Lifetime-aware Design Methodology for Dynamic Partially Reconfigurable Systems

Siva Satyendra Sahoo, Tuan D. A. Nguyen, Bharadwaj Veeravalli
Department of Electrical and Computer Engineering
National University of Singapore
Singapore, Singapore
e-mail: satyendra@u.nus.edu, tuann@u.nus.edu, elebv@nus.edu.sg

Akash Kumar
Center for Advancing Electronics Dresden
Technische Universität Dresden
Dresden, Germany
e-mail: akash.kumar@tu-dresden.de

Abstract— Dynamic Partial Reconfiguration (DPR) in reconfigurable platforms can be used for the mitigation of aging-related permanent faults. We propose an application-specific system-level design methodology for determining the appropriate number of Partially Reconfigurable Regions and their compatibility with Partially Reconfigurable Modules for maximizing the system lifetime. Specifically, we propose a lifetime-aware scheduler that maximizes system MTTF. We use the scheduler along with an automated floorplanner for design space exploration at design-time to generate a heterogeneous PRR system. Our experiments show that the heterogeneous systems can offer up to $2x$ lifetime improvement over homogeneous ones.

I. Introduction

Embedded systems are used in a host of different applications – *Consumer Electronics, Telephony, In-Vehicle Infotainment, Medical Equipments, Automobiles, Military etc.* – with widely varying performance and dependability requirements. Reconfigurable systems, specifically FPGAs, have emerged as a key concept to cope with such diverse application requirements [1]. The state-of-the-art FPGAs now can incorporate sophisticated multi-processor system-on-chip with hundreds of general purpose processors and hardware accelerators. A major enabling factor behind such dense integration is the continuous technology scaling and architectural innovations in the semiconductor industry over the last four decades. However, the breakdown of Dennard Scaling has resulted in the increase of power density. The situation is worse with the reduced transistor sizes where the increased operating temperature due to higher power density leads to faster aging. Thus, the aging-related intermittent and permanent faults occur more frequently, leading to reduced system lifetime.

In reconfigurable platforms, *Dynamic Partial Reconfiguration* (DPR) allows replacing some hardware modules at runtime without affecting the rest of the system [2]. In addition to allowing multiplexing of different hardware accelerators (called Partially Reconfigurable Modules, PRMs) on compatible Partially Reconfigurable Regions (PRRs), DPR can be used to mitigate the permanent hardware faults at runtime by migrating the fault-affected PRM to another fault-free PRR. However, with such an approach, the system lifetime, depends on available PRR redundancy in the system architecture. Most of the research into DPR-based system design has been focused on homogeneous PRRs where each PRR can accommodate any PRM. While it provides the flexibility to configure each PRM to any PRR; given the limited FPGA resources, the redundancy of PRRs is limited by the largest PRM. Therefore, such an approach may not be appropriate for PRMs that have large resources variation.

The aforementioned issue opens a possibility of using heterogeneous systems to utilize the FPGA resources better. The PRRs in these systems now only host a subset of the original list of PRMs. The new freedom of assigning PRMs to PRRs leads to an interesting observation that it can be optimized to proactively improve the lifetime of the system. Aging-related fault rates are usually proportional to the number of execution cycles of the PRM on the PRR [3]. Therefore, an aging-aware approach to task-scheduling on reconfigurable platforms can increase the system lifetime by distributing the execution of stress-inducing PRMs across different PRRs. To this end, we propose a design-time methodology that analyzes the PRM-PRR mapping/scheduling space in both homogeneous and heterogeneous system w.r.t. system lifetime.
Contributions: Our contributions are listed below.

1. **A lifetime-aware scheduler** to improve the expected lifetime in DPR systems. We formulate a *Mixed Integer Linear Programming* (MILP) problem that incorporates the aging effect of the PRMs. The objective is to minimize the overall aging of each PRR and hence extend the overall lifetime of the system – for both homogeneous and heterogeneous systems.

2. **A resource-constraint-aware design methodology** to enable efficient usage of FPGA resources. The final system can be homogeneous or heterogeneous depending on the needs of the system designer.

We provide a brief overview of aging-mitigation techniques in FPGA-based systems, and state-of-the-art research in DPR-based systems in Section II. In Section III, we provide detailed description of our system model. Various stages of the proposed DPR system design methodology are detailed in Section IV. The experiment setup, and results for evaluating the proposed design methodology are described in Section V. Finally, we conclude the paper in Section VI with directions and scope for future work.

