An Efficient MILP-Based Aging-Aware Floorplanner for Multi-Context Coarse-Grained Runtime Reconfigurable FPGAs

Bo Hu, Mustafa Shihab, Yiorgos Makris, Benjamin Carrion Schaefer, Carl Sechen
{bo.hu, mustafa.shihab, yiorgos.makris, schaferb, carl.sechen}@utdallas.edu
Department of Electrical & Computer Engineering, The University of Texas at Dallas

***Abstract*—Shrinking transistor sizes are jeopardizing the reliability of runtime reconfigurable Field Programmable Gate Arrays (FPGAs), making them increasingly sensitive to aging effects such as Negative Bias Temperature Instability (NBTI). This paper introduces a reliability-aware floorplanner which is tailored to multi-context, coarse-grained, runtime reconfigurable architectures (CGRRAs) and seeks to extend their Mean Time to Failure (MTTF) by balancing the usage of processing elements (PEs). The proposed method is based on a Mixed Integer Linear Programming (MILP) formulation, the solution to which produces appropriately-balanced mappings of workload to PEs on the reconfigurable fabric, thereby mitigating aging-induced lifetime degradation. Results demonstrate that, as compared to the default reliability-unaware floorplanning solutions, the proposed method achieves an average MTTF increase of 2.5× without introducing any performance degradation.**

I. Introduction

Coarse-Grained Runtime Reconfigurable Architectures (CGRRAs) achieve superior computation and area efficiency by enabling dynamic adaption to changing workloads. CGRRAs consist of PEs that contain Arithmetic Logic Units (ALUs) and Data Manipulation Units (DMUs); a block diagram of a typical CGRRA is shown in Fig. 1. State-of-the-art CGRRAs [1] are supported by a CAD flow from high-level synthesis (HLS) to placement and routing. Higher design productivity and reduced design time are achieved by employing widely used programming languages, such as ANSI-C or C++ in HLS. Furthermore, dividing an application into multiple contexts, as shown in Fig. 1, enables multi-context runtime reconfiguration, where the CGRRA fabric is time-shared among the contexts at single clock-cycle granularity.

The frequent reconfiguration of multi-context CGRRAs significantly increases the stress on the fabric and, thereby, accelerates aging mechanisms such as NBTI, HCI (Hot Carrier Injection), EM (Electromigration) and TDDB (Time-Dependent Dielectric Breakdown). These mechanisms are the dominant reliability degradation factors and are usually reflected by an increase in the threshold voltage of transistors [2], [3]. Commercial FPGA technology mappers and placement tools, however, do not consider circuit lifetime; rather, they mainly focus on reducing area while meeting timing constraints. In this work, we address this shortcoming by proposing an aging-aware CAD flow, specifically targeted to CGRRAs, which extends the MTTF for these architectures by balancing PE usage. Our method is based on a MILP formulation which incorporates path delay constraints and seeks the optimal PE usage for mitigating aging-induced lifetime degradation without incurring performance overhead.

Aging-aware CAD flows have been previously developed for fine-grained runtime reconfigurable FPGAs, leveraging

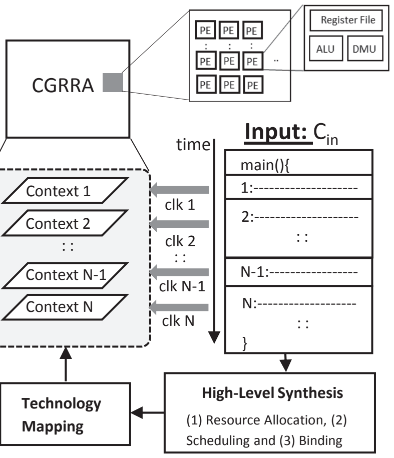


Fig. 1: CGRRA architecture and multi-context mapping flow.

their ability to reconfigure themselves. Indeed, aging mitigation in such FPGAs has been achieved by using spare FPGA resources in the re-mapping stage. This can be accomplished by changing the floorplan of the used logic resources to reduce the maximum stress and increase MTTF [4]. To date, the main approach has been the generation of multiple FPGA configurations to balance the stress on the used logic resources [5]–[8]. In [8] and [4], methods are proposed for reducing the maximum stress on Configurable Logic Blocks (CLBs) of fine-grained FPGAs, by periodically swapping between different CLB configurations. The authors of [5] presented a re-mapping Design Space Exploration method that used an aging-aware scheduler to maximize the system MTTF on a heterogeneous, dynamic, partially reconfigurable, fine-grained FPGA. In [6], an aging-aware FPGA floorplanner was proposed to reduce the stress time at the block- or system-level, by using a delay-based degradation estimation model.

