Integration, the VLSI Journal xxx (xxxx) xxx

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Multi-objective design space exploration for system partitioning of FPGA-based Dynamic Partially Reconfigurable Systems

S.S. Sahoo [a,*], T.D.A. Nguyen [b], B. Veeravalli [a], A. Kumar [b]

[a] Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583, Singapore
[b] Center for Advancing Electronics Dresden, Technische Universität Dresden, Dresden 01187, Germany

ARTICLE INFO

Keywords:
Dynamic partial reconfiguration
Field programmable gate arrays
Lifetime-aware scheduling
Task-graphs
Reliability
Heterogeneous systems
Real-time systems

ABSTRACT

Dynamic Partial Reconfiguration (DPR) enables resource sharing in FPGA-based systems. It can also be used for the mitigation of aging-related permanent faults by increasing the number of redundant Partially Reconfigurable Regions (PRRs). Normally, these PRRs are able to host any of the Partially Reconfigurable Modules (PRMs), or tasks, at one particular instance. This kind of system is called homogeneous. However, the FPGA resource constraints limit the amount of *homogeneous* redundancy that can be used and hence affect the lifetime of the system. This issue can be addressed by utilizing the heterogeneous approach where each PRR now only hosts a subset of the tasks. Further, the deadlines of the applications must also be taken care of in the design phase to decide the mapping and scheduling of tasks to PRRs. To this end, we propose an application-specific multi-objective system-level design methodology to determine the appropriate number of PRRs and the mapping and scheduling of tasks to the PRRs. Specifically, we propose a lifetime-aware scheduling method that maximizes the system's mean time to failure (MTTF) with different tolerances in the makespan specification of an application. We use the scheduler along with an automated floorplanner for design space exploration at design-time to generate a feasible heterogeneous PRR-based system. Our experiments show that the heterogeneous systems can offer more than 2*x* lifetime improvement over homogeneous ones. It also offers better scaling with increased tolerance in makespan specification.

1. Introduction

Embedded systems are used in a host of different applications – *Consumer Electronics, Telephony, In-Vehicle Infotainment, Medical Equipment, Automobiles, Military etc.* – with widely varying performance and dependability requirements. Reconfigurable systems, specifically FPGA, have emerged as a key concept to cope with such diverse application requirements [1]. Recent advancements in High Level Synthesis and related design space exploration techniques have resulted in improved performance estimation and programmability of a wide variety of workloads on FPGAs [2,3]. 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 even worse with the reduced transistor size where the increased operating temperature due to the 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 [4]. In addition to the ability to multiplex different hardware accelerators (called Partially Reconfigurable Modules, PRMs) on compatible Partially Reconfigurable Regions (PRR), DPR can be used to mitigate the permanent hardware faults at runtime by migrating the fault-effected 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 and the size of the FPGA. Therefore,

* Corresponding author.
E-mail addresses: satyendra@u.nus.edu (S.S. Sahoo), tuan_duy_anh.nguyen1@tu-dresden.de (T.D.A. Nguyen), elebv@nus.edu.sg (B. Veeravalli), akash.kumar@tu-dresden.de (A. Kumar).

https://doi.org/10.1016/j.vlsi.2018.10.006
Received 15 May 2018; Received in revised form 26 August 2018; Accepted 26 October 2018
Available online xxxx
0167-9260/© 2018 Elsevier B.V. All rights reserved.

such an approach might 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 heterogeneity of the PRRs is in terms of their compatibility to different PRMs. The PRRs in such heterogeneous systems now only host a subset of the original list of PRMs. The incompatibility of a specific PRR to some PRMs is due to the insufficient resources allocated to the PRR. Homogeneous PRRs, on the other hand, are compatible with all PRMs in the list. This 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 [5]. 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 analyses the PRM-PRR mapping/scheduling space in both homogeneous and heterogeneous system for improving system lifetime.

Contributions: Our contributions are listed below.

1. **A lifetime-aware scheduler** to improve the expected lifetime in DPR systems. In Ref. [6], we proposed a *Mixed Integer Linear Programming* (MILP)-based problem formulation and optimization methodology to incorporate the aging effect of individual PRMs to minimize the overall aging of each PRR and hence extend the overall lifetime of the system – for both homogeneous and heterogeneous systems. In our current work, we propose a *Genetic Algorithm* (GA)-based optimization approach that can be used for larger application sets and for multi-objective optimization.
2. **A resource-constraint-aware system partitioning methodology** to make sure that the final system can make use of the available FPGA resources more efficiently. The final system can be homogeneous or heterogeneous depending on the needs of the system designer. The design methodology is integrated with the state-of-the-art PR-floorplanner to verify the feasibility of implementing the system on the FPGA. The methodology was implemented using both the MILP-based and GA-based approach.
3. **A multi-objective optimization** case-study was investigated using the proposed approach. Specifically, we compared the performance scaling of the homogeneous PRR (*HomPRR*)-based and heterogeneous PRR (*HetPRR*)-based systems with increasing tolerances to the makespan requirements of an application.