II. Background and Related Work

Lifetime Reliability: Solid-state devices tend to degrade with time and stress. Transistor scaling and higher

temperatures make the devices more susceptible and also accelerates the occurrence of aging-related faults. Recent surveys [3] suggest that the fault rates in processing elements (PEs) correlate with the number of cycles executed by the PE. Fault mechanisms that are activated by wear-out, resulting in the faults are – *Gate-oxide breakdown, Negative Bias Thermal Instability*, *Hot Carrier Injection* and *Electromigration*. Detailed description of these mechanisms can be found in [4, 5].

Various approaches have been proposed for mitigating the aging effects in FPGAs. In [5], the authors propose few phenomenon-specific methods such as selective alternate routing, load balancing, and leakage optimization to counter each failure mechanism. In [6], the authors propose three wear-levelling techniques to reduce electrical stress hotspots. The discussed methods provide generic reconfiguration solutions and do not consider any application specific requirements like deadlines and periodicity and the FPGA resource constraints. Such methods can be augmented to improve the performance of system-level design techniques discussed in the current article.

Dynamic Partial Reconfiguration: DPR enables different PRMs to share the same PRRs in time-multiplexed way leading to use of smaller devices, reduced power and more functionality. In addition to power and performance benefits, DPR offers a method for mitigating permanent faults. DPR-based lifetime extension methods can be broadly classified into two approaches– *Reactive*: involves relocating the PRMs from a faulty PRR to a functional region, and *Pro-active*: methods that reduce the electric stress on the PRRs, thereby reducing their wear-out. An aging-aware floorplanner along with a proactive aging-aware reconfiguration policy was proposed in [7] which aims to reduce the stress on PRRs by using the delay-based degradation estimates of previous execution cycles. In [8], the authors propose a methodology to periodically swap bitstreams of same PRM with placements that use different Configurable Logic Blocks (CLBs). In [9], a cross-layer aging-aware placement method for accelerators in FPGA-based runtime reconfigurable architectures is proposed. The described methodology involves module diversification, as proposed in [8], during synthesis and stress-aware placement at runtime to reduce wear-out. A stress-aware placement algorithm for DPR systems, that uses run-time aging estimation, is proposed in [10]. In [11], the authors propose a distributed architecture that uses DPR to mitigate soft-errors and permanent hardware faults in FPGA-based systems. The proposed methodology uses distributed control and same reactive recovery mechanism for all faults, thereby providing predictable recovery time.

Most of the proactive approaches to DPR-based lifetime extension assume homogeneous PRRs. Such an assumption simplifies the PRM to PRR mapping and reduces the complexity of designing mitigation techniques. However, if the PRMs have large variation in their resource requirements, each PRR area is dictated by the most resource consuming PRM. With limited reconfigurable resources, this can lead to reduced spatial redundancy and may result in reduced performance. Further, the most stress-inducing PRM dictates the aging of each PRR. A smaller PRM, that uses only a fraction of the available homogeneous PRR, but causes more electrical stress, can lead to faster aging and result in making the whole PRR unusable. Therefore, we propose a novel application-specific heterogeneous PRR-based partially reconfigurable system design methodology.

Parameter	Description
$prrID_r$	Serial number of PRR
$prrCLBs_r$	CLBs present in the PRR
$prrBRAMs_r$	BRAMs present in the PRR
$prrDSPs_r$	DSPs present in the PRR
$prrMTTF_r$	Estimated MTTF of the PRR
$prrPRMs_r$	List of PRMs supported by the PRR
$prrExTrace_r$	Schedule of task execution on the PRR

Fig. 1. PRR parameters

III. System Model

A. Architecture model

In this work, we perform Design Space Exploration (DSE) on various DPR systems with varying number of PRRs to find the system configuration which gives the best lifetime. In any DPR system, beside PRRs, there are other static modules that make up the whole functional system such as network-on-chip, reconfiguration/resource manager, reconfiguration module, etc. The system template must be flexible enough to automatically instantiate the corresponding static modules to support the varying number of PRRs. Therefore, we utilize the PR-HMPSoC template provided by Nguyen et al. [12]. All *Tiles* (or PEs) are connected to a network-on-chip for high bandwidth communication between them. The interactions with peripherals are done via the PLB bus. Each tile corresponds to one PRR.

We represent each PRR $\mathcal{R}_r$ with the parameters shown in Fig. 1, where r refers to the $prrID$ and varies between 1 and R, the total number of PRRs. Any parameter $< param >$ of r^{th} PRR is represented as $< param >_r$.