With regards to CGRRAs, the authors of [3] proposed periodic re-mapping of the design to less-used regions by using two different configurations and switching between them to mitigate HCI degradation. The authors of [9] proposed a method to lower the temperature of the CGRRA by generating different configurations that incrementally use different PEs. Similarly, the authors of [10] provided a rotation-based mapping strategy to balance stress on multi-context CGRRAs. However, none of the above approaches take into consideration the performance degradation introduced by these stress-time re-balance strategies. Indeed, when spare resources on CGRRAs are used to mitigate stress and stress time, there may be an increase in the critical path delay. To alleviate this shortcoming of prior solutions, herein we propose an algorithm for generating aging-aware floorplans for multi-context CGRRAs, which not only mitigates stress time and temperature of PEs,

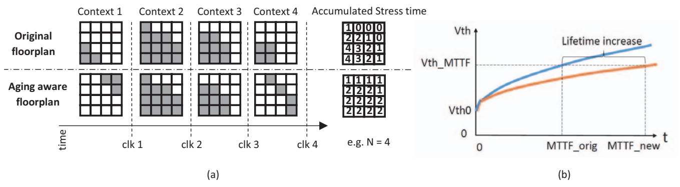


Fig. 2: (a) Accumulated stress time balance to achieve aging aware floorplan. (b) Threshold voltage shift (V_{th}) vs. MTTF for both the original and re-mapped floorplans [4].

but also incorporates path delay constraints to ensure that the critical path delay does not increase.

II. Multi-context CGRRA architecture and MTTF

The configuration of a multi-context CGRRA starts with an input of ANSI-C. HLS and technology mapping are, then, performed to divide it into contexts, such that a different context is loaded onto the device in every clock cycle, as shown in Fig. 1. The number of contexts is determined by the desired latency of the circuit and vice versa. Division into multiple contexts is shown in the first row of Fig. 2(a), which illustrates an example of an original floorplan. In this case, 4 contexts are mapped to the CGRRA, one per clock cycle. The gray squares denote the PEs being used in that particular context. The required fabric size for the CGRRA is determined by the context which has the maximum number of PEs.

In the example of Fig. 2(a), suppose that the stress time of each used PE is 1 unit, as indicated by the gray PEs. Summed over all contexts, some PEs have an accumulated stress time of 4 units after aging-unaware floorplanning. Since the lifetime of CGRRAs is determined by the failing time of the PEs with the highest stress time, we propose a method to increase CGRRA lifetime by re-mapping operations onto PEs in each context such that the stress time of all PEs on the fabric is leveled. As shown in the second row of Fig. 2(a), the new aging-aware floorplan decreases maximum accumulated stress time to 2, while PE usage is more evenly distributed across the fabric.

III. NBTI modeling and MTTF Computation

NBTI usually dominates the reliability degradation factors and is measured by the increase in the magnitude of the threshold voltage (V_{th}). We follow the same formulation as in most previous work [2], [3], [7], [10] for expressing V_{th} in the presence of NBTI, which is shown in (1),

$$V_{th} = A_{NBTI} \times (ST_{t_0 \to t})^n \times e^{(-Ea/kT)} \times V_{th0} \qquad (1)$$

where A_{NBTI} is a technology-dependent factor, n is a fabrication-dependent constant, k is Boltzmann's constant, Ea is the activation energy, T is the temperature, V_{th0} is the starting V_{th}, t_0 is the corresponding starting time when $V_{th} = V_{th0}$ and $ST_{t_0 \to t}$ is the stress time from t_0 till time t, which equals $SR \times (t_0 \to t)$, where SR is the stress rate and equals the duty cycle (*i.e.,* percentage of time transistor is on).