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 2. In Section 3, we provide a detailed description of our system model. Various stages of the proposed DPR system design methodology is detailed in Section 4. The experiment setup and results for evaluating the proposed design methodology are described in Section 5. Finally, we conclude the paper in Section 6 with directions and scope for future work.

2. 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 accelerates the occurrence of aging-related faults. Recent surveys suggest that the fault rates in processing elements (PEs) correlate with the number of cycles executed by the PE [5]. 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*. A detailed description of these mechanisms can be found in Refs. [7,8].

Various approaches have been proposed for mitigating the aging effects in FPGAs. In Ref. [8], the authors propose a few phenomenon-specific methods such as selective alternate routing, load balancing, and leakage optimization to counter each failure mechanism. In Ref. [9], the authors propose three wear-levelling techniques to reduce electrical stress hot-spots. The discussed methods provide generic reconfiguration solutions and do not consider any application-specific requirements like deadlines and periodicity and FPGA resource constraints. Such methods can be augmented to improve the performance of system-level design techniques discussed in this article.

Dynamic Partial Reconfiguration: DPR enables different PRMs to share the same PRRs in a time-multiplexed way leading to the use of smaller devices with potentially reduced power consumption while maintaining the same 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* and *Pro-active*. The former approach involves relocating the PRMs from a faulty PRR to another functional one. The *Pro-active* methods, on the other hand, aim to reduce the electric stress on the PRRs by distributing it across multiple PRRs, thereby reducing their wear-out. In our current work, we focus on improving the system's mean time to failure with the *pro-active* approach. An aging-aware floorplanner along with a proactive aging-aware reconfiguration policy was proposed in Ref. [10] which aims to reduce the stress on PRRs by using the delay-based degradation estimates of previous execution cycles. In Ref. [11], the authors propose a methodology to periodically swap multiple bitstreams of the same PRM in which Configurable Logic Blocks (CLB) placements are different. In Ref. [12], 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 Ref. [11], 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 Ref. [13]. In Ref. [14], 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.

The DSE for application-specific, DPR-based FPGA system design involves multiple design challenges, namely – *Task-to-PRR scheduling*, *PRR resource allocation*, and *multi-objective optimization*. Considering all these aspects together can lead to an explosion in the design space. As shown in Table 1, most state-of-the-art approaches do not consider all aspects of the problem. For example, 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 a 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 potentially unusable PRR. Similarly, while MILP-based methods may be sufficient for single-objective optimization, such methods do not scale with an increasing number of objectives and the problem size. Therefore, we propose a novel application-specific heterogeneous PRR-based partially reconfigurable system design methodology that uses both analytical and stochastic search-based optimization methods.

3. System model

3.1. Architecture model

In this work, we perform Design Space Exploration (DSE) on various DPR systems with a varying number of PRRs to find the system configuration which gives the best lifetime. In any DPR system, besides 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

Table 1
Comparing related work.

Comparison Criteria	Optimization Objectives		PRR Heterogeneity		Design Activities	
Related Work	Single	Multiple	Homogeneous	Heterogeneous	Scheduling	System Partitioning
Current Work	Aging mitigation, Performance	✓	✓	✓	✓	✓
[10]	Aging mitigation	✗	✓	✗	✗	✗
[11]	Aging mitigation	✗	✓	✗	✓	✗
[12]	Aging mitigation, Performance	✓	✓	✗	✓	✗
[13]	Aging mitigation	✗	✓	✗	✗	✗
[14]	Fault tolerance, Performance	✓	✓	✗	✓	✗

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. [15]. Overview of PR-HMPSoC is shown in Fig. 1a. 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 R_r with the parameters shown in Fig. 1b, where *r* is the *prrID* and varies between 1 and *R*, where *R* is the number of PRRs. Any parameter $<param>$ of *r*th PRR is represented as $<param>_r$.

3.2. Application model

Mathematically, we model an application as $G_{app} = (T_{app}, E_{app}, P_{app})$, where T_{app}, E_{app} and P_{app} correspondingly 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 behaviour of the application. For the rest of the article, any parameter $<param>$ of *t*th task node will be represented as $<param>_t$, where *t* is the *TaskID* and varies between 1 and $|T_{app}|$.

3.3. 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 Ref. [16]. 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.