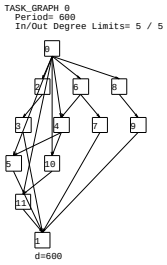
B. Application model

Mathematically, we model an application as a task-graph $G_{app} = (T_{app}, E_{app}, P_{app})$, where T_{app}, E_{app} and P_{app} represent the set of task nodes, the directed connectivity of the nodes representing task dependencies, and the periodicity of the application. Fig. 2a shows the application model represented as task-graphs. Fig. 2b describes the parameters used to represent each task node in the task-graph. The resource requirement and expected lifetime parameters represent the estimated resources used in the implementation of the accelerator or PRM and the estimated lifetime respectively. The deadline parameter $TaskD$ implements the real-time behavior of the application. For the rest of the article, each parameter $< param >$ of t^{th} task node will be represented as $< param >_t$, where t is the $TaskID$ and varies between 1 and $|T_{app}|$.

C. Lifetime Reliability model

We represent the expected lifetime of the system, $SysMTTF$, by the Mean Time To Failure (MTTF) of the system. The reliability model used is similar to that presented in [13]. Assuming a Weibull distribution of failures, the reliability of hardware resources and corresponding MTTF can be represented as shown in Equation 1. β, the shape parameter, can be used to represent the hardware fault profile and η, the scale parameter, represents the inverse of the aging effect of executing some PRM on the hardware.

$$R(t) = e^{-(t/\eta)^{\beta}} \; ; \; MTTF = \eta \times \Gamma(1 + 1/\beta) \tag{1}$$



(a) Task-graph

Parameter	Description
TaskID	Serial number of task
TaskType	Type of PRM used
StartT	Start time of task
ExecT	Expected execution time
EndT	End time of task
TaskCLBs	CLBs used for PRM implementation
TaskBRAMs	BRAMs used for PRM implementation
TaskDSPs	DSPs used for PRM implementation
TaskMTTF	Expected MTTF of the task PRM
TaskD	Any soft/hard deadline of the task

(b) Task node parameters

Fig. 2. Application model

Considering the temporal variation in aging effect, caused by the time multiplexing of different PRMs on a PRR, the effective scale parameter η_{eff} over a time interval t can be obtained as shown in Equation 2. η_i and $MTTF_i$ represent the aging effect due to execution of i^{th} PRM on a PRR.

$$\eta_{eff} = \frac{\sum \Delta t_i}{\sum \frac{\Delta t_i}{\eta_i}} \ , \ t = \sum \Delta t_i \ , \ \eta_i = \frac{MTTF_i}{\Gamma(1 \ + \ 1/\beta)} \tag{2}$$

Considering P_{app} as the representative time interval, the effective MTTF of a PRR, $prrMTTF$, and $SysMTTF$ can be obtained as shown in Equation 3. M refers to the number of tasks mapped on the PRR. We do not include the effect of process variations in our model. Hence, the shape parameter β remains constant.

$$\begin{aligned} prrMTTF_r &= \frac{P_{app}}{\sum_{i=1}^{M} \frac{ExecT_i}{TaskMTTF_i}} \\ SysMTTF &= \min_{all\ PRRs}(prrMTTF_r) \end{aligned} \tag{3}$$

IV. Heterogeneous lifetime-aware DPR System Design

The overall flow of the proposed design methodology is shown in Fig. 3. We use the application task-graph to generate a feasible execution trace that is used by the MILP to determine the appropriate PRR to map to each task node. The PRMs' MTTF values, determined during PRM characterization stage, are used to constraint the MILP solver to optimize for system lifetime. DPR resource estimation stage involves estimating the available resources for DPR after generating the static components of the system. This information, along with PRMs' resoure requirement estimates obtained during PRM characterization, is used to constraint the MILP from generating an infeasible solution. Floorplanning stage involves verifying the feasibility of the heterogeneous mapping information generated by the solver and getting a feasible system design. We perform design space exploration (DSE) to find the maximum number of PRRs that maximizes the $SysMTTF$ and is still feasible with the limited resources of the FPGA. We obtain this by incrementing the number of PRRs in the system, solving the MILP and using PRFloor to find the feasibility of the design. The DSE is completed when the MILP solver fails to find a valid mapping of tasks to PRRs. The design with maximum $SysMTTF$ having the lowest number of PRRs is selected as the best design.