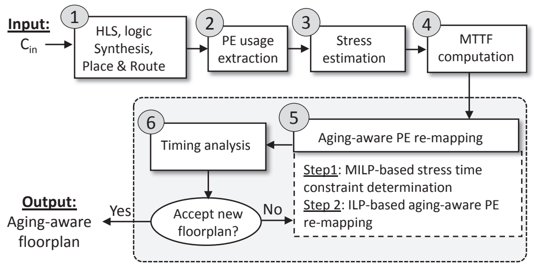
The MTTF model that we use assumes failure due to a V_{th} increase by V_{th_shift}. This V_{th_shift} is defined as V_{th_MTTF} - V_{th0}, where V_{th_MTTF} is the threshold voltage corresponding to the MTTF, as shown in Fig. 2(b). Therefore, computing V_{th} allows us to compute the MTTF of the CGRRA for the default aging-unaware PE floorplan. In our example, the original case without aging-aware remapping is shown as the blue curve in Fig. 2(b), while the case with aging-aware remapping is shown as the orange curve. As in previous work, we assume that the fabric will fail when the V_{th_shift} reaches a certain threshold (*e.g.* 10% [3]) and the corresponding operation time will be the MTTF. As may be observed in Fig. 2(b), due to the reduced maximum accumulated stress time across all PEs, the orange curve has a lower slope compared to the blue curve, which results in a higher MTTF. The critical factors which determine V_{th_shift} are the highest stress time and temperature among all PEs. For a given PE, the accumulated stress time is the total stress time of the PE after execution of all contexts. Because the temperature is also affected by the stress time [11], reduction of the highest accumulated stress time on a PE will prolong its lifetime.

In order to obtain the original MTTF (*e.g.,* the blue curve in Fig. 2(b)), we first determine the stress time for each PE in the provided floorplan. Since different functional units may be mapped to a PE in different contexts, the stress time of each PE in every clock cycle (every context) may be different. For example, the characterization of the ALU and DMU within one PE shows their delays as 0.87 ns and 3.14 ns, respectively, which, divided by the clock period, gives their stress rates (SR). Thus, based on the default mapping obtained in the aging-unaware mapping generation phase and the corresponding operations mapped on each PE in every context, we obtain the stress time map of each PE. Next, we leverage the commonly-used thermal simulator HotSpot [11] to compute the temperature of each PE. The thermal simulator inputs the stress time maps and floorplans generated in the aging-unaware mapping generation phase and generates a thermal map for each context. The PE with the maximum accumulated temperature across all contexts is, then, identified and the corresponding stress times are obtained from the floorplan. Then, V_{th_shift} is computed using (1) and the blue plot in Fig. 2(b) is used to obtain the failing point MTTF. The same process is used to obtain the MTTF for the aging-aware floorplan, which corresponds to the orange curve and is larger due to the reduction in the maximum accumulated stress times.

IV. Proposed CAD design flow for CGRRAs

An overview of the proposed CAD design flow for CGRRAs is shown in Fig. 3. The input to this flow is a behavioral description for HLS and the output is an aging-aware floorplan consisting of N contexts, which are used to configure the CGRRA fabric every clock cycle. It should be noted that this is different from leveling the PE usage across contexts, as each PE can execute different types of operations of different bitwidths and, hence, can produce different amounts of stress time. Below, we describe the proposed approach in detail.

Phase 1: Aging-unaware PE Mapping and MTTF Computation: This first phase takes as input the behavioral description to be mapped onto the CGRRA and executes the complete design flow from HLS to bitstream generation. This includes the technology mapping onto the PEs and the placement and



Input:
C_in
1
HLS, logic Synthesis, Place & Route
2
PE usage extraction
3
Stress estimation
4
MTTF computation
5
Aging-aware PE re-mapping
Step1: MILP-based stress time constraint determination
Step 2: ILP-based aging-aware PE re-mapping
6
Timing analysis
Accept new floorplan?
Yes
No
Output:
Aging-aware floorplan

Fig. 3: Aging-aware CGRRA re-mapping design flow.

routing. This is accomplished using the commercial CAD flow provided with the CGRRA [1]. The result is an aging-unaware configuration that minimizes the bounding box area of the used PEs while meeting the specified timing constraints. This phase continues by taking as input the PEs used in each context, as well as the operations mapped on each PE, and calculates the *stress time* of each PE and its temperature. Based on NBTI modeling and the MTTF computation Equation (1) in Section III, the MTTF is computed. The result from this phase is the baseline PE mapping for each of the contexts.

Phase 2: Aging-aware re-mapping Design Flow: This phase generates new configurations that optimize the lifetime without incurring any delay increase. This is accomplished by re-binding the operations in each context to new PEs in order to increase the MTTF and the lifetime of the CGRRA. As shown in (1), the dominant factor that affects V_{th_shift} is the accumulated stress time. The goal is, therefore, to reduce the highest accumulated stress time among all PEs. Intuitively, this could be achieved by spreading the PE usage as much as possible. This naïve approach, however, can cause significant delay increase due to longer wire lengths. We, therefore, propose a re-binding strategy that is aware of the path delays while attempting to maximize the lifetime.

