations increases 2-CLB and 3-CLB fault tolerability further to 88% and 57%, respectively.

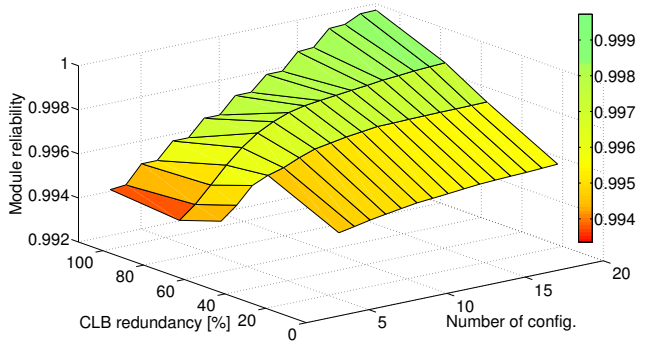

Fig. 3. Module reliability of *apex4* for different ratios of CLB redundancy and number of configurations with CLB reliability 0.999

Larger container sizes imply higher redundancy which reduces the probability $R_{CLB}(t)^{m \cdot n}$ that *all* CLBs in the container are fault free (cf. Eq. 11). This may reduce the overall module reliability as seen on the left in Figure 3. With increasing number of configurations, the tolerance of f-fold CLB-faults rises and very high module reliability can be achieved.

Figure 4 shows the reliability of modules using module diversification versus a single configuration without fault tolerance measure for different CLB reliabilities, computed according to Eq. 11. For all modules, the minimal set of diversified configurations implemented in the smallest containers is evaluated. It is clear that with diversified configurations the module reliability is substantially higher than without any fault tolerance measures. The module reliability of *des_perf* and *aes_core* ranges from 0.59 to 0.95, respectively 0.83 to 0.98, when module diversification is not applied. With module diversification, the reliability increases from 0.895 to 0.999 and from 0.980 to 0.9998, respectively.

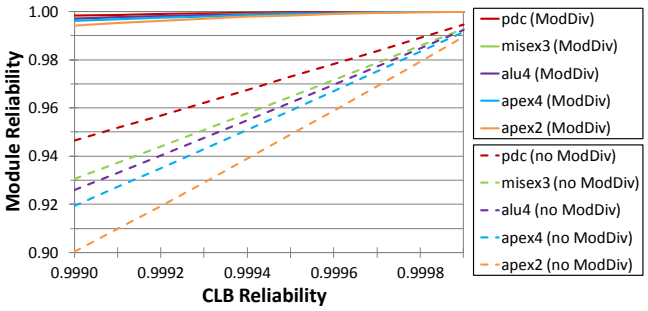

Fig. 4. Module reliability *with* and *without* module diversification for different CLB reliabilities. (Reliabilities of des_perf and aes_core are not shown in the figure for clarity, but discussed in the text).

The reliability improvement factor (RIF) is a metric to estimate the effectiveness of fault tolerance schemes [37]. The RIF is the ratio of the failure probability of the original system and the failure probability of the fault tolerant system, i.e. the system using diversified module configurations:

$$\text{RIF} = \frac{1 - R_{\text{No FT}}}{1 - R_{Div}} \tag{16}$$

Figure 5 plots the RIF for the five investigated modules and CLB reliabilities ranging from 0.9990 to 0.9999. With the proposed module diversification design method, reliability improvement factors of up to 330 are achieved.