A. DPR Resource estimation

The DPR system consists of static components in addition to the DPR resources. The amount of FPGA resources dedicated to static components varies with the number of PRRs implemented in the system. We use the automatic floorplanner, PRFLoor [14] to estimate the remaining resources in terms of number of CLBs, Block RAMs (BRAMs), and Digital Signal Processing blocks (DSPs), that can be utilized for creating heterogeneous PRRs. We denote the remaining DPR resource quantities by $remCLBs$, $remBRAMs$, and $remDSPs$.

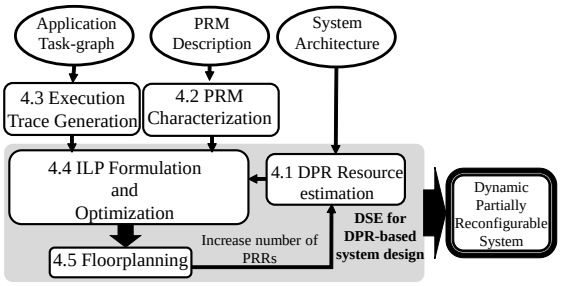


Fig. 3. Overall Design methodology

B. PRM characterization

Each PRM is characterized to determine their resource requirements and their aging effect. Each task node of the application task-graph involves the execution of any one of the PRMs. Hence, the resource requirements for each task is obtained from the synthesis of the respective PRMs. Similarly, the power estimation from the synthesized netlist is used to generate the expected junction temperature. Out of the four dominant wear-out methods, we model EM related wear-out failures for our current work. However, any other effects can be easily incorporated either standalone or using Sum-of-Failure Rate (SOFR) model for any combination of the failure mechanisms. The estimated aging effect at temperature T_i can be obtained based on the relation shown in Equation 4. A_0 is a constant determined by the physical interconnect, E_a is the activation energy, J (and J_{crit}) refer to the current density (and critical current density), n is an empirically determined constant, and K is the Boltzmann constant.

$$\eta(T_i) \ = \ \frac{A_0(J - J_{crit})^{-n} e^{E_a/(KT_i)}}{\Gamma(1 + 1/\beta)} \tag{4}$$

C. Execution Trace Generation

A greedy algorithm is used to generate a list, $ExTrace$, of tasks from the application task-graph. A two-stage approach is used for this purpose. In the first stage, we update the $StartT_t$ and $EndT_t$ of each task t, with the assumption of infinite parallel resources available in the system. This provides us with the best case $StartT_t$ of each task. In the second stage, we create a linear array of tasks $ExTrace$ by using a greedy approach. We parse through all tasks in the set T_{app} that are not already in $ExTrace$. A list of all feasible options for the next entry into $ExTrace$, i.e. tasks whose parents nodes are already in the trace, is generated. From this list we choose the task with the least $ExecT_t$ as the next entry in $ExTrace$. This linear list of tasks, prevents the solver from evaluating infeasible sequences of task executions.

D. MILP Formulation

The MILP problem formulation and its solution are used to determine the Task to PRR mapping. We formulate the problem as finding appropriate entries for a binary-valued matrix, $MapMatrix$. The columns correspond to the tasks in the $Exec\_Trace$, and the rows correspond to the available PRRs in the system. Hence the

$MapMatrix$ is of size $R \times |T_{app}|$. The matrix entries $rt_{(r,t)}$ denote whether task t is executed ($rt_{(r,t)} = 1$) on PRR r or not ($rt_{(r,t)} = 0$). The different constraints and objective function of the MILP are described below.

Deadline constraints: For every task t with a deadline, a constraint, $StartT_t + ExecT_t \leq TaskD_t$, is added to the problem.

Dependability constraints: For every task t a constraint for every parent task node p a constraint: $StartT_t \geq StartT_p + ExecT_p$, is used in the problem formulation.

Task start time constraints: For a task-PRR pair (t, r), we introduce a new variable $StartT_{(t,r)}$ that signifies feasible start time of task t when mapped to PRR r. Since the execution trace signifies a sequence of task execution, the first relation in Equation 5 signifies the feasible values of $StartT_{(t,r)}$. The second relation in the equation provides the overall equivalent start time of the task.

$$\begin{gathered} For\ every\ task,\ t,\ for\ every\ PRR\ r: \\ StartT_{(t,r)} \geq \sum_{i=1}^{t-1} (StartT_{(i,r)} + ExecT_i) \times rt_{(r,i)} \\ where,\ i\ is\ any\ task\ that\ is\ before\ t\ in\ Exec\_Trace \\ For\ each\ task\ t,\quad StartT_t = \sum_{r=1}^{R} StartT_{(t,r)} \times rt_{(r,t)} \end{gathered} \tag{5}$$

Lifetime Constraints: For each PRR, we use a variable, $InvMTTF_r = \sum_{t=1}^{T} \frac{ExecT_t \times rt_{(r,t)}}{TaskMTTF_t}$, to denote the net aging effect on a PRR in each cycle. Please note from Equation 3, maximizing for $SysMTTf$ is equivalent to minimizing the maximum of $InvMTTF_r$ across all PRRs. Therefore we use a variable $SysInvMTTF$ to denote the maximum $InvMTTF_r$.