V. Effective delay- and aging-ware re-mapping

A. Primary aging-aware ILP formulation

In this section, we formulate the operation-to-PE re-binding problem as an ILP:

$$\begin{array}{ll}
\text{Minimize:} & \text{Accumulated stress time among all PEs} \\
\text{Constraints:} & \text{Each operation is bound to exactly one PE} \\
 & \text{Path delay on each path} \leqslant \text{critical path delay} \\
 & \text{Critical path operations frozen in original PEs}
\end{array} \tag{2}$$

Let M be the number of operations and let C be the number of contexts. Also, define the set P to be the number of timing paths from primary inputs to primary outputs after the original aging-unaware placement and routing across all contexts. In order to map a design onto the available CGRRA resources with N PEs, the ILP formulation which aims at minimizing the accumulated stress time among all PEs under the above three constraints will need to find an optimal solution among $2^{(M\times N\times C)}$ possible cases through reasoning on $M \times N \times C$ binary variables with $M \times N + P$ constraints. The main problem with this approach is that it does not scale well; as we observed for larger benchmarks, the ILP solver could not find a solution within a reasonable amount of time (5 days). This is a well-known problem with ILP-formulations, especially as the problem grows rapidly to extremely large size, as in our case. Thus, we propose to relax the problem formulation to a two-step MILP formulation, in order to provide an effective aging-aware floorplan optimization.

B. ILP based delay- and aging-aware re-mapping

Our ILP formulation is shown in (3), where c is the number of contexts, M_i is the number of operations in context i and N is the number of PEs. The probability that operation j in context i is mapped to PE_k is represented as OP_{ijk}. Since the k^{th} PE in context i is either used or not for operation j, $OP_{ijk} \in \{0,1\}$. The stress time of the j^{th} operation in context i is represented as $ST(OP_{ij})$, while the accumulated stress time constraint for each PE is represented as ST_{target}.

$$\begin{array}{lll}
\text{ObjFunc:} & \text{Null} & \\
\text{subject to:} & \displaystyle\sum_{i=1}^{c}\sum_{j=1}^{M_i} OP_{ijk} \times ST(OP_{ij}) \leqslant ST_{target} & \\
 & \displaystyle\sum_{k=1}^{N} OP_{ijk} = 1 & i \in [1,...,c] \\
 & \text{For operation } (OP_{nip}) \text{ on critical path:} & \\
 & \{OP_{nip} \rightarrow PE_{k\_orig}\} & p \in [1,...,P_{ni}] \\
 & \text{For operation } (OP_{nip}) \text{ off critical path:} & \\
 & \displaystyle\sum_{p=1}^{P_{ni}} wire\ length(OP_{nip}) \leqslant \frac{\text{CPD}-\sum_{p=1}^{P_{ni}} PEdelay(OP_{nip})}{\text{unit wire delay}} & \\
 & \text{if LP then each } OP_{ijk} \in [0,1] & \\
 & \text{if mixed LP/ILP, linear } OP_{ijk} \in [0,1] & {\scriptstyle j\in[1,\ldots,M_i]} \\
 & \text{and binary } OP_{ijk} \in \{0,1\} & {\scriptstyle k\in[1,\ldots,N]}
\end{array} \tag{3}$$

Step 1: MILP-based stress time constraint determination

A suitable starting value for the accumulated stress time constraint (ST_{target}) is required in (3). This starting value should be a lower bound for an actual feasible ST_{target}. To this end, we execute (3) without considering the Critical Path delay (CPD) constraints. Since we allow the delays to exceed the CPD during this step (effectively, making it delay-unaware), the obtained accumulated stress time target (ST_{target}) will certainly be a lower bound. This lower bound is determined based on a binary search strategy, in which ST_{target} is iteratively set between an upper value (ST_{up}) and a lower value (ST_{low}), all the while ignoring the critical path and path delay constraints in (3). ST_{up} is the highest accumulated stress time among all PEs from the original aging-unaware floorplan, and ST_{low} is taken as the average accumulated stress time among all PEs, also from the original aging-unaware floorplan. The smallest value of ST_{target} that yields a valid (albeit delay-unaware) floorplan solution represents the initial value of ST_{target} that is used as the initial (lower bound value) for the target accumulated stress time constraint in (3). Due to the large scale of the ILP formulation, we propose to relax this problem to a two-step MILP formulation. The first step is linear relaxation, *i.e.,* to solve it as an LP formulation, by not forcing OP_{ijk} to be an integer, but rather a real number between 0 and 1, that is, $OP_{ijk} \in [0.0, 1.0] \in \mathbb{R}$. We then set those OP_{ijk} values which are very close to 1 (> 0.95) to be exactly 1 (pre-mapping for the next ILP step) (we did try other well-known approaches such as randomized rounding, but they did not work as well). To get the entire operator mapping solution, we pre-map the operations to PEs resulting from the previous LP formulation solution and then solve it as an ILP to map the rest of the operations to available PEs, where OP_{ijk} is set back to integer type ($OP_{ijk} = [0, 1]$).