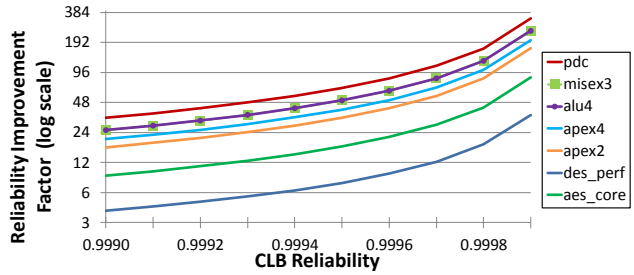

Fig. 5. Reliability improvement factor for the modules when module diversification is applied

C. Stress Balancing

For the generated diversified configurations, we compute an optimal schedule according to the Minimum Stress Problem introduced in Section IV. We consider the stress for the two aging mechanisms Hot Carrier Injection (HCI) and Negative Bias Temperature Instability (NBTI). Details on the computation of stress matrices for HCI and NBTI are given in Sections IV-B and V. The following results compare the stress for the initial configuration and the minimum number of configurations required for single CLB-fault tolerance (cf. Eq. 6).

Figure 6(a) shows the switching activity (HCI stress) in the CLBs used in the initial configuration of module *alu4* with red values for higher and blue values for lower activity. Figure 6(b) shows the switching activity for another configuration which is max diversified w.r.t. (a). Although configuration (b) uses the CLBs unused in (a), there is still significant overlap between them. If both configurations are used in a simple alternating schedule (as proposed by [8, 27]), stress is not well balanced and the maximum stress is hardly reduced at all, as shown in Figure 6(c).

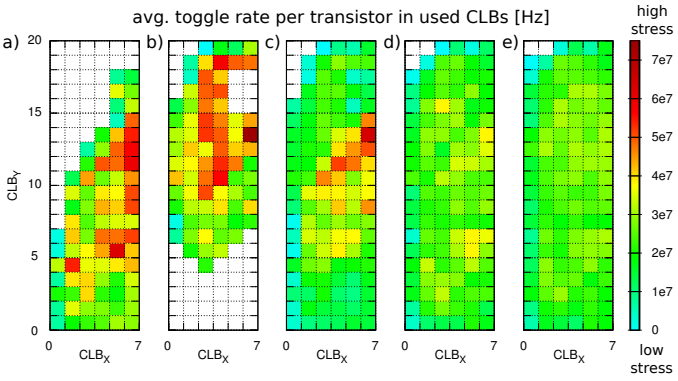

Fig. 6. Switching activities of a), b) two max. diversified configurations, c) an alternating schedule thereof, d) a balanced scheduled with min. number of four configurations and e) eight configurations (*alu4*)

Figure 6(d) shows the stress imposed on CLBs using the proposed module diversification with the minimal number of configurations and optimal scheduling for stress balancing. The stress maxima are clearly reduced. For eight configurations in Figure 6(e), the maxima can be even further reduced[2].

Table II summarizes the HCI and NBTI stress reduction of the optimal scheduling w.r.t. the initial configuration $\mathbf{A}_1$. For each module, the reduction of the maximum stress (worst case CLB) and the average over all used CLBs are listed. Since the stress reduction depends on the available CLB redundancy, we investigate the cases with the minimal container width W_{min} and maximal container width W_{max}.

The results clearly show that both maximum and average stress are reduced for all modules. The comparison of minimum and maximum container width W_{min}, W_{max} shows that the stress reduction increases with the amount of available CLB redundancy in the container.

[2]The white CLBs in the upper left are unused due to timing optimization of the vendor place-and-route tool.

TABLE II
STRESS REDUCTION [%] BY PROPOSED BALANCED SCHEDULE WITH MIN. NUMBER OF CONFIGURATIONS (EQ. 6)

Module	HCI stress reduction [%] Max. W_{min}	Max. W_{max}	Avg. W_{min}	Avg. W_{max}	NBTI stress reduction [%] Max. W_{min}	Max. W_{max}	Avg. W_{min}	Avg. W_{max}
pdc	33.8	43.6	8.9	36.5	12.0	34.2	9.7	35.8
misex3	37.7	41.2	10.6	41.6	13.9	37.4	10.3	43.7
alu4	32.6	51.7	3.1	43.4	8.6	35.5	5.7	44.5
apex4	35.2	54.6	15.4	50.5	22.3	49.8	20.2	50.6
apex2	24.8	50.3	13.2	49.0	17.0	49.3	14.4	49.1
des_perf	68.9	50.2	13.3	51.1	14.0	50.0	7.0	49.4
aes_core	26.7	49.7	20.7	48.6	20.8	50.0	20.2	48.4

The average stress reduction is bound by the CLB redundancy. The more CLBs are available to distribute the load to, the higher the reduction. The maximum stress reduction depends on the scheduling of the configurations.