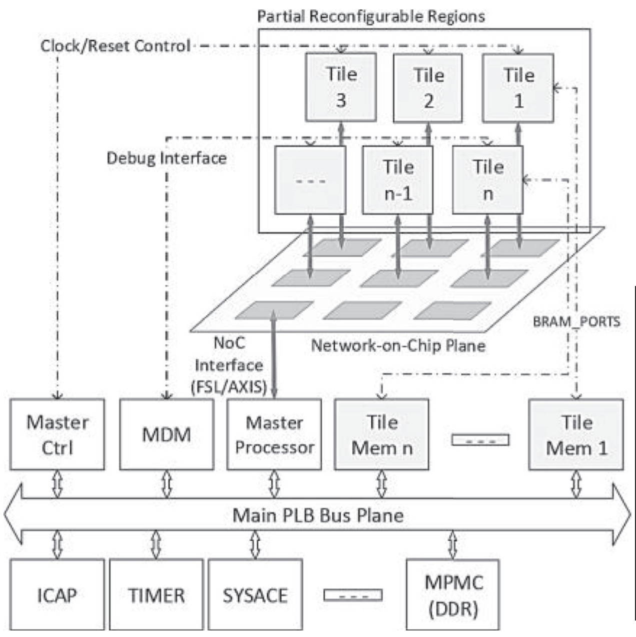
$$R(t) = e^{-\left(\frac{t}{\eta}\right)^{\beta}} ; MTTF = \eta \times \Gamma(1 + 1/\beta) \tag{1}$$

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 the execution of *i*th PRM on a PRR.

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

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.

$$prrMTTF_r = \frac{P_{app}}{\sum_{i=1}^{M} \frac{ExecT_i}{TaskMTTF_i}} \; ; SysMTTF = \min_{all\ PRRs} prrMTTF_r \tag{3}$$

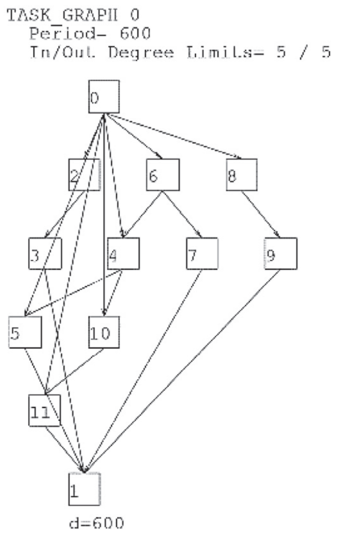


(a) PR-HMPSoC architecture [15]

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

(b) PRR parameters

Fig. 1. Architecture model.



(a) Application 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.

4. Lifetime-aware system partitioning methodology

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 of that application. This trace is then used during the design space exploration (DSE) to enforce precedence constraints among tasks to determine the appropriate PRR to map to each task node. The PRMs' MTTF values, determined during PRM characterization stage, are used to constrain the *optimizer* to solve for maximum expected system MTTF. DPR resource estimation stage involves estimating the available resources for DPR after generating the static components of the system. This information, along with PRMs' resource requirement estimates obtained during PRM characterization, is used to implement the resource constraints of the system. Floorplanning stage involves verifying the feasibility of the heterogeneous mapping information generated by the optimizer and getting a feasible system design. We perform DSE to find the maximum number of PRRs that maximizes the *SysMTTF* and is still feasible within the limited resources of the FPGA. We obtain this by incrementing the number of PRRs in the system, solving the appropriate optimization problem and using PRFloor to find the feasibility of the design. The DSE is completed when the optimizer fails to find a valid mapping of tasks to PRRs. The results from all the problems solved with varying number of PRRs are collected to generate the result – a best possible *SysMTTF* or a Pareto front for multiple objectives.

4.1. 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 [17] 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*.

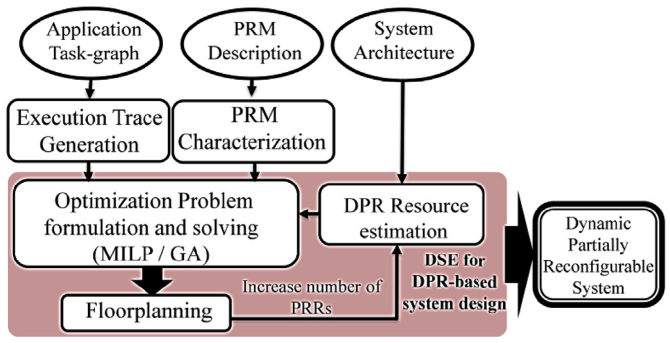

Application Task-graph
PRM Description
System Architecture
Execution Trace Generation
PRM Characterization
Optimization Problem formulation and solving (MILP / GA)
DPR Resource estimation
Floorplanning
Increase number of PRRs
DSE for DPR-based system design
Dynamic Partially Reconfigurable System

Fig. 3. Methodology for design of Lifetime-aware Dynamic Partially Reconfigurable Systems.

4.2. PRM characterization

Each PRM is characterized by determining 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 the EM-related wear-out failures for our current work. However, any other effects can be easily incorporated either individually or using the 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.

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

4.3. Execution trace generation

A greedy algorithm is used to generate a list, *ExecTrace*, 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 *Exec trace* by using a greedy approach. We parse through all tasks in the set T_{app} that are not already in *Exec trace*. A list of all feasible options for the next entry into *ExecTrace*, i.e. tasks whose parent 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 *ExecTrace*. This linear list of tasks prevents the solver from evaluating infeasible sequences of task executions.