Step 2: ILP-based aging-aware PE re-mapping

The performance of a benchmark is usually evaluated by its critical path delay, consisting of delays within PEs and wire delays between PEs. In our multi-context CGRRA architecture, the critical path delay is the longest path delay among all contexts. While the delay within PEs is not changed, the wire delays may change (even significantly) due to new operation-to-PE binding. Even if the wire delay on the critical path is not increased, it is possible that the delay on other paths might be increased to larger than the original critical path delay.

The ILP formulation in (3) includes two path delay constraints. The first constraint is the critical path constraint. Assume there is a set of N_i critical paths in context i. For each critical path n in the set N_i, let P_{ni} be the number of operations on that critical path in context i. OP_{nip} represents the p^{th} operation on critical path n in context i. Operation OP_{nip} is bound to its original PE (PE_{k_orig}). This constraint guarantees that the critical path delay is not increased, by freezing the location of all operations on a critical path. Meanwhile, the second constraint deals with operations that are on paths which are not (currently) critical. These path delays will be strictly limited to have no more delay than the critical path delay (CPD), where CPD is chosen as the maximum CPD among all contexts. In this way, the delay of the entire benchmark is not increased, while at the same time, it provides flexibility for the PEs to move around. Since the path delay consists of delays within PEs, as well as wire delays between PEs, $PEdelay(OP_{nip})$ is the delay within one PE for operation OP_{nip} for $p \in [1, ..., P_{ni}]$ and $wire\ delay(OP_{nip})$ is the delay of the wire driven by operation OP_{nip}. In general, the path delay can be formulated as:

$$Path\ delay = \sum_{p=1}^{P_{ni}} PEdelay(OP_{nip}) + \sum_{p=1}^{P_{ni}} wire\ delay(OP_{nip}) \quad (4)$$

In the CGRRA architecture, the wire lengths between PEs are long and, hence, have some number of buffers on the path from one PE to another. Inserting buffers in this manner causes the delay of the "wire" to be largely proportional to its length (rather than quadratic, as it would be without any buffers). Through simulation, we have determined what the proportionality constant is between the length of the wire and its delay, and this is termed the unit wire delay. Furthermore, when a PE drives a fanout greater than one, we have found that the buffers effectively shield the other fanouts, which are not on the path of interest, from the driver PE. Therefore, with reasonable accuracy, we can focus our attention only on the length of wire from the driver PE to the load PE on the path of interest, and then multiply this by the unit wire delay to obtain the wire delay associated with the driver PE. Thus, the path delay constraint can be formulated as: the wire delay on path n in context i cannot be larger than the available wire delay slack, which is defined as CPD - $\sum_{p=1}^{P_{ni}} PEdelay(OP_{nip})$. We then transform the path delay constraint problem into a path wire length constraint problem. Given a CPD from the original mapping floorplan, the path delay constraint for path n in context i can be formulated as:

$$\sum_{p=1}^{P_{ni}} wire\ length(OP_{nip}) \leqslant \frac{CPD - \sum_{p=1}^{P_{ni}} PEdelay(OP_{nip})}{\text{unit wire delay}} \quad (5)$$

The wire length connected to a driver PE (OP_{nip}) on a path is calculated by the Manhattan distance between the driver PE and its load PE on the path. This Manhattan distance is represented by: $wire\ length(OP_{nip})$. The above two constraints together assure that the critical path is not replaced by other longer delay paths.