The maximum (average) HCI stress reduction ranges up to 68.9% (51.1%) using the minimum number of configurations for single CLB-fault tolerance combined with the stress balancing schedule. For maximum (average) NBTI stress, the reduction reaches up to 50.0% (50.6%). For the module *des_perf*, the large HCI stress reduction is achieved even for small containers, i.e. low CLB redundancy. In this circuit, there is one CLB with significantly higher HCI stress than all other CLBs and by balanced scheduling this high stress is well distributed to other CLBs.

Figure 7 shows the relation between ΔV_{th} degradation due to HCI and operation time. According to Section IV-B, we assume that stress is uniformly distributed over the lifetime and the exponent n is 0.5. When ΔV_{th} reaches a critical value $\Delta V_{th}^{\text{crit}}$, the device fails. The period of time is marked as mean time to failure $\text{MTTF}_{\text{no\_ModDiv}}$ in the graph. If module diversification and stress balancing is applied, the mean time to failure increases. Reducing the stress by 68.9% increases time to failure by 222%. This point is labeled by $\text{MTTF}_{\text{ModDiv}}$ in the graph.

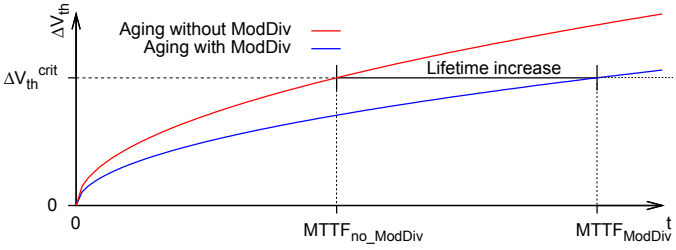

Fig. 7. Relation between ΔV_{th} and operation time for HCI degradation

The consideration of NBTI time to failure is similar to HCI since it can also be modeled by a power-law relation. Cutting the stress in half by module diversification combined with optimal stress balancing doubles the time to failure.

If more configurations than the minimum number required for single CLB-fault tolerance are used, the stress can be reduced even more.

VII. Conclusions

The reliability of nano-CMOS circuits is threatened by different aging mechanisms. We proposed a novel *module diversification* design method for runtime reconfigurable FPGA-based systems to ensure single and multiple CLB-fault tolerance as well as aging mitigation. We developed an algorithm which generates the minimal number of diversified configurations required to tolerate at least any single CLB-fault in a reconfigurable container and which can generate additional configurations for multi-fold CLB-fault tolerance. We computed an optimal schedule of diversified configurations for aging mitigation by stress balancing.

Experimental results show the large improvement of system reliability due to fault tolerance and the stress reduction on CLBs due to optimal scheduling. Reliability improvement factors between 4 and 330 are achieved. Module diversification and optimal stress balancing allow to reduce the maximal HCI (NBTI) stress by up to 68.9% (50.0%) resulting in an increase in expected lifetime by up to 222% (100%).

Acknowledgments

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

References

[1] Convey Computer Corp., "Homepage of Convey Computer", http://www.conveycomputer.com/, accessed at Feb. 11, 2013.

[2] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems", in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000, pp. 205–214.

[3] T. T.-O. Kwok and Y.-K. Kwok, "On the design of a self-reconfigurable SoPC cryptographic engine", in *24th International Conference on Distributed Computing Systems - Workshops*, 2004, pp. 876–881.

[4] W. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2005.

[5] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An Overview of Reconfigurable Hardware in Embedded Systems", *EURASIP Journal on Embedded Systems*, pp. 1–19, 2006.