4.4. Optimization problem formulation

4.4.1. MILP-based optimization

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, *Mapping Matrix*, shown in Fig. 4a. The columns correspond to the tasks in the *Exec Trace*, and the rows correspond to the available PRRs in the system. Hence the *Mapping Matrix* 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 the 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.

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

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. It can be inferred 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.

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

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

Mode 1: We maximize the system lifetime with the objective function: *Minimize SysInvMTTF*.

Mode 2: We minimize the makespan of the application by minimizing the start time of the last task. The corresponding objective function: *Minimize* $StartT_T$.

4.4.2. GA-based optimization

The optimized PRM–PRR scheduling is obtained by executing the PRM for each task on an *appropriate* PRR at the *right* time. Different PRM-PRR schedules for all tasks of an application will result in varying resource requirements for each PRR (and the system), *prrMTTF* (and *SysMTTF*) and makespan (*SysMS*) for the application. We propose a Genetic Algorithm (GA)-based multi-objective optimization method for both *SystemMTTF* and *SysMS*. Each of the ordered sequences shown in Fig. 4b denotes an individual of a *population*. Each *individual* contains T sub-sequences, one for each task in the application task-graph. Therefore, each individual encodes one schedule for all tasks in an application. Each sub-sequence, $s_{(i,q)}$, of i_{th} individual in the population can be represented by the tuple $(t_{(i,q)}, r_{(i,q)})$, the task index and the PRR index respectively. The task execution schedule on each PRR, θ_r, is implicitly determined by the order of task indexes in the sequence (for each PRR). Each individual in the population, S_i, denotes a point on the design space and can be quantified by the tuple of metrics:($SysMTTF_i$, $SysMS_i$, $SysCLBs_i$, $SysBRAMs_i$, $SysDSPs_i$), where:

$$\begin{aligned}
&SysCLBs_i \geq \sum_{r=1}^{R} prrCLBs_r;\\
&SysDSPs_i \geq \sum_{r=1}^{R} prrDSPs_r\\
&SysBRAMs_i \geq \sum_{r=1}^{R} prrBRAMs_r
\end{aligned} \tag{7}$$

The optimization problem can then be defined as searching for an optimal sequence S_{opt} that maximizes the reward function:

$$Rwd_i = Wt_{MTTF} \times SysMTTF_i - Wt_{MS} \times SysMS_i + Cost_{IN(i)} \tag{8}$$

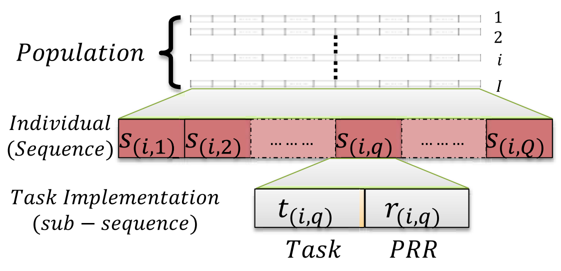
In Equation (8), Wt_{MTTF} and Wt_{MS} denote the user specified importance to MTTF and makespan. The cost $Cost_{IN}$ denotes the cost of infeasibility w.r.t. the FPGA resource and the makespan specification ($Spec_{MS}$). It is expressed as:

$$Cost_{IN(i)} = \begin{cases} -100000,\ if\ SysCLBs_i > remCLBs \\ -100000,\ if\ SysDSPs_i > remDSPs \\ -100000,\ if\ SysBRAMs_i > remBRAMs \\ -100000,\ if\ SysMS_i > Spec_{MS} \\ 0,\ otherwise \end{cases} \tag{9}$$

GA-based optimization involves traversing the design space by generating new (and better) individuals for each subsequent generation. We use the following operations for generating new individuals for the next *generation*'s population.

$Exec\_Trace$	T_1	T_2	....	T_t	....	T_T
$PRRs$						
R_1	$rt_{(1,1)}$	$rt_{(1,2)}$	....	$rt_{(1,t)}$	....	$rt_{(1,T)}$
R_2	$rt_{(2,1)}$	$rt_{(2,2)}$	....	$rt_{(2,t)}$	....	$rt_{(2,T)}$
:	:	:	..:..	:		:
R_r	$rt_{(r,1)}$	$rt_{(r,2)}$	....	$rt_{(r,t)}$	....	$rt_{(r,T)}$
:	:	:		:		:
R_R	$rt_{(R,1)}$	$rt_{(R,2)}$	....	$rt_{(R,t)}$	....	$rt_{(R,T)}$

(a) Binary valued matrix for MILP



(b) Encoding for GA-based optimization

Fig. 4. Optimization for system partitioning DSE.