1) Critical path constraint generation: The first constraint is the critical path constraint. Each operation (OP_{nip}) on the critical path of each context is bound to its original PE (PE_{k_orig}). While freezing the operation-to-PE binding for the critical paths can effectively avoid delay degradation, it has a drawback of overly restricting PE movement and, thus, affecting the minimization of accumulated stress time among all PEs. For the cases that the PE with the highest accumulated stress time is bound with operators on critical paths for each context, there is no way to reduce this stress time; thus, it is not possible to increase the MTTF. We therefore propose to rotate each so-called frozen critical path on each context among various orientations to minimize the critical path overlap of operations on particular PEs. It should be apparent that any path has 8 unique orientations. In particular: 1) the original

ALGORITHM 1: Aging-aware Re-mapping Design Flow

```
   CP: Critical path      CPD: Critical path delay
   Δ: stepsize of accumulated stress time constraint increase
1  /* Step 1: Stress time constraint determination */
2  execute (3) without CP and path delay constraints to get the
     lower bound for $ST_{target}$;
3  /* Step 2.1: Critical path constraint generation*/
4  rotate frozen CP in each context to minimize the CP overlap
     of operations on particular PEs;
5  /* Step 2.2: Path delay constraint generation */
6  for each timing path:
     accumulated wire delay $\leqslant$CPD−accumulated PE delay
7  path filter to get constraints for the M longest timing paths;
8  /* Step 2.3: ILP-based delay and aging aware re-mapping */
9  while (1) do
10     execute (3);
11     if solution exists then
12         if new CPD $\leqslant$ original CPD then
13             update new re-mapping solution;
14             break;
15         else
16             $ST_{target}$ = $ST_{target}$ + Δ;
17         end
18     else
19         $ST_{target}$ = $ST_{target}$ + Δ;
20     end
21 end
22 /* Step 3: MTTF calculation */
23 MTTF calculation based on the new re-mapped floorplan;
```

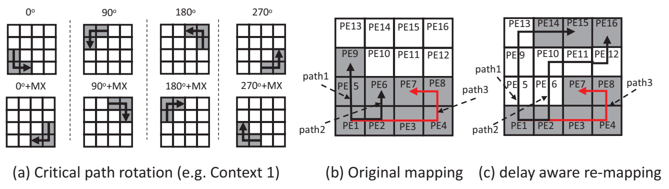


Fig. 4: (a) 8 unique orientations for one critical path. (b) Original mapping from aging-unaware floorplan. (c) Delay-aware re-mapped floorplan.

orientation, 2) rotate by 90 degrees clockwise, 3) rotate by 180 degrees clockwise, and 4) rotate by 270 degrees clockwise. The 5th through the 8th orientations are obtained by, for example, mirroring the x-coordinates of each of the above 4 orientations about the y-axis. Fig 4(a) shows an example of the 8 unique orientations for a critical path for a given context. For a given benchmark with C contexts, the number of possible critical path floorplan orientations across all contexts is 8^C. All of these orientation cases under the same accumulated stress time budget then need to be considered by (3) in order to find a valid floorplan. Thus, the runtime of our ILP formulation would increase by 8^C times in the worst case.

In order to reduce computational complexity without significant sacrifice of solution quality, we limited the combinational orientation cases for critical path rotation. A randomly selected orientation among the eight possible orientations is made for the critical paths for each of the contexts such that a) if the number of contexts is not more than eight, then no context has the critical paths in the same orientation, or b) if the number of contexts exceeds eight, then no orientation for the critical paths appears in the contexts more than the number of contexts integer-divided by 8 plus 1, and each possible orientation must be present among the contexts a number of times equal to the number of contexts integer-divided by 8.

2) Path delay constraint generation: Consider a case where the normalized delay within a stressed PE (*e.g.,* PE1) is 2, as shown in Fig 4(b), and that the unit wire delay is 1. Further, for this case, let the wire length between adjacent PEs is 1. In this case, the delay of *path1* is given by 2x3 (PE delay) +1x1x2 (the wire delay from PE1 to PE9) and thus the total path delay is 8. Also, the delay of *path3* is critical at 2x6+1x1x5 = 17. The upper bound on the wire length in *path1* is calculated as (17-2x3)/1 = 11 using (5). Therefore, its wire length slack is 11-1x1x2 = 9. Since PE1 is on the critical path, it must be frozen. However, PE5 and PE9 are off the critical path, and therefore they can be freely re-mapped within a wire length slack of 9. As shown in Fig 4(c), to mitigate the accumulated stress time of PE5 and PE9, these stressed PEs are replaced by PE14 and PE15. The re-mapped delay-aware floorplan shows that the wire length increase for *path1* is 5-2=3.

