STRAP: Stress-Aware Placement for Aging Mitigation in Runtime Reconfigurable Architectures

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

Proceedings of the 34th IEEE/ACM International Conference on Computer-Aided Design (ICCAD'15) Austin, Texas, USA, 2-6 November 2015

url: http://dl.acm.org/citation.cfm?id=2840825

Abstract: Aging effects in nano-scale CMOS circuits impair the reliability and Mean Time to Failure (MTTF) of embedded systems. Especially for FPGAs that are manufactured in the latest technology node, aging is a major concern. We introduce the first cross-layer aging-aware placement method for accelerators in FPGA-based runtime reconfigurable architectures. It optimizes stress distribution by accelerator placement at runtime, i.e. to which reconfigurable region an accelerator shall be reconfigured. Additionally, it optimizes logic placement at synthesis time to diversify the resource usage of individual accelerators, i.e. which CLBs of a reconfigurable region shall be used by an accelerator. Both layers together balance the intra- and interregion stress induced by the application workload at negligible performance cost. Experimental results show significant reduction of maximum stress of up to 64% and 35%, which leads to up to 177% and 14% MTTF improvement relative to state-of- the-art methods w.r.t. HCI and BTI aging, respectively.

[1] **ACM COPYRIGHT NOTICE**
©2015 ACM. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STRAP: Stress-Aware Placement for Aging Mitigation in Runtime Reconfigurable Architectures

Hongyan Zhang*, Michael A. Kochte‡, Eric Schneider‡, Lars Bauer*, Hans-Joachim Wunderlich‡ and Jörg Henkel*
*Chair for Embedded Systems, Karlsruhe Institute of Technology, Karlsruhe, Germany
{hongyan.zhang, lars.bauer, henkel}@kit.edu
‡Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany
{michael.kochte, eric.schneider, wu}@informatik.uni-stuttgart.de

***Abstract*—Aging effects in nano-scale CMOS circuits impair the reliability and Mean Time to Failure (MTTF) of embedded systems. Especially for FPGAs that are manufactured in the latest technology node, aging is a major concern.**

We introduce the first cross-layer aging-aware placement method for accelerators in FPGA-based runtime reconfigurable architectures. It optimizes stress distribution by accelerator placement at runtime, i.e. to which reconfigurable region an accelerator shall be reconfigured. Additionally, it optimizes logic placement at synthesis time to diversify the resource usage of individual accelerators, i.e. which CLBs of a reconfigurable region shall be used by an accelerator. Both layers together balance the intra- and inter-region stress induced by the application workload at negligible performance cost. Experimental results show significant reduction of maximum stress of up to 64% and 35%, which leads to up to 177% and 14% MTTF improvement relative to state-of-the-art methods w.r.t. HCI and BTI aging, respectively.

I. Introduction and Related Work

Runtime reconfigurable architectures enable dynamic adaptation to changing workloads, which allows to optimize area, performance and power [1]. They consist of a general purpose processor core and a reconfigurable fabric that is implemented as an SRAM-based field programmable gate array (FPGA) to allow fast runtime reconfiguration. The reconfigurable fabric of the system is divided into multiple rectangular *regions* into which *hardware accelerators* can be reconfigured during runtime. Fig. 1 shows an example of a reconfigurable fabric with 8 regions that are composed of an array of *configurable logic blocks* (CLBs) with interconnects. Each CLB consists of several *look-up-tables* (LUTs) that can be programmed with a *configuration* to implement a desired logic function. Complex applications typically utilize multiple reconfigurable regions and multiple different accelerators for different computations. For instance, video encoding can be expedited by using accelerators for motion estimation, transformations, filters, entropy coding etc. Whenever switching from one computational kernel to another (e.g. from motion estimation to transformations), some or all regions may be reconfigured with different accelerators for the next kernel.

FPGAs are implemented in the latest technology node (e.g. 20nm/16nm for Xilinx' recent/announced UltraScale family) and FPGA-based reconfigurable regions suffer from degradation due to aging [2, 3]. This paper focuses on FPGA aging, as in reconfigurable architectures most of the application's computations are offloaded to the reconfigurable regions. The manifestations of aging can range from increased transistor switching delay up to severe permanent defects that cause a transistor or interconnect wire to fail entirely. Different types of 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) [4]. 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 [2, 5]. Due to the increasing susceptibility of current CMOS technology nodes, these effects cannot be ignored anymore [6] and their consideration has become essential for dependable designs.

Aging mitigation by wear-leveling in FPGA-based runtime reconfigurable architectures can be achieved by using alternative logic mappings in CLBs, using spare resources in the fabric, and changing placements of accelerators [7]. In [8], 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.

Since typically not all CLBs in a region are actively used by an accelerator configuration [5], 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 [7]. This reduces the maximum stress in the resources and increases the system's Mean Time to Failure (MTTF), as demonstrated in [5, 9, 10]. However, [5, 9] target systems without runtime reconfiguration. They create alternative configurations for the entire FPGA, i.e. placing *one* accelerator anywhere on the FPGA. Zhang *et al.* [10] consider runtime reconfiguration with multiple regions and accelerators. For each accelerator they create multiple alternative configurations *independently*, i.e. without considering other accelerators. As their optimization is performed at synthesis time, they cannot consider dynamically changing workload at runtime. With regard to place-and-route during synthesis, the method of [10] is the current state-of-the-art approach and we will compare with it in the evaluation.

The online placement of accelerators proposed in [11] extends the KAMER placement algorithm [12] by considering the maximum stress in the regions. The accumulated stress of the resources in the candidate region are stored in a degradation table. The algorithm performs a *local optimization* that considers the accelerators one after the other. With regard to runtime accelerator placement, the method of [11] is the current state-of-the-art approach and we will compare with it in the evaluation to show the advantages of our cross-layer placement.