Crossover: We use the following two operations for implementing crossover between two individuals of a generation – As shown in Fig. 6a, we implement a **two-point** crossover for exchanging the PRR configuration data of some tasks – determined by the two randomly selected crossover points. Further, we use a **single-point** crossover, as shown in Fig. 6b, to exchange the scheduling information of some tasks – determined by the same randomly selected point in the two sequences.

Mutation: Mutation is essential for preventing the search from getting trapped in some local maxima. As shown in Fig. 5b, we use a **singlepoint** mutation for randomly altering the PRR configuration of a randomly selected task. Further, we implement a **two-point** mutation for altering scheduling data by swapping the position of two randomly selected sub-sequences, as shown in Fig. 5a.

Selection: We use a tournament selection method for choosing individuals for the next generation. This selection method involves randomly choosing 3 (in our case) individuals from the current population and selecting the one with *best* fitness for the next generation. The population size for each generation is kept constant by repeating the selection process for a number of times equal to the population size.

4.5. Floorplanning

The automatic floorplanning tool PRFloor [17] is used to verify the feasibility of the mapping generated by the MILP solver.

5. Experiments and results

5.1. Experiment setup

All our experiments are run on a computer with two CPUs – Intel^{TM} Xeon^{TM} E5-2609 v2 @ 2.50 GHz (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 [18] is used to solve our scheduler with parameter $Presolve = 2$. The PR-HMPSoC template is provided by Nguyen et al. [15] together with the floorplanner, PRFloor [17].

Experiments for performance evaluation involved using synthetic application task-graphs with varying number of tasks. These task-graphs were generated using Task Graphs for Free (TGFF) tool [19]. Further, 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 as shown in Fig. 7. Each task can be considered as a specific function that is realized by an accelerator. The hardware implementation of each such accelerator represents a PRM that has to be mapped to a PRR to execute its functionality. The tasks for each application task-graph were randomly selected from the set of PRMs described in the next sub-section. Further, PRM sets with varying range of resource requirements were used for evaluation.

5.2. IP pool

In this work, we collected 50 real-world hardware accelerators (PRMs) from CHStone benchmarks [20], Opencores [21], EPFL benchmark [22] and Xilinx XPS IP core library [23]. Table 2 shows several notable PRMs out of the 50 PRMs used in the experiments. The resources requirement from the synthesis report and operating temperature obtained from Xilinx Vivado HDL Power Analysis Tool are also provided. As can be seen, the sizes and operating temperature of the PRMs vary quite significantly which reflect the real-world requirements. 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 of 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 into our proposed flow to perform a more accurate analysis. In Ref. [17], the authors mention the discrepancy between the resource usages of PRMs obtained from the synthesis reports versus the actual physical occupation when they are placed on the FPGA. If our lifetime-aware scheduler does not take this issue into account, it may blindly map PRMs to PRRs that results in a design which is too big to implement. Therefore, we utilize our in-house tool to predict the resources occupied by the PRMs after placement. The actual PRM resources used by our lifetime-aware scheduler to build the MILP program are therefore larger. However, they reflect the actual resources occupation on FPGA more accurately.

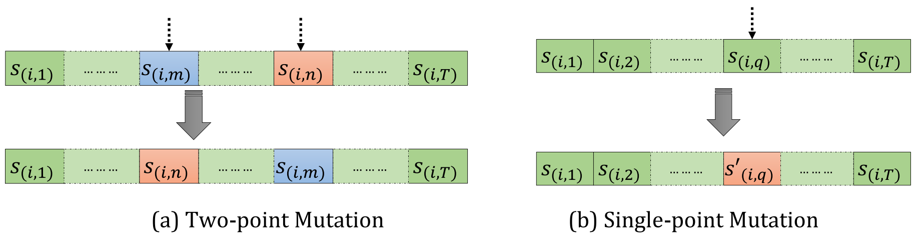


Fig. 5. Mutation.

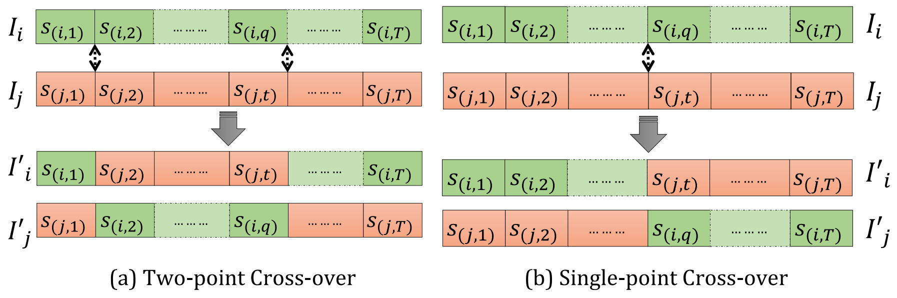


Fig. 6. Cross-over.