[6] J. McPherson, "Reliability Challenges for 45nm and Beyond", in *ACM/IEEE Design Automation Conference*, 2006, pp. 176–181.

[7] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung, "Degradation in FPGAs: Measurement and Modelling", in *ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2010, pp. 229–238.

[8] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. Irwin, and K. Sarpatwari, "Toward increasing FPGA lifetime", *IEEE Trans. on Dep. and Secure Computing*, vol. 5, no. 2, pp. 115–127, 2008.

[9] C. Stroud, E. Lee, and M. Abramovici, "BIST-based diagnostics of FPGA logic blocks", in *IEEE Int'l Test Conference (ITC)*, 1997, pp. 539–547.

[10] M. S. Abdelfattah, L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Transparent structural online test for reconfigurable systems", in *IEEE International On-Line Testing Symposium (IOLTS)*, June 2012, pp. 37–42.

[11] M. Dales, "Managing a reconfigurable processor in a general purpose workstation environment", in *Design, Automation and Test in Europe (DATE)*, 2003, pp. 980–985.

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

[13] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "Test pattern and test configuration generation methodology for the logic of RAM-based FPGA", in *Asian Test Symposium*, 1997, pp. 254–259.

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

[15] W. K. Huang, F. J. Meyer, X.-T. Chen, and F. Lombardi, "Testing configurable LUT-based FPGA's", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 276–283, 1998.

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

[17] 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.

[18] M. Abramovici, C. Strond, C. Hamilton, S. Wijesuriya, and V. Verma, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications", in *IEEE International Test Conference (ITC)*, 1999, pp. 973–982.

[19] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 2, pp. 216–226, 2007.

[20] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich, "Test strategies for reliable runtime reconfigurable architectures", *IEEE Transactions on Computers*, 2013, doi: 10.1109/TC.2013.53, ISSN: 0018-9340.

[21] 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.

[22] M. Abramovici, C. Stroud, and J. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 12, pp. 1284–1294, 2004.

[23] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 212–221, 1998.

[24] A. Kanamaru, H. Kawai, Y. Yamaguchi, and M. Yasunaga, "Tile-based fault tolerant approach using partial reconfiguration", in *International Workshop on Reconfigurable Computing (ARC)*, 2009, pp. 293–299.

[25] W.-J. Huang and E. J. McCluskey, "Column-based precompiled configuration techniques for FPGA", in *IEEE Symp. Field-Prog. Custom Computing Machines (FCCM)*, 2001, pp. 137–146.

[26] S. Mitra, W.-J. Huang, N. Saxena, S.-Y. Yu, and E. McCluskey, "Reconfigurable architecture for autonomous self-repair", *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 228–240, 2004.

[27] E. Stott and P. Cheung, "Improving FPGA Reliability with Wear-Levelling", in *International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 323–328.

[28] M. Psarakis and A. Apostolakis, "Fault tolerant FPGA processor based on runtime reconfigurable modules", in *17th IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[29] J. B. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor, "Electronic circuit reliability modeling", *Microelectronics Reliability*, vol. 46, no. 12, pp. 1957–1979, 2006.

[30] D. K. Schroder, "Negative bias temperature instability: What do we understand?" *Microelectronics Reliability*, vol. 47, no. 6, pp. 841–852, 2007.

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

[32] F. Firouzi, S. Kiamehr, and M. B. Tahoori, "A linear programming approach for minimum NBTI vector selection", in *Great Lakes Symposium on VLSI (GLVSI)*, 2011, pp. 253–258.

[33] Xilinx, *Constraints Guide (UG625, v. 13.4)*, 2012.

[34] S. Kiamehr, A. Amouri, and M. B. Tahoori, "Investigation of NBTI and PBTI Induced Aging in Different LUT Implementations", in *Int'l Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[35] 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.

[36] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0", 1991.

[37] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2000.