Resource Constraints: For any task to be executed on a PRR, the PRR must have sufficient resources. Further, the sum of all resources in all PRRs must be less than the remaining resources for DPR. Equation 6 shows the resource constraints used in the MILP formulation.

$$\begin{gathered} For\ every\ task,\ t,\ for\ every\ PRR\ r: \\ prrCLBs_r \geq TaskCLBs_t \times rt_{(r,t)} \\ prrDSPs_r \geq TaskDSPs_t \times rt_{(r,t)} \\ prrBRAMs_r \geq TaskBRAMs_t \times rt_{(r,t)} \\ Overall\ Resources: \\ remCLBs \geq \sum_{r=1}^{R} prrCLBs_r;\ remDSPs \geq \sum_{r=1}^{R} prrDSPs_r \\ remBRAMs \geq \sum_{r=1}^{R} prrBRAMs_r; \end{gathered} \tag{6}$$

Objective Function: To compare the performance of our lifetime-aware scheduler, we run two optimization modes:

In mode 1 we maximize the system life time with the objective function: $Minimize\ SysInvMTTF$.

In mode 2 we minimize the makespan of the application by minimizing the start time of the last task. The corresponding objective function: $Minimize\ StartT_T$.

E. Floorplanning

The automatic floorplanning tool PRFloor [14] is used to determine the feasibility of the mapping generated by the MILP solver.

V. Experiments and Results

A. Experiment Setup

All our experiments are run on a computer with two CPUs – Intel^{TM} Xeon^{TM} E5-2609 v2 @ 2.50GHz (each CPU is quad-core) and 32 GB of memory. The operating system is Ubuntu 14.04 LTS 64-bit. Even though our method is made general enough for all kinds of Xilinx FPGA, the one we are experimenting with is Virtex-6 XC6VLX240T. Gurobi Solver is used to solve our scheduler with parameter *Presolve=2*.

B. IP Pool

In this work, we collected 50 real-world hardware accelerators (PRMs) from CHStone benchmark, Opencores, EPFL benchmark and Xilinx XPS IP core library. The MTTF for different PRMs was obtained using the relation shown in Equation 4. The scaling parameter was used to obtain an MTTF of 75 years at 25℃ and the PRMs' MTTF values were truncated and scaled to obtain a range from 2-10 years. Please note that our contributions do not include the PRM characterization. We used the generated data to get realistic estimates about the performance of our proposed lifetime-aware scheduler and heterogeneous system design tool. More accurate estimates of PRMs' MTTF can be plugged directly to our proposed flow to perform more accurate analysis.

C. Experiments

Experiments for performance evaluation involved using application task-graphs with the number of tasks varying from 5 to 50, in increments of 5. The task-graphs were generated using Task Graphs For Free (TGFF) tool [15]. Task-graphs with different levels of branching and depth were used to compare the performance. We use terms *Fat* and *Slim* to describe applications that demand higher parallel resources and longer serial chains respectively. Further, PRM sets with different range of resource requirements were used for evaluation. In order to estimate the performance of the scheduler and heterogeneous PRR-based systems simultaneously, we performed experiments in 4 modes. $Mode\_1\_1$ and $Mode\_1\_2$ refer to maximization of system lifetime and minimization of application makespan respectively in a heterogeneous PRR system. Similarly $Mode\_2\_1$ and $Mode\_2\_2$ refer to optimization of system lifetime and makespan respectively in a homogeneous PRR system. We limit the number of available PRRs to 15, as the homogeneous system design fails for almost all cases beyond that.

D. Performance Results

We quantify the performance of our proposed methodology in terms of increased system MTTF. Figs. 4, 6, 7 and 9 show the result for all four modes for different *scenarios* – applications with different number of tasks, set of PRMs with different resource distribution (small /large PRMs) and type of applications (*Fat* / *Slim*). The bars represent the maximum system MTTF in years, among all feasible values of number of PRRs, R, for each mode. The minimum value of R at which we obtain the maximum MTTF for a mode are shown as labeled markers in the figure.