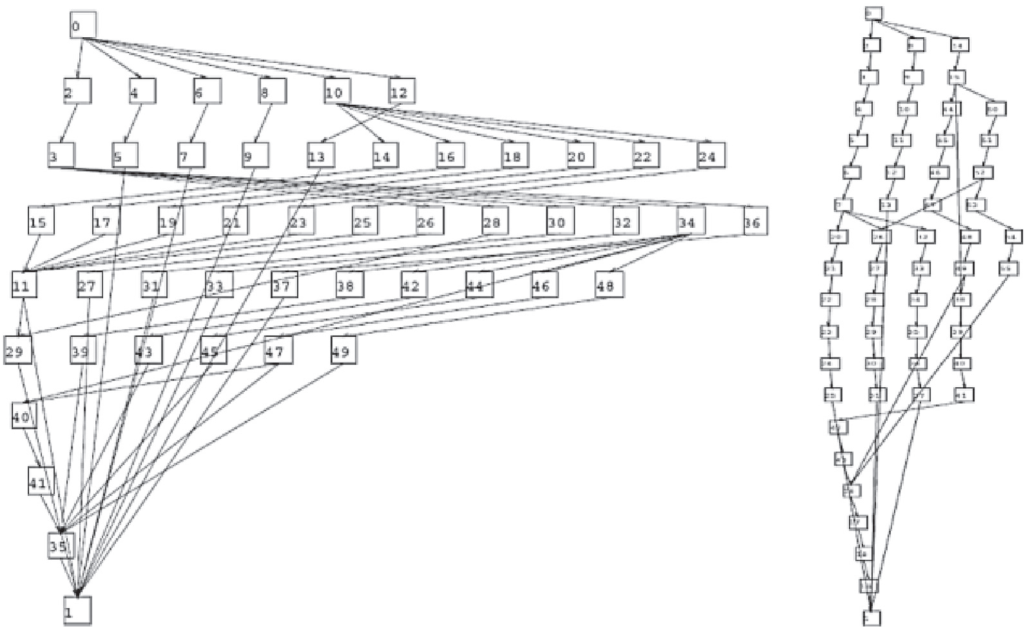


Fig. 7. *Fat* and *Slim* application task-graphs. Both the task-graphs have 50 tasks each with deadlines of 1000 and 2000 time units respectively.

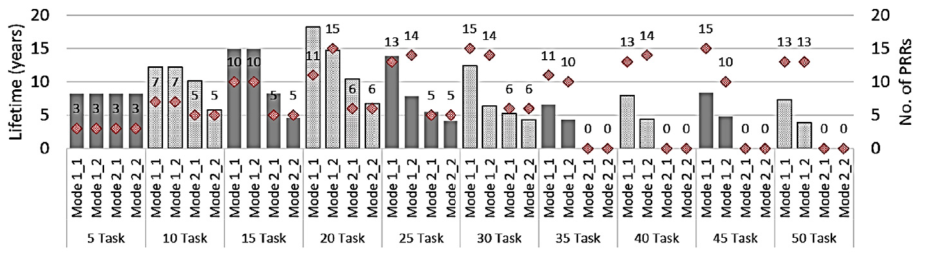
Table 2
Several notable PRMs used in the experiments.

PRM	LUT	BRAM	DSP	Temp(C)	Lifetime	Source
DFDIV	7309	1	24	96.8	3.3	CHStone
DFMUL	4051	1	16	82.6	5.6	CHStone
Log2	8212	0	0	125.0	2.0	EPFL
ADPCM	6222	6	126	125.0	2.0	OpenCores
FFT1024	19796	18	52	125.0	2.0	OpenCores
SHA	3069	20	0	27.2	10.0	OpenCores
JPEG	6581	11	10	120.4	2.0	OpenCores
Video Stream Scaler	524	2	11	59.0	9.0	Xilinx
Video Test Pattern	2543	3	12	56.3	8.5	Xilinx
Microblaze (Max Area)	5539	5	6	125	2.0	Xilinx

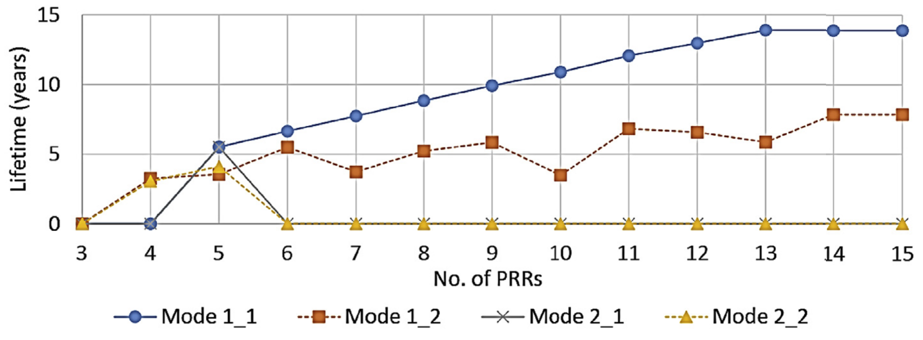
5.3. MILP-based system partitioning

To estimate the performance of the MILP-based scheduler and heterogeneous PRR-based systems simultaneously, we performed experiments in 4 modes. *Mode* 1_1 and *Mode* 1_2 refer to the 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 all cases beyond that. Applications with increasing number of tasks – from 5 to 50 and in increments of 5 – were used for the experimental evaluation.