This paper proposes STRAP, a novel cross-layer placement method to reduce the maximum stress by aging mitigation. For the first time, it combines complex offline optimizations at the synthesis layer with situation-dependent adaptation at the runtime layer to optimize the intra- and inter-region stress distribution simultaneously. At the runtime layer, STRAP proposes an algorithm that places accelerators

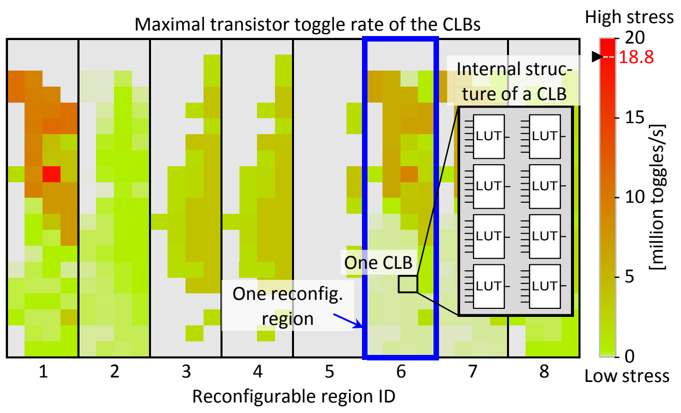


Fig. 1: Transistor stress distribution in a reconfigurable fabric with 8 regions; each region consists of 4×20 CLBs with 8 LUTs each (same setup as for evaluation); the color of a CLB corresponds to the highest toggle rate of any of its transistors; the symbol ▶ on the right scale denotes the maximum stress over all regions

to different reconfigurable regions (i.e. it decides to which region they shall be reconfigured) while considering the induced intra- and inter-region stress distribution simultaneously. At the synthesis layer, STRAP proposes an algorithm that diversifies stress during place-and-route by preventing overlapping of high stress CLBs from different accelerators, which further improves the intra-region stress distribution at runtime. For prototyping purposes, we have integrated STRAP into the Xilinx tool-chain and the runtime system of a reconfigurable processor.

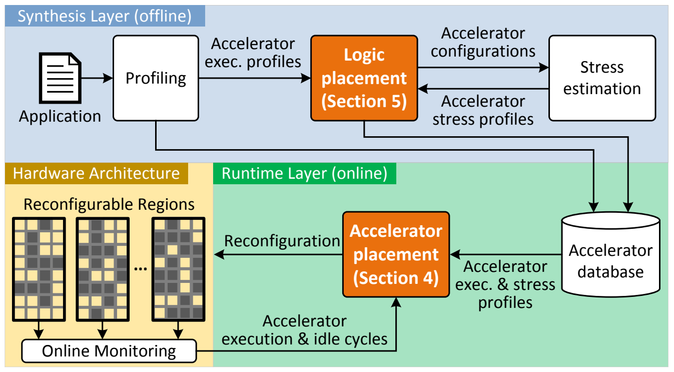
Paper structure: Section II presents the system overview. Section III provides background information on stress modeling and representation. Sections IV and V explain the details of our accelerator placement and logic placement, respectively. Results are discussed in Section VI.

II. Overview of our STRAP method

The MTTF of a system is constrained by the component with the highest stress [5]. In order to prolong the MTTF of a reconfigurable fabric, we need to reduce its peak stress and avoid stress accumulation. Fig. 1 shows a typical reconfigurable fabric with 8 reconfigurable regions and 4×20 CLBs per region. The figure visualizes the distribution of HCI stress after running an H.264 video encoder. Higher HCI stress corresponds to more toggles per second of a transistor (stress and aging background is explained in Section III). For each CLB, the highest toggle rate of any transistor is identified and plotted in a color-scale from 0 (*low stress*, bright gray) to 20 million toggles per second (*high stress*, dark red). It is noticeable that several CLBs are not used, e.g. most parts of region 5 and some parts of regions 3 and 4, whereas some CLBs in region 1 contain transistors that are highly stressed. The latter represent *stress hotspots* where high stress accumulates in some of the components of the fabric, hence reducing the MTTF of the system.

Our goal is to place accelerators such that the maximal stress is minimized. Our method considers stress at the granularity of CLBs, whereas our evaluation in Section VI considers stress at transistor granularity. If the stress from a stress hotspot can be distribute to less stressed CLBs like in regions 3–5 in Fig. 2, then the maximum stress in the reconfigurable regions is reduced, leading to increased MTTF [5].

Fig. 2 provides an overview of our cross-layer placement method STRAP, showing the synthesis layer, the runtime layer, and how they interact with the hardware architecture of a



Synthesis Layer (offline)
Application
Profiling
Accelerator exec. profiles
Logic placement (Section 5)
Accelerator configurations
Accelerator stress profiles
Stress estimation
Hardware Architecture
Runtime Layer (online)
Reconfigurable Regions
Reconfiguration
Accelerator placement (Section 4)
Accelerator exec. & stress profiles
Accelerator database
Accelerator execution & idle cycles
Online Monitoring

Fig. 2: Overview of our cross-layer stress-aware placement method

reconfigurable system. For logic placement at synthesis time, the challenge is to place-and-route accelerators in a way that supports stress balancing at runtime, but without having runtime information. STRAP first performs an offline application profiling of each application kernel to obtain estimates on (i) how often accelerators will be executed relative to each other and (ii) how long each accelerator executes to finish its task. This information is used to steer runtime accelerator placement (Section IV) and synthesis time logic placement (Section V).

Based on the accelerator configuration after place-and-route, the stress estimation process in Fig. 2 analyzes the signal activities in all CLBs used by the accelerator to obtain the information how much stress it induces to a reconfigurable region. Accelerator execution and stress profiles are stored together with the accelerator bitstreams in main memory for runtime decision making.