It turns out that the path delay constraint dominates the runtime of (3) due to the fact that the number of possible timing paths is exponentially related to the number of PEs. To reduce the number of timing paths that are considered, we apply the well-known Dijkstra's algorithm [12] to find the M longest timing paths among all paths. By default, we retain all paths whose initial delay is within 20% of the CPD. While there is a small chance that an unmonitored path produces a larger CPD than the original CPD (although we have not observed this is our experiments), we have added an extra step to check the new CPD after each re-mapping shown in Algorithm 1. If the new CPD is larger than the original CPD, the accumulated stress time constraint (ST_{target}) in (3) is relaxed by a stepsize of Δ until a valid floorplan exists.

3) ILP-based delay- and aging- aware re-mapping: Step 1 of Algorithm 1 generates the initial lower bound for the stress time target (ST_{target}). Meanwhile, step 2.1 generates the optimized orientations for the critical paths in each context. Next, the path wire length constraints are generated for each context in step 2.2. As shown in step 2.3, if the solution does not exist, the accumulated stress time constraint (ST_{target}) is incremented by a stepsize of Δ (line 19). This provides a higher probability for the MILP formulation to find a valid solution under the delay constraints. Note that, at line 10, (3) is executed each time ST_{target} is updated. If the solution exists (lines 11-17), the new CPD is calculated via a commercial timing analysis tool. If the new CPD is larger than the original CPD, ST_{target} is incremented and (3) is executed again until the new CPD is no larger than the original CPD. Upon such success, this re-mapped floorplan is produced as the aging-aware floorplan. Finally, the new MTTF is calculated based on the new re-mapped floorplan using (1).

VI. Experimental Results

Renesas Electronics' STP CGRRA was used as the target device and its commercial tool flow, called Musketeer [1], was used for HLS, Logic Synthesis (LS) and Placement & Routing (PAR). The HLS target frequency was fixed to 200MHz in all cases. HotSpot 6.0 [11] is the thermal simulator which was used to generate temperature reports. CPLEX [13] was used as the ILP solver and an open-source Python 2.7-based scripting tool, PuLP 1.6.1 [14], was used to call CPLEX and process results. Experiments were conducted on an Intel i7-6700 (3.50 GHz) CPU with 16 GB RAM, running CentOS 7.0.

A total of 27 synthesizable C benchmarks (B1-B27) were selected from different sources, ranging from low, to medium and to high with respect to the required number of contexts, size of CGRRA fabric and number of PEs. The first two columns of Table I show the number of contexts and the size of the CGRRA fabric (*i.e.,* array of PEs) on which we mapped these benchmarks, respectively. The rest of the table is organized in three super-columns for benchmarks with low, medium and high fabric usage rates, respectively. In each super-column, we provide the benchmark name, the number of PEs that it requires, as well as the MTTF increase achieved by our method. We note that the number of contexts equals the number of clock cycles (or latency) for the design. We also note that the CGRRA fabric size selected for each benchmark is based on the context with the maximum number of PEs.

For each benchmark, the MTTF improvement factor achieved by our delay-aware re-mapping formulation is reported with respect to the MTTF of the original aging-

TABLE I: MTTF increase for benchmarks with **low, medium and high** fabric usage rate

		low fabric usage rate				**medium** fabric usage rate				**high** fabric usage rate			
				MTTF incr. (x)				MTTF incr. (x)				MTTF incr. (x)	
context #	fabric size	benchmark	PE#	Freeze	Rotate	benchmark	PE#	Freeze	Rotate	benchmark	PE#	Freeze	Rotate
4	4*4	B1	24	1.94	1.94	B10	35	1.67	1.67	B19	52	1.18	1.52
4	8*8	B2	79	2.17	2.17	B11	148	1.44	1.82	B20	175	1.27	1.70
4	16*16	B3	192	2.26	2.28	B12	451	1.54	1.77	B21	554	1.76	2.00
8	4*4	B4	44	2.77	2.80	B13	62	2.05	2.36	B22	87	1.56	2.06
8	8*8	B5	142	2.69	2.89	B14	280	1.97	2.84	B23	327	1.48	1.98
8	16*16	B6	534	2.93	3.39	B15	1101	1.93	2.97	B24	1521	1.59	2.05
16	4*4	B7	88	3.76	3.85	B16	147	2.89	3.18	B25	193	1.61	2.06
16	8*8	B8	259	3.19	3.79	B17	531	2.62	2.94	B26	737	1.95	2.31
16	16*16	B9	1011	3.35	3.73	B18	2165	2.39	3.08	B27	3089	2.07	2.44
Avg.				**2.78**	**2.98**			**2.06**	**2.51**			**1.61**	**2.01**

unaware mapping by Musketeer. Results in the Columns under the *Freeze* header, indicate the MTTF improvement achieved without using the critical path delay rotation method, which implies that operations on the critical path cannot be moved to a different PE from the one in the original floorplan. In contrast, results in the Columns under the *Rotate* header, inicate the MTTF improvement achieved by the complete aging-aware re-mapping method. Evidently, the results are better in the latter case, corroborating the utility of Step 2.1 of Algorithm 1, wherein an optimized orientation selection is made for the critical paths in each context.