We quantify the performance of our proposed methodology in terms of the system MTTF. Figs. 8a, 9, 10a and 11a show the result for all four



(a) *SysMTTF* in *Fat* applications with large PRMs



(b) Variation of *SysMTTF* with number of PRRs for a *Fat* task-graph with 25 tasks and large PRMs

Fig. 8. *Fat* applications with large PRMs.

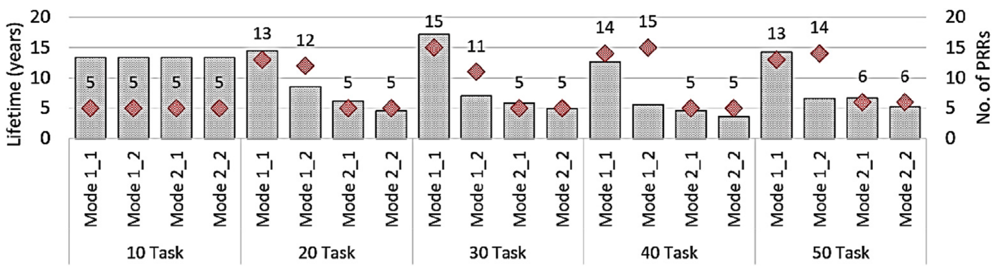
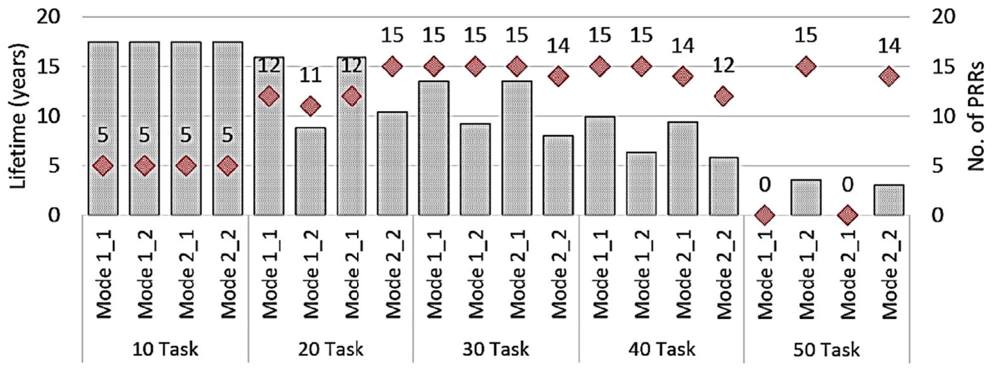
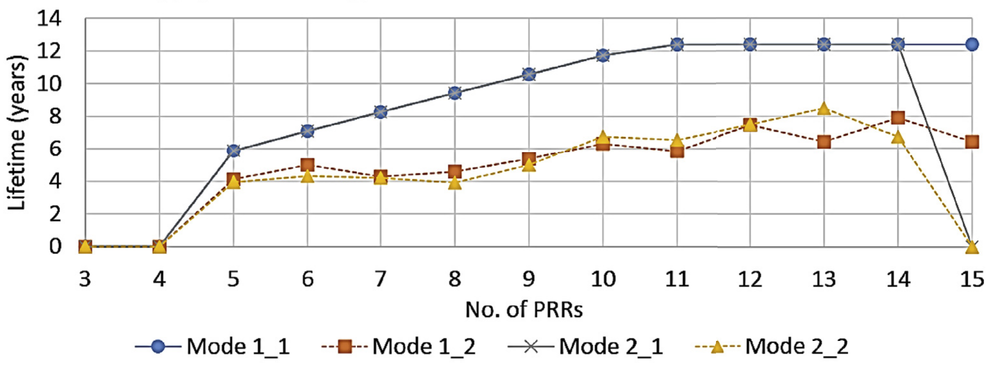


Fig. 9. *SysMTTF* in *Slim* applications with large PRMs.



(a) *SysMTTF* in *Fat* applications with small PRMs



(b) Variation of *SysMTTF* with number of PRRs for a *Fat* task-graph with 25 tasks and small PRMs

Fig. 10. *Fat* applications with small PRMs.

modes for different *scenarios* – applications with a 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 the number of PRRs, *R*, for each mode. The minimum values of *R* at which we obtain the maximum MTTF for a mode are shown as labelled markers in the figure.

Figs. 8b, 10b and 11b 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 next.