Figs. 5, 8 and 10 provide detailed results for a representative application (with 25 tasks) under different scenarios. They show the variation in system MTTF with increasing number of PRRs in the system. The performance of the proposed methodology in different scenarios is discussed under two subsections below.

D.1 Lifetime Reliability-aware scheduling

The system's MTTF ($SysMTTF$) obtained using the lifetime-aware scheduler ($Mode\_*\_1$), shows considerable

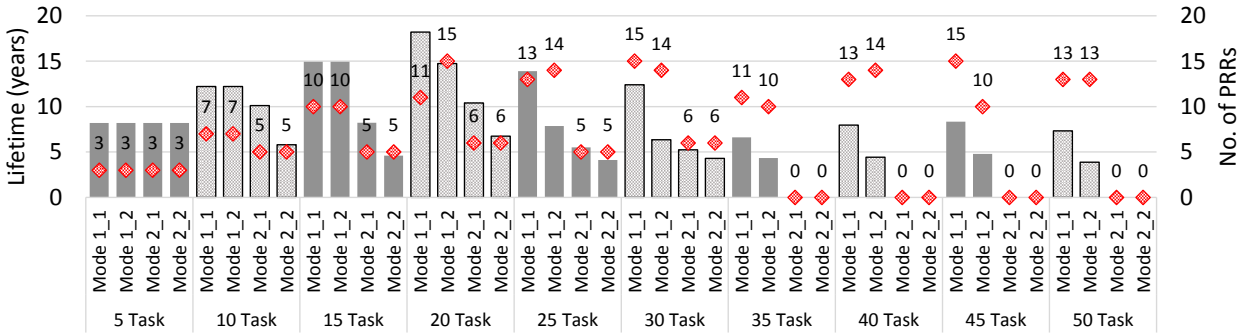


Fig. 4. *SysMTTF* in *Fat* applications with large PRMs

increase over a makespan-optimization approach (*Mode_* _2*) for almost every scenario.

In applications with very few tasks (e.g. 10 tasks in Fig. 7, 9), both scheduling modes exhibit similar performance. In smaller task-graphs, the scope for improvement of lifetime is limited as there are insufficient tasks to exploit the parallelism. In our current work, we are yet to explore the effect of redundant PRRs, created from the spare DPR resources, and is left for future research. We only consider applications with sufficient tasks to exploit all available parallelism in the system. So, the results for minimum makespan are similar to that of aging-aware optimization for smaller task-graphs.

In Fig. 5 (with 4 PRRs), it can be observed that the aging-aware scheduler, unlike makespan optimization, could not find a feasible result. This is expected as the deadline constraints imposed in the aging-aware MILP are not used in the makespan optimization. Similar behavior can also be observed in Fig. 7 for 50 tasks. Here, the maximum achievable parallelism in both homogeneous and heterogeneous PRR types is insufficient for meeting the application deadline.

Further, for all scenarios, the quality of results of our aging-aware scheduler increases with increasing number of PRRs in the system. The scheduler uses increasingly available parallel resources to spread the electrical stress spatially, thereby reducing stress hotspots, resulting in improved lifetime. In some cases, the scheduler performance flattens beyond a certain point. In Fig. 5 there is no improvements beyond 13 PRRs, as the resource redistribution to create additional heterogeneous PRRs, does not result in extra PRRs for the more stress-inducing PRMs. Similarly, as shown in Fig. 8, performance benefits with the aging-aware scheduler do not improve beyond 11 PRRs for neither homogeneous nor heterogeneous PRRs as the aging is dominated by one single PRM mapped to one PRR.

Overall, it can be concluded that the aging-aware scheduler results in considerable system lifetime improvements

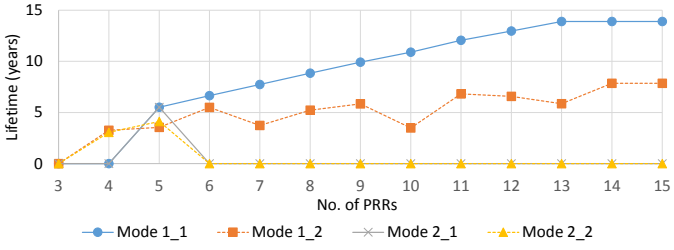


Fig. 5. Variation of *SysMTTF* with number of PRRs for a *Fat* task-graph with 25 tasks and large PRMs