Fig. 5 summarizes the MTTF increase (×) achieved by the aging-aware mapping algorithm for all 27 benchmarks, grouped by CGRRA fabric size. The x axis label shows the fabric size, where the number following C represents the number of contexts and the number following F represents the number of PEs in one dimension (which is equal to the number of PEs in the other dimension). For example, C4F8 means that a benchmark is mapped to 4 contexts and a total of 8×8=64 PEs. For each of the 9 CGRA fabric sizes considered, we compare the achieved MTTF increase for 3 benchmarks, with low, medium, and high fabric utilization, shown in blue, red and yellow respectively. As intuitively expected, the lower the fabric utilization (*i.e.,* the more spare PEs it has available), the higher the MTTF increase.

VII. Conclusion

We presented a reliability-aware floorplanner which is specifically targeted towards multi-context CGRRAs and seeks to extend their MTTF by balancing PE usage. This floorplanner is based on an efficient MILP formulation which binds operations to PEs in the CGRRA in order to mitigate aging-induced lifetime degradation. Unlike previously developed solutions, our formulation ensures that no delay degradation is introduced, as compared to the original timing-driven aging-unaware floorplan generated by a commercial tool (Musketeer [1]) wherein our solution is integrated. Results on a multitude of benchmark circuits corroborate that, without introducing any delay overhead, the proposed solution can improve MTTF by an average of 2.5× over the original timing-driven aging-unaware floorplan.

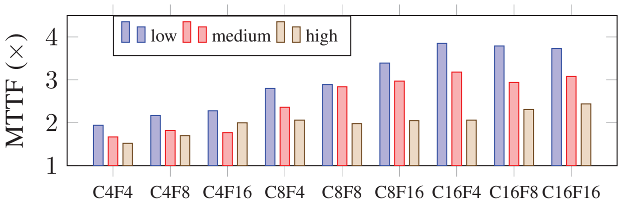


Fig. 5: Aging aware re-mapping for MTTF increase (×).

References

[1] Renesas, "STP, https://www.renesas.com/us/en/products/power-management/pmic/stp-engine.html," 2018.

[2] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and minimization of PMOS NBTI effect for robust nanometer design," in *ACM/IEEE Design Automation Conference*, 2006, pp. 1047–1052.

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

[4] 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, pp. 1–10.

[5] S. S. Sahoo, T. D. Nguyen, B. Veeravalli, and A. Kumar, "Lifetime-aware design methodology for dynamic partially reconfigurable systems," in *IEEE Asia and South Pacific Design Automation Conference*, 2018, pp. 393–398.

[6] Z. Ghaderi and E. Bozorgzadeh, "Aging-aware high-level physical planning for reconfigurable systems," in *IEEE Asia and South Pacific Design Automation Conference*, 2016, pp. 631–636.

[7] H. Zhang, M. A. Kochte, E. Schneider, L. Bauer, H.-J. Wunderlich, and J. Henkel, "STRAP: stress-aware placement for aging mitigation in runtime reconfigurable architectures," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 38–45.

[8] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, H.-J. Wunderlich, and J. Henkel, "Aging resilience and fault tolerance in runtime reconfigurable architectures," *IEEE Transactions on Computers*, vol. 66, no. 6, pp. 957–970, 2017.

[9] H. Afzali-Kusha, O. Akbari, M. Kamal, and M. Pedram, "Energy and reliability improvement of voltage-based, clustered, coarse-grain reconfigurable architectures by employing quality-aware mapping," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 480–493, 2018.

[10] J. Gu, S. Yin, and S. Wei, "Stress-aware loops mapping on cgras with considering nbti aging effect," in *ACM/IEEE Design Automation Conference*, 2017, pp. 1–6.

[11] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.

[12] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma, and N. Nosovic, "Dijkstra's shortest path algorithm serial and parallel execution performance analysis," in *International Convention MIPRO*, 2012, pp. 1811–1815.

[13] IBM-CPLEX, "Users manual for CPLEX, V12.1," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[14] S. Mitchell, M. OSullivan, and I. Dunning, "PuLP: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.