At runtime, STRAP decides into which reconfigurable region an accelerator shall be reconfigured, whenever the application demands different accelerators. It performs online monitoring of each region to track when the region was reconfigured last and how often the currently-reconfigured accelerator was executed. Whenever a region is reconfigured, the execution counter and reconfiguration timestamp is read and reset. Together with the accelerator stress profile created at synthesis time, STRAP then calculates the exact *stress state* for all CLBs of the region. This information is used to decide the runtime accelerator placement (details in Section IV). Note that with the feature of partial reconfiguration provided by FPGA vendors, reconfigurable regions are spatially isolated from each other, i.e. the reconfiguration of one region does not affect the resource usage (and thus stress) of any other region. The size of the regions are chosen such that accelerators can be fitted into any region.

III. Stress and Aging Background

Before describing the details of our cross-layer stress-aware placement method in Sections IV and V, we clarify basics about aging, explain the assumptions we make for our method, and how we represent stress.

A. Basic Stress Properties

Aside from material constants, aging mainly depends on three non-material factors: supply voltage, temperature, and transistor activities. For the algorithms in our 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 (they are often optimized for

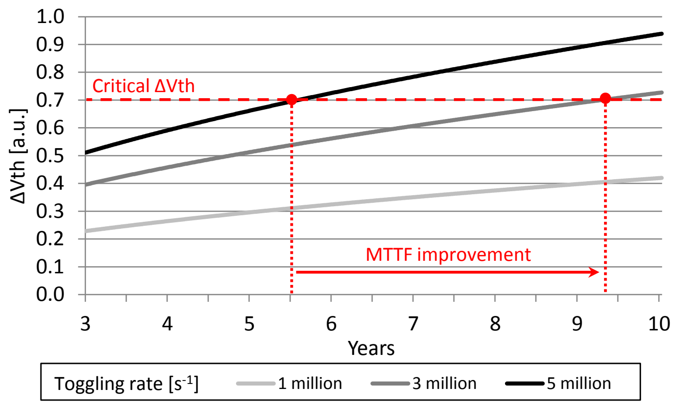


Fig. 3: Threshold voltage increases over time due to HCI under different toggling rates

data-level parallelism and thus run at rather low frequency [13] with correspondingly low power density [14, 15]). However, for the evaluation we use a more accurate model that also considers the influence of temperature more detailed as described in the experimental setup in Section VI-A.

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. Different aging mechanisms exist that lead to threshold voltage shift. For instance, Fig. 3 shows the threshold voltage increase due to HCI that depends on the transistor toggling rate. We define the *Mean Time to Failure (MTTF)* of a transistor as the time until its threshold voltage exceeds a certain critical value. Reducing the threshold voltage shift by reducing the stress increases the MTTF.

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

There are different models for these aging effects, e.g. [2, 4, 5, 16, 17], and 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 [16, 17]. 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 [17, 18]. 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 [3] 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 decision making (cf. Sections IV and V) rather than evaluating complex aging models at runtime. However, even though *stress* is additive, the resulting *degradation* (ΔV_{th}) is not necessarily so. While our proposed STRAP method exploits the monotonic and additive property during decision making, we use state-of-the-art aging models for evaluation and their details are given in Section VI-A and Appendix A. As STRAP applies to both types of stress, we will refer to "*stress*" when we do not need to explicitly differentiate between static and dynamic stress. Note that STRAP optimizes either for dynamic or for static stress.

B. 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 cross-layer placement method. We define *CLB stress* as the sum of the stress experienced by all its transistors. With this definition, CLB stress preserves the 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.

Runtime reconfigurable architectures can execute different applications. Each of them can use different accelerators that use different CLBs to implement their timing-critical path. As any reconfigurable region can be reconfigured to implement any of these accelerators, it is not possible to identify upfront which CLBs are more important than others w.r.t. protection against aging. Therefore, our method treats all CLBs in the reconfigurable fabric equally important.

With the established stress properties, we are able to describe the stress in the reconfigurable fabric in a formal way. The stress state of a reconfigurable region (as it is visualized in Fig. 1) 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 induces per clock cycle is obtained from offline stress estimation and called *unit stress*, denoted by matrix $\mathbf{U}$ of the same size as $\mathbf{S}$. In general, the stress increase due to the work done by an accelerator is shown in Eq. (1), where scalars τ_{exec} and τ_{idle} denote the number of clock cycles when the accelerator is in execution or idle, while matrices $\mathbf{U_{exec}}$ and $\mathbf{U_{idle}}$ denote the unit stress induced by the accelerator during execution or idle time:

$$\mathbf{s} := \tau_{exec}\mathbf{U_{exec}} + \tau_{idle}\mathbf{U_{idle}}. \quad (1)$$

During idle, we assume all inputs to the accelerator are hold at constant values, e.g. all zeros. In this case, the accelerator exhibits a different stress pattern from when it is being executed.

To obtain the unit stress of an accelerator, the placed-and-routed configuration and its input signal activities (toggle rate and average duty cycle) are fed to a power analyzer[1] that

[1]we used Xilinx XPower

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 LUT in the CLBs used by the accelerator. We employ a transistor-level LUT model [2, 19] to calculate the toggle rate and stress duty cycle of individual transistors by propagating the LUT input signal activity through the LUT model.

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. They are used by the stress-diversifying logic placement and the runtime system. 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 (cf. Section II) 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.

IV. Runtime Accelerator Placement

The reconfigurable fabric consists of N equally sized rectangular regions. During runtime, the application requests to configure M ($M \le N$) accelerators to speed up its computational kernels. The runtime system has to decide to which regions the M accelerators shall be configured. For $M < N$ application-requested accelerators, the runtime system first decides which $N - M$ regions shall *not* be reconfigured, e.g. by using a least recently used replacement policy. The decision to which of the remaining regions an accelerator is placed does not affect the application performance.

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 $1 \le k \le N$, $1 \le x \le X$, $1 \le y \le Y$). Similarly, the stress that will be induced by an accelerator j ($1 \le j \le M$) is denoted as $[\mathbf{s_j}]_{xy}$ (cf. Eq. (1)). It depends on how often the accelerator will be executed, as determined by offline profiling (cf. Section II). If an accelerator j is placed into region k, then the accelerator executions increase the stress state of the region to $\mathbf{S'_k} = \mathbf{S_k} + \mathbf{s_j}$.