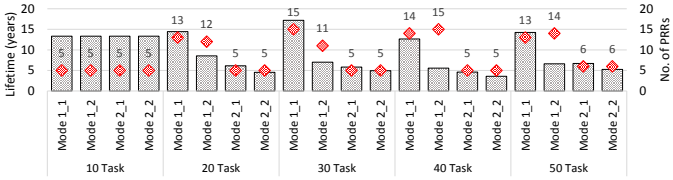


Fig. 6. *SysMTTF* in *Slim* applications with large PRMs

TABLE I
SysMTTF IMPROVEMENTS OF HETEROGENEOUS VS. HOMOGENEOUS SYSTEMS

Scenarios	T= 5	T=10	T=15	T=20	T=25	T=30	T=35	T=40	T=45	T=50
Fat, Large	0.00	0.21	0.82	0.75	1.52	1.37	**6.62**	**7.96**	**8.33**	**7.33**
Slim, Large	0.00	0.00	1.24	1.36	1.42	1.95	**9.57**	1.76	**13.16**	1.13
Fat, Small	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.06	0.06	**0.00**
Slim, Small	0.00	0.00	0.00	0.00	0.05	0.11	0.00	0.00	0.08	0.00

over our baseline makespan optimization scheduler. irrespective of the heterogeneity of PRRs.

D.2 Lifetime-aware DPR-based System Design

The aging-aware scheduler was used in conjunction with the DPR resource constraints of the system to generate the PRM to PRR mappings that maximize the system MTTF. Table I summarizes the improvements of a heterogeneous system over a homogeneous one for different scenarios. The entries in bold-face signify the inability to find a feasible homogeneous design for the application.

For scenarios that use large PRMs, we observe significant improvements by using a heterogeneous system. Since homogeneous PRRs need to be compatible with all PRMs, the size of each PRR is significantly increased with larger PRMs. Therefore, the resource constraints of the FPGA limit the number of maximum parallelism with such large PRRs. The reduced parallelism may be insufficient for meeting deadlines for real-time applications. A heterogeneous system, on the other hand, allows redistribution of resources to create more PRRs for PRMs that need more parallel modules. As shown in Fig. 4, the homogeneous system design fails for *Fat* applications with 35 or more tasks.

Even in scenarios where both types of systems are feasible, heterogeneity allows allocation of more PRRs that are compatible with the more stress-inducing PRMs, thereby increasing the system MTTF. As seen in Fig. 5, the homogeneous system can support only 5 PRRs compared to a heterogeneous system with up to 15 PRRs. The additional PRRs were used to accomodate the more stress-inducing PRMs, and show improvements in system MTTF for up to 13 PRRs. The flattening of performance beyond that

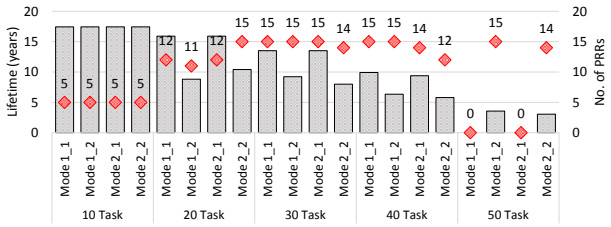


Fig. 7. *SysMTTF* in *Fat* applications with small PRMs

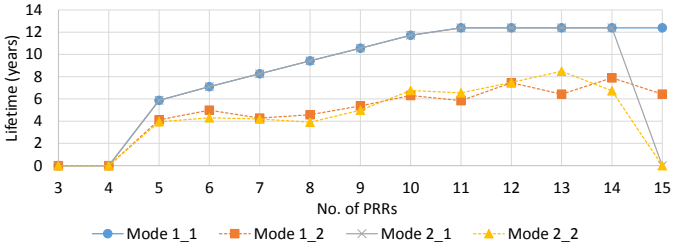


Fig. 8. Variation of *SysMTTF* with number of PRRs for a *Fat* task-graph with 25 tasks and small PRMs

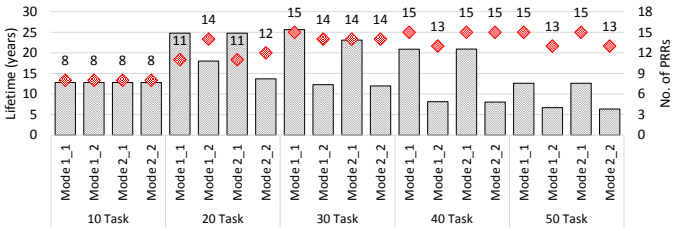


Fig. 9. *SysMTTF* in *Slim* applications with small PRMs

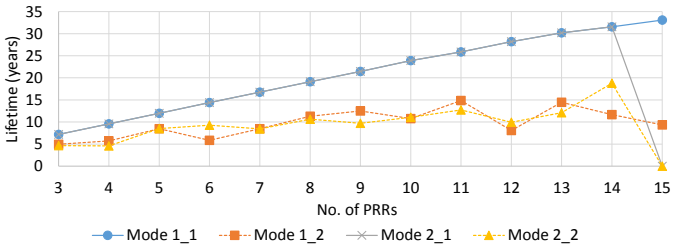


Fig. 10. Variation of *SysMTTF* with number of PRRs for a *Slim* task-graph with 25 tasks and small PRMs

was explained in the previous sub-section.