The task 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

$$\frac{1}{NXY}\left(\sum_{k}^{N}\mathbf{S_k} + \sum_{j}^{M}\mathbf{s_j}\right), \qquad (2)$$

which 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. To achieve this at runtime, we propose a heuristic that follows these two rules: 1) maximal utilization of under-stressed CLBs within one region, i.e. the stress shall be evenly distributed among different CLBs within the region (*intra-region* distribution), and 2) avoid placing high-stress accelerators into highly stressed regions, i.e. the stress shall be evenly distributed among different regions (*inter-region* distribution). We define the profit function of placing accelerator j into region k as

$$\text{Profit}_{jk} = \text{Profit}_{jk}^{intra} + \text{Profit}_{jk}^{inter}, \qquad (3)$$

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

$$\text{Profit}_{jk}^{intra} = \sum_{x,y}\left|[\mathbf{S_k}]_{xy} - \lambda_k\right| - \sum_{x,y}\left|[\mathbf{S_k} + \mathbf{s_j}]_{xy} - \lambda'_{k,j}\right| \qquad (4)$$

$$\text{with } \lambda_k = \frac{1}{XY}\sum_{x,y}[\mathbf{S_k}]_{xy} \text{ and } \lambda'_{k,j} = \frac{1}{XY}\sum_{x,y}[\mathbf{S_k} + \mathbf{s_j}]_{xy}$$

$$\text{Profit}_{jk}^{inter} = \left|\sum_{x,y}[\mathbf{S_k}]_{xy} - \Lambda\right| - \left|\sum_{x,y}[\mathbf{S_k} + \mathbf{s_j}]_{xy} - \Lambda'\right| \qquad (5)$$

$$\text{with } \Lambda = \frac{1}{N}\sum_{k,x,y}[\mathbf{S_k}]_{xy} \text{ and } \Lambda' = \frac{1}{N}\left(\sum_{k,x,y}[\mathbf{S_k}]_{xy} + \sum_{j,x,y}[\mathbf{s_j}]_{xy}\right).$$

The two summation operations in the intra-region profit function (Eq. (4)) express the sum of the CLB stress deviation from the average stress value before and after placing accelerator 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 j into it. In a similar manner, the inter-region profit function in Eq. (5) 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.

Our stress-aware runtime accelerator placement (Alg. 1) iterates through all required accelerators (Lines 2 to 17). In each iteration, it calculates the profits of placing the accelerator into all available regions (Lines 5 to 14) and places the accelerator into the region that provides the highest profit (Line 15). The complexity of this algorithm is $\mathcal{O}(M^2XY)$. If the application decides to keep an accelerator configuration for a longer time (i.e. to not reconfigure it), then the stress may not be distributed evenly to all regions. The region where this accelerator resides would be constantly stressed by one accelerator without stress redistribution. This happens if an accelerator delivers high speedup and is frequently required by the application. 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

Algorithm 1 Stress-aware 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. for j := 1 to len(Acc) do
 3.   max_profit := $-\infty$
 4.   selected_reg := null
 5.   for k := 1 to len(Reg) do
 6.     if occupied[k] == 1 then
 7.       continue
 8.     end if
 9.     profit := CalcProfit(Acc[j], Reg[k]) // Eq. (3)
10.     if profit > max_profit then
11.       max_profit := profit
12.       selected_reg := k
13.     end if
14.   end for
15.   Place accelerator j into region selected_reg
16.   occupied[selected_reg] := 1
17. end for
```

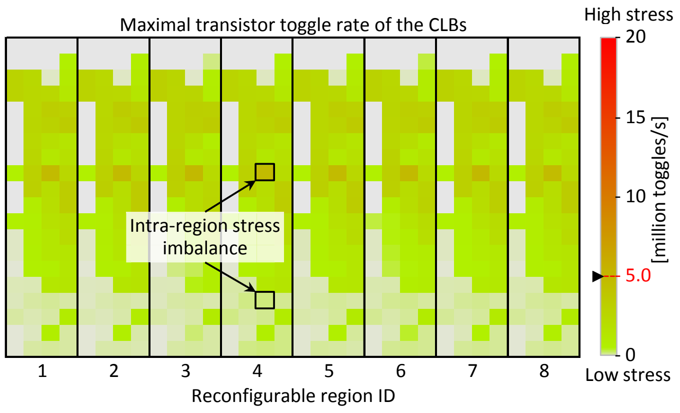


Fig. 4: Transistor stress distribution using stress-aware runtime accelerator placement

reconfiguration overhead, while also not too long to avoid stress accumulation. For instance, a time period of 100 million cycles (1 s at 100 MHz) is short enough to avoid aging accumulation and the induced application performance degradation is only 0.21%.

V. Synthesis time Logic Placement

Fig. 4 shows that dynamic stress is uniformly distributed over all reconfigurable regions when employing our runtime accelerator placement from Section IV, compared to the stress-unaware placement in Fig. 1. The maximal transistor toggle rate is reduced by more than 73% from 18.8 to 5.0 million toggles/s. However, when high stress CLBs of different accelerators *overlap* at the same relative (x, y) location, the runtime accelerator placement cannot achieve intra-region stress distribution, as noticeable in the upper-middle part of all reconfigurable regions in Fig. 4. STRAP addresses this problem by applying placement constraints at *synthesis time* to diversify the CLB usage among different accelerators, which reduces the overlapping of high stress CLBs. To minimize the timing impact on accelerators, the mapping of logic functions to CLBs is left to the vendor place-and-route algorithm. Instead we only constrain *which* CLBs shall be used to place-and-route an accelerator without additional constraints on logic mapping or routing.

Our logic placement algorithm (Alg. 2) diversifies the high stress CLBs of different accelerators to different (x, y) CLB locations in the reconfigurable regions. First, unconstrained configurations of all accelerators are generated (Lines 1 to 5). For each accelerator configuration the CLB stress is estimated (see Section III-B), and the maximal achievable frequency is extracted from the place-and-route log files (Lines 3 and 4). The generated initial configurations are then sorted in ascending order of their maximal achievable frequencies (Line 6). The reconfigurable fabric typically runs at the frequency of the slowest accelerator f_{min}. In order to minimize the impact on system performance, it is placed and routed without stress-diversifying placement constraints. Its CLB stress distribution is taken as the initial reference distribution (Line 7). As long as the proposed logic placement does not reduce the frequency of an accelerator below f_{min}, there is no performance impact/penalty for the whole system. During the generation of other accelerator configurations, $\mathbf{R}$ keeps track of the sum of the stress distribution of all $j-1$ previously generated accelerators, i.e. $\mathbf{R} = \sum_{i=1}^{j-1} \mathbf{s_i}$.

The remaining accelerators will be placed-and-routed again in ascending order of their maximal frequencies (Lines 8 to 23).

Algorithm 2 Stress-diversifying logic placement

```
Input: List of accelerators Acc.
 1. for j := 1 to len(Acc) do
 2.   Place-and-route Acc[j] without any placement constraints
 3.   $s_j$ := get_stress(Acc[j])
 4.   Acc[j].max_freq := get_max_freq(Acc[j])
 5. end for
 6. Acc := sort_ascending(Acc, key=max_freq)
 7. $\mathbf{R}$ := $s_1$
 8. for j := 2 to len(Acc) do
 9.   prohibit_xy := $\emptyset$
10.   for x := 1 to Acc[j].n_cols do
11.     for y := 1 to Acc[j].n_rows do
12.       if Condition Eq. (6) is satisfied for (x,y) then
13.         prohibit_xy.add((x,y))
14.       end if
15.     end for
16.   end for
17.   Place-and-route Acc[j] with prohibited CLB locations
      listed in prohibit_xy
18.   if Place-and-route failed then
19.     prohibit_xy.remove($argmin_{xy \in \texttt{prohibit\_xy}} [\hat{\mathbf{R}} + \hat{\mathbf{s}}_\mathbf{j}]_{xy}$)
20.     goto Line 17
21.   end if
22.   $\mathbf{R}$ := $\mathbf{R}$ + get_stress(Acc[j])
23. end for
```

To avoid that high stress CLBs of the currently placed accelerator Acc[j] overlap with those in previously placed accelerators Acc[1],...,Acc[j-1], we prohibit the placement to specific CLB locations for Acc[j] (Lines 9 to 17), identified by their (x, y) coordinates, if the following condition is satisfied:

$$[\hat{\mathbf{R}}]_{xy} > \frac{1}{L_j} \sum_{uv} [\hat{\mathbf{s}}_\mathbf{j}]_{uv} \tag{6}$$

$$\text{with } \hat{\mathbf{R}} = \frac{1}{\max_{uv} [\mathbf{R}]_{uv}} \mathbf{R} \text{ and } \hat{\mathbf{s}}_\mathbf{j} = \frac{1}{\max_{uv} [\mathbf{s_j}]_{uv}} \mathbf{s_j}$$

where L_j is the number of used CLBs by the currently place-and-routed accelerator Acc[j]. $\hat{\mathbf{R}}$ and $\hat{\mathbf{s}}_\mathbf{j}$ are normalized stress matrices of $\mathbf{R}$ and $\mathbf{s_j}$. In earlier iterations, the reference distribution is less even, which implies that few CLB locations in the reference distribution have much higher values than the others, and therefore it is less likely that the condition (Eq. (6)) is satisfied. In turn, fewer locations are prohibited for placement in earlier iterations, which implies less timing impact on slower accelerators. If place-and-route fails due to too many prohibited CLB locations, the locations xy where the stress overlapping $[\hat{\mathbf{R}} + \hat{\mathbf{s}}_\mathbf{j}]_{xy}$ is lowest are removed from prohibit_xy (Line 19), and place-and-route is re-executed with the relaxed constraints.

With synthesis time stress diversification, high stress CLBs from different accelerators are placed to different CLB locations, and thus better intra-region stress distribution can be achieved during runtime placement. As shown in Fig. 5, after applying both stress-aware runtime placement and synthesis time stress diversification for dynamic stress, the maximal transistor toggle rate is further reduced by additional 44% from 5.0 to 2.8 million toggles/s.

VI. Experimental Evaluation

A. Experimental Setup

The presented method is evaluated in a reconfigurable architecture implemented on a Xilinx Virtex-5 LX110T FPGA. Each region consists of 4×20 CLBs with eight 6-input LUTs per CLB. Our method performs optimizations on CLB granularity.

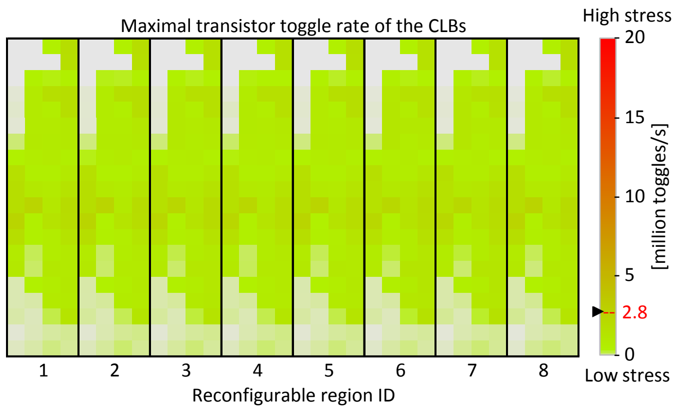
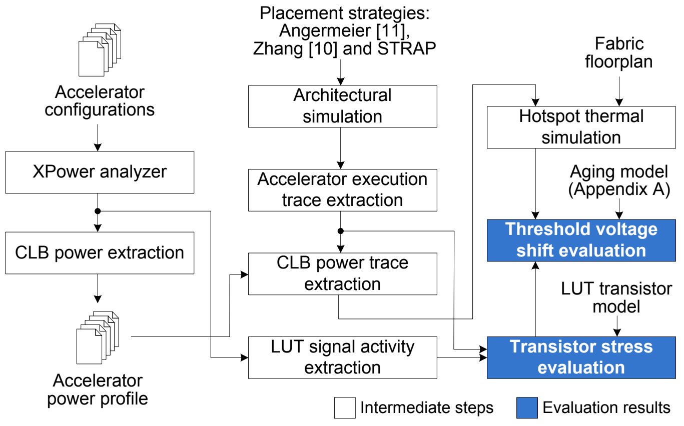


Fig. 5: Transistor stress distribution using both stress-aware runtime accelerator placement and synthesis time stress diversification



Accelerator configurations
Placement strategies: Angermeier [11], Zhang [10] and STRAP
Fabric floorplan
Architectural simulation
Hotspot thermal simulation
XPower analyzer
Accelerator execution trace extraction
Aging model (Appendix A)
Threshold voltage shift evaluation
CLB power extraction
CLB power trace extraction
LUT transistor model
LUT signal activity extraction
Transistor stress evaluation
Accelerator power profile
Intermediate steps
Evaluation results

Fig. 6: Experimental flow to evaluate the transistor stress and threshold voltage shift

To evaluate the actual stress for each transistor, we use a transistor-level model of LUTs that is based on [2, 19] using NMOS pass transistors for multiplexers. To evaluate the threshold voltage shift due to stress, state-of-the-art aging models are employed (detailed equations and used parameters are given in Appendix A). A complex H.264 video encoder that uses nine distinct accelerators (first column in Table II) was chosen as target application since video and image processing are typical applications for reconfigurable architectures [1]. The resource usage of each accelerator within one region ranges from 8.8% to 66.3%. A SystemC-based cycle-accurate architectural simulator is used to evaluate the STRAP method for systems that differ in the number of reconfigurable regions and runtime strategies, and to compare it with related work. 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. Alg. 1 is integrated into the simulator and Alg. 2 is implemented as a script that generates the placement constraints and automatically calls the Xilinx place-and-route tools.