For *Slim* applications with large PRMs, the heterogeneous approach ensures feasibility for all applications. The homogeneous system design may lead to infeasible results based on the parallelism requirements of the application for meeting deadlines.

In scenarios with small PRMs, the resource constraints do not play a significant role. Therefore, the PRR count can be increased considerably for homogeneous systems to achieve the required level of parallelism. Hence, heterogeneous systems do not show any significant improvements in lifetime or feasibility over homogeneous ones. As shown in Figs. 7 and 9, both the systems perform almost similarly for all scenarios. Both approaches fail to generate feasible designs for a *Fat* application with 50 tasks, as the resources are insufficient for sufficient parallelism. As seen in Figs. 8 and 10, with increasing number of PRRs, the performance of the homogeneous system matches that of the heterogeneous system till there are sufficient resources for creating more homogeneous PRRs. Therefore, for smaller PRMs, the proposed design methodology allocates sufficient resources to each PRR to create a homogeneous system.

VI. Conclusion

In this paper, we propose a novel lifetime-aware proactive methodology DPR-based system design. Our approach analyzes different aspects of designing such systems – the aging effects of PRMs, the dependencies between them from the application task graph, the PR-based system architecture and FPGA resource constraints. These information are used to build an MILP mathematical model to make sure that all of them are globally considered. The PRRs of the resulting system are heterogeneous; therefore the FPGA resources are used more efficiently to optimize the lifetime.
Currently, we are working on exploring the possibility of having more heterogeneous PRRs as redundancy resources. These PRRs can be utilized at runtime to further improve the system lifetime. A lifetime-aware runtime scheduler will also be developed to efficiently monitor and determine the mapping of PRMs to PRRs.

Acknowledgments

This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden" (cfaed) at the Technische Universität Dresden and the HiPEAC4 Network of Excellence.

References

[1] M. Glesner, H. Hinkelmann, T. Hollstein, L. S. Indrusiak, T. Murgan, A. M. Obeid, M. Petrov, T. Pionteck, and P. Zipf. *Reconfigurable Embedded Systems: An Application-Oriented Perspective on Architectures and Design Techniques*. 2005.

[2] D. Koch. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Springer Science & Business Media, 2012.

[3] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer pcs. In *Proceedings of the sixth conference on Computer systems*, 2011.

[4] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung. Degradation in FPGAs: Measurement and Modelling. In *Proceedings of FPGA*, 2010.

[5] 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*, 2008.

[6] E. Stott and P. Y. K. Cheung. Improving FPGA reliability with wear-levelling. In *Proceedings of FPL*, 2011.

[7] Z. Ghaderi and E. Bozorgzadeh. Aging-aware high-level physical planning for reconfigurable systems. In *Proceedings of ASP-DAC*, 2016.

[8] 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 *Proceedings of ITC*, 2013.

[9] 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 *Proceedings of ICCAD*, 2015.

[10] J. Angermeier, D. Ziener, M. Glaß, and J. Teich. Stress-aware module placement on reconfigurable devices. In *Proceedings of FPL*, 2011.

[11] V. Dumitriu, L. Kirischian, and V. Kirischian. Run-time recovery mechanism for transient and permanent hardware faults based on distributed, self-organized dynamic partially reconfigurable systems. *IEEE Transactions on Computers*, 2016.

[12] T. D. A. Nguyen and A. Kumar. PR-HMPSoC: A versatile partially reconfigurable heterogeneous Multiprocessor System-on-Chip for dynamic FPGA-based embedded systems. In *Proceedings of FPL*, 2014.

[13] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang. System-level reliability modeling for MPSoCs. In *Proceedings of CODESS*, 2010.

[14] T. D. Nguyen and A. Kumar. PRFloor: An Automatic Floorplanner for Partially Reconfigurable FPGA Systems. In *Proceedings of FPGA*, 2016.

[15] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, 1998.