The experimental evaluation flow is shown in Fig. 6. The placed-and-routed accelerators are fed to Xilinx XPower analyzer to obtain the signal activities and power consumption of logic elements and nets. The power consumption is then aggregated to CLB granularity by summing up the power consumed by LUTs and their fan-in nets in one CLB. The leakage power of a region is proportional to its size. Architectural simulation produces the accelerator execution trace, i.e. the complete execution and idle history of each accelerator in each region. Together with the power profile of each accelerator, we obtain the power trace of each CLB. The power trace and the fabric floorplan of the FPGA (based on a die image acquired from chipworks.com) is then fed into Hotspot[2] [20] to obtain the temperature trace of each CLB, which will be used to evaluate the threshold voltage shift. The accelerator execution trace and the LUT signal activities of each accelerator are combined to calculate the LUT signal activities for the regions. This is then used to evaluate the stress of individual transistors by using the LUT transistor model [2, 19].

We vary the number of regions from 5 to 12 and perform separate evaluation of the proposed method for dynamic and static stress mitigation. The baseline system does not use any stress distribution method. For comparison, we implemented two state-of-the-art stress distribution methods [10, 11]. Zhang *et al.* [10] use three different configurations for each accelerator and switch between them to migrate stress, whereas Angermeier *et al.* [11] consider the peak stress of regions to place an accelerator (cf. Section I). As proposed for STRAP, we extended [10, 11] to replace an accelerator if its reconfigurable region has not been reconfigured for 100 million cycles (see Section IV). This improvement reduces the peak stress of [10, 11] and thus makes the comparison with state-of-the-art more competitive. Regarding temperature variation, we perform a conservative comparison. To calculate the threshold voltage shift for [10, 11] we use the lowest temperature that was observed for any CLB at any time in the obtained temperature trace as constant temperature for all CLBs, while we use the highest observed temperature for STRAP. Thus, the threshold voltage shift reported for [10, 11] is a lower limit, whereas the one for STRAP is a conservative upper limit.

B. Results

Fig. 7 shows the maximal (lighter color) and average (darker color; arithmetic mean) dynamic transistor stress, measured in million toggles/s, in the whole reconfigurable fabric for systems with different number of regions. Similarly, Fig. 8 shows the static transistor stress measured in normalized stress time (stress duty cycle), i.e. the fraction of operation time the transistor is under static stress. The figures show that all methods reduce the average stress compared to the baseline because they all distribute the stress to more transistors. While the reduction of the average stress is similar for all three methods, the reduction of the maximal stress (i.e. the critical part for system mean time to failure/MTTF) differs significantly. The reason is that Angermeier *et al.* [11] perform only runtime inter-region stress distribution, while Zhang *et al.* [10] perform only synthesis time intra-region stress distribution for individual accelerators. In contrast, STRAP performs cross-layer stress-aware placement at runtime and synthesis time, which leads to the highest reduction of maximal stress in all evaluated cases. The reduction of the maximum stress by STRAP in Fig. 7 and 8 is up to 64% and 35% higher than the closest competitors w.r.t. dynamic and static stress, respectively. Table I summarizes the stress reduction shown in Fig. 7 and 8.

We calculate the MTTF improvement by assuming that a device fails when ΔV_{th} of any transistor exceeds 50% of its original value (V_{th0}). The MTTF improvement due to dynamic and static stress reduction is shown in the last two columns in

[2]smallest possible heat spreader and heat sink with 10 µm thickness, ambient temperature 50°C

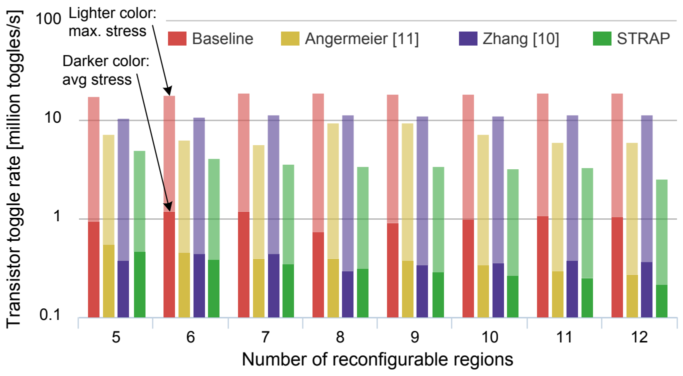


Fig. 7: Comparison to related work for static stress in systems with different number of reconfigurable regions

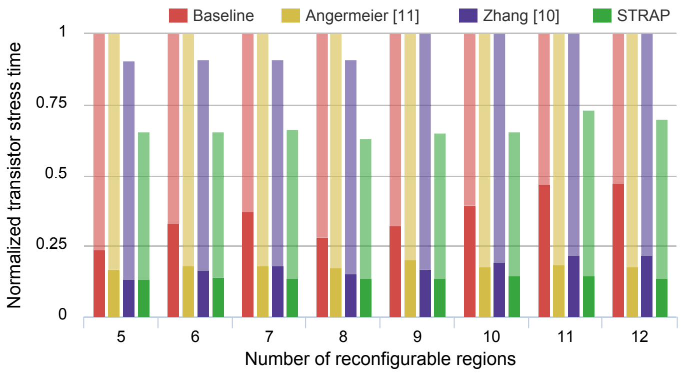


Fig. 8: Comparison to related work for static stress in systems with different number of reconfigurable regions

Table I. With the STRAP method, the MTTF improvement relative to the baseline is 413% and 13% in average for HCI and BTI aging, respectively. Relative to the closest competitors, STRAP achieves up to 177% and 14% MTTF improvement w.r.t. HCI and BTI aging, respectively.

Fig. 9 analyzes the detailed per-transistor stress (static and dynamic) of a system with 8 reconfigurable regions. It compares STRAP that optimizes for static stress or dynamic stress against the baseline. Each point represents the stress value of one transistor in the reconfigurable fabric. Points closer to the lower-left corner denote less dynamic and less static stress. The horizontal and vertical lines represent the upper boundary for dynamic stress and static stress in all three cases. Although during optimization only one type of stress is considered, actually both types of stress are reduced simultaneously. With STRAP targeting the static stress distribution, we observe a reduction of 52% in dynamic and 38% in static stress. When targeting dynamic stress, STRAP delivers 82% reduction in dynamic stress and 21% reduction in static stress. The reason behind the reduction of both stress types is that STRAP implicitly distributes the transistor usage as well, which reduces the individual static and dynamic transistor stress.

STRAP's stress-diversifying logic placement at synthesis time may affect the accelerator frequency. The timing impact of the placement constraints is shown in Table II. 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. On average, the maximum accelerator frequency decreases by 7%. Since accelerators with longer critical path (lower maximum frequency) are imposed with fewer constraints (see Section V), their maximum frequencies are less affected.

TABLE I: Reduction of avg./max. stress and MTTF increase of STRAP and state-of-the-art [10, 11] compared to the baseline; averaged over all numbers of reconfigurable regions

Strategy	Reduction of avg. stress[%]		Reduction of max. stress[%]		MTTF improvement[%]	
	dyn.	stat.	dyn.	stat.	HCI	BTI
Angermeier [11]	60.6	47.4	61.2	0.02	157.7	0.0
Zhang [10]	62.6	49.6	39.9	4.5	66.4	2.3
Our STRAP	**67.9**	**59.6**	**80.5**	**33.1**	**413.0**	**13.4**

TABLE II: Change in maximum frequency of accelerators

Accelerator	Original [MHz]	STRAP [MHz]	Worstcase Δ [%]
Clip3	133	122–130	8.2
CollapseAdd	158	158–158	0.0
LF_BS4	121	115–120	5.0
LF_Cond	146	132–140	9.6
PointFilter	89	89	0.0
QuadSub	257	232–257	9.7
SADrow_4	100	96–96	4.0
SAV	139	120–138	13.7
Transform	167	145–166	13.2
System freq.	89	89	0.0

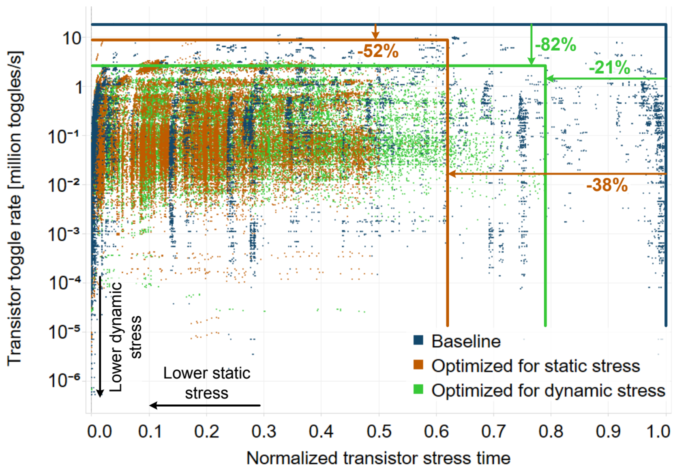


Fig. 9: Transistor stress for different STRAP optimization goals

The maximum *system* frequency is however limited by the accelerator with the longest critical path, i.e. PointFilter, which runs at $f_{min} = 89$ MHz (cf. Section V). Therefore, STRAP has no negative timing impact on the whole system.

For the evaluated systems, the worst-case overhead of the runtime accelerator placement algorithm ranges from 1.2 ms for 5 regions to 6 ms for 12 regions on a SPARC V8 LEON3 processor running at 100 MHz.

VII. Conclusions

The dependable operation of runtime reconfigurable architectures is threatened by aging. This paper presented the novel cross-layer stress-aware placement method STRAP, which for the first time combines 1) stress-aware runtime placement of accelerators, and 2) stress-diversifying logic placement at synthesis time. STRAP mitigates aging by balancing stress both within a reconfigurable region as well as across all reconfigurable regions in the system. Compared to state-of-the-art methods, STRAP significantly reduces the maximum dynamic and static stress by up to 64% and 35% with negligible impact on application performance, respectively. As a result, STRAP achieves up to 177% and 14% MTTF improvement relative to the closest competitors w.r.t. HCI and BTI aging, respectively.

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] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems", *EURASIP Journal on Emb. Systems*, pp. 1–19, 2006.

[2] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung, "Degradation in FPGAs: Measurement and modelling", in *Int'l Symp. on Field Progr. Gate Arrays*, 2010, pp. 229–238.

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

[4] 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.

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

[6] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends", in *IEEE/ACM Design Automation Conference (DAC)*, 2013.

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

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

[9] E. Stott, J. Wong, and P. Cheung, "Degradation analysis and mitigation in FPGAs", in *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2010, pp. 428–433.

[10] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures", in *IEEE International Test Conference*, 2013.

[11] J. Angermeier, D. Ziener, M. Glass, and J. Teich, "Stress-aware module placement on reconfigurable devices", in *Int'l Conf. on Field Progr. Logic and Appl.*, 2011, pp. 277–281.

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

[13] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs", *IEEE Trans. on Computer-Aided Design of Integr. Circuits and Sys.*, vol. 26, no. 2, pp. 203–215, 2007.

[14] A. N. Nowroz and S. Reda, "Thermal and power characterization of field-programmable gate arrays", in *Int'l Symposium on FPGAs*, 2011, pp. 111–114.

[15] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring temperature in FPGA based SoCs", in *Int'l Conf. on Computer Design (ICCD)*, 2005, pp. 634–637.

[16] Y. Cao, J. Velamala, K. Sutaria, M.-W. Chen, J. Ahlbin, I. Sanchez Esqueda, M. Bajura, and M. Fritze, "Cross-layer modeling and simulation of circuit reliability", *IEEE Trans. on CAD of ICs and Systems*, vol. 33, no. 1, pp. 8–23, 2014.

[17] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms", in *International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 478–485.

[18] C. Ma, H. Mattausch, M. Miyake, T. Iizuka, M. Miura-Mattausch, K. Matsuzawa, S. Yamaguchi, T. Hoshida, M. Imade, R. Koh, T. Arakawa, and J. He, "Compact reliability model for degradation of advanced p-MOSFETs due to NBTI and hot-carrier effects in the circuit simulation", in *Reliability Physics Symp. (IRPS)*, 2013, pp. 2A.3.1–2A.3.6.

[19] 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.

[20] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage VLSI design", *IEEE Trans. VLSI Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[21] K. Joshi, S. Mukhopadhyay, N. Goel, and S. Mahapatra, "A consistent physical framework for N and P BTI in HKMG MOSFETs", in *Reliability Physics Symposium (IRPS)*, 2012, pp. 5A.3.1–5A.3.10.

[22] "Predictive Technology Model", http://ptm.asu.edu/.

[23] C. Hu, S. C. Tam, F.-C. Hsu, P.-K. Ko, T.-Y. Chan, and K. W. Terrill, "Hot-electron-induced MOSFET degradation – model, monitor, and improvement", *IEEE Trans. on Electron Devices*, vol. 32, no. 2, pp. 375–385, 1985.

[24] T. Ning, C. Osburn, and H. Yu, "Emission probability of hot electrons from silicon into silicon dioxide", *Journal of Applied Physics*, vol. 48, no. 1, pp. 286–293, 1977.

Appendix

A. Details of the aging model

We employed state-of-the-art physics-based aging models for BTI and HCI effects to evaluate the threshold voltage shift depending on stress, temperature and time. The BTI aging model is adopted from [17, 21]:

$$\begin{aligned}
\Delta V_{th} &= \frac{q}{C_{ox}}\left(\Delta N_{IT} + \Delta N_{ET} + \Delta N_{OT}\right) \qquad (7)\\
\Delta N_{IT} &= A(V_{GS} - V_{th0} - \Delta V_{th})^{\Gamma_{IT}} e^{-\frac{E_{AIT}}{kT}} d^{\frac{1}{6}} t^{\frac{1}{6}}\\
\text{with } d &= \frac{\Lambda}{1 + \sqrt{\frac{1-\Lambda}{2}}}\\
\Delta N_{ET} &= B(V_{GS} - V_{th0} - \Delta V_{th})^{\Gamma_{ET}} e^{-\frac{E_{AET}}{kT}} \Lambda\\
\Delta N_{OT} &= C(1 - e^{-\left(\frac{t}{n}\right)^{\beta_{OT}}})\Lambda\\
\text{with } n &= \eta(V_{GS} - V_{th0} - \Delta V_{th})^{\frac{\Gamma_{OT}}{\beta_{OT}}} e^{\frac{E_{AOT}}{kT\beta_{OT}}}
\end{aligned}$$

where T denotes temperature, t operating time and Λ stress duty cycle. A, B, C are fitting parameters depending on the manufacturing process [21]. Device dependent parameters such as C_{ox} and V_{th0} are extracted from the PTM 22-nm HKMG model [22]. The supply voltage is 1.0 V. All other model parameters are as in [21].

The HCI aging model is adopted from [17, 18, 23]:

$$\Delta V_{th} = \frac{q}{C_{ox}} D \left(t \frac{rQ_{DS}}{W} e^{-\frac{\phi_{it}}{q\lambda E_m}}\right)^{\frac{1}{2}} \qquad (8)$$

where t denotes operating time, r the toggling rate, and Q_{DS} the amount of charges flowing through the channel during one toggle calculated using PTM 22-nm HKMG model [22]. ϕ_{it} and E_m are extracted from [23]. λ is temperature dependent and is calculated based on [24]. W is assumed to be the same as the transistor channel length. D is a fitting parameter depending on the transistor geometry and manufacturing process [17].