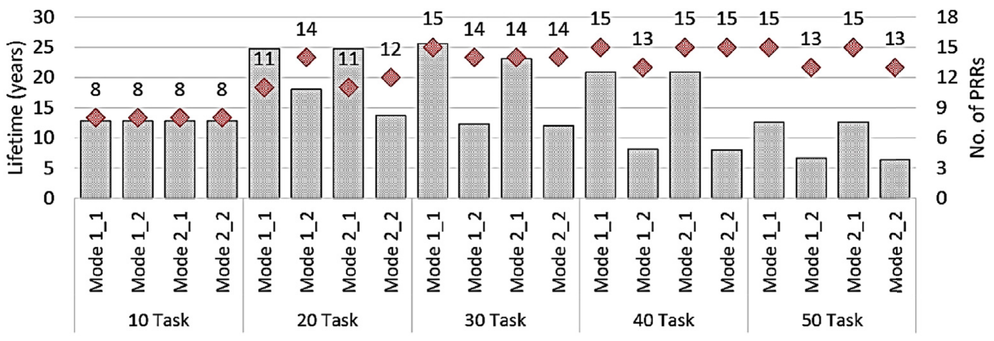
5.3.1. *Lifetime Reliability-aware scheduling*

The system's MTTF (*SysMTTF*) obtained using the lifetime-aware scheduler (*Mode* *_1), shows a considerable increase over a makespan-optimization approach (*Mode* *_2) for almost every scenario.

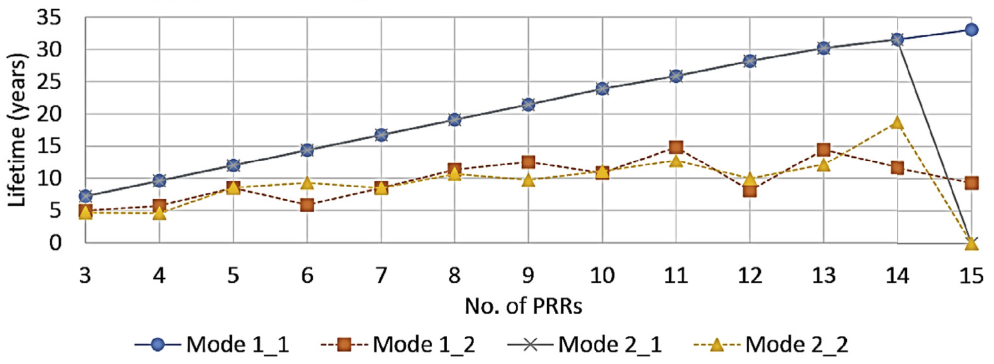
In applications with very few tasks (e.g. 10 tasks in Figs. 10a and 11a), both scheduling modes exhibit similar performance. In smaller task-graphs, the scope for improvement of system lifetime is limited as there are insufficient tasks to exploit the parallelism. In our current work, we are yet to explore the effect of having redundant PRRs, created from the spare DPR resources and is left for future research. We only consider applications with enough 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. 8b (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 behaviour can also be observed in Fig. 10a 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 an improved lifetime. In some cases, the scheduler performance flattens beyond a certain point. In Fig. 8b there are no improvements beyond 13 PRRs, as the resource redistribution to create additional heterogeneous



(a) *SysMTTF* in *Slim* applications with small PRMs



(b) Variation of *SysMTTF* with number of PRRs for a *Slim* task-graph with 25 tasks and small PRMs

Fig. 11. *Slim* applications with small PRMs.

PRRs, does not result in extra PRRs for the more stress-inducing PRMs. Similarly, as shown in Fig. 10b, performance benefits with the aging-aware scheduler do not improve beyond 11 PRRs for either homogeneous or 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 over our baseline makespan optimization scheduler, irrespective of the heterogeneity of PRRs.

5.3.2. *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 3 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 to accommodate the largest PRM. 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. 8a, the homogeneous system design fails for *Fat* applications with 35 or more tasks.

In the 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. 8b, the homogeneous system can support only 5 PRRs compared to a heterogeneous system with up to 15 PRRs. The additional PRRs were

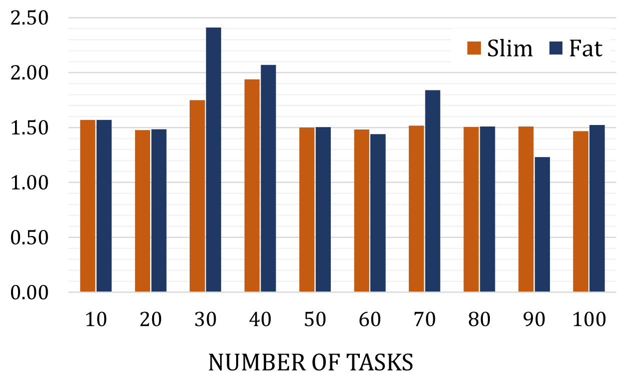


Fig. 12. Average improvement in System MTTF across different tolerances to $Spec_{MS}$. The vertical axis indicates the ratio of system MTTF due to *HetPRRs* to that using *HomPRRs.*

Table 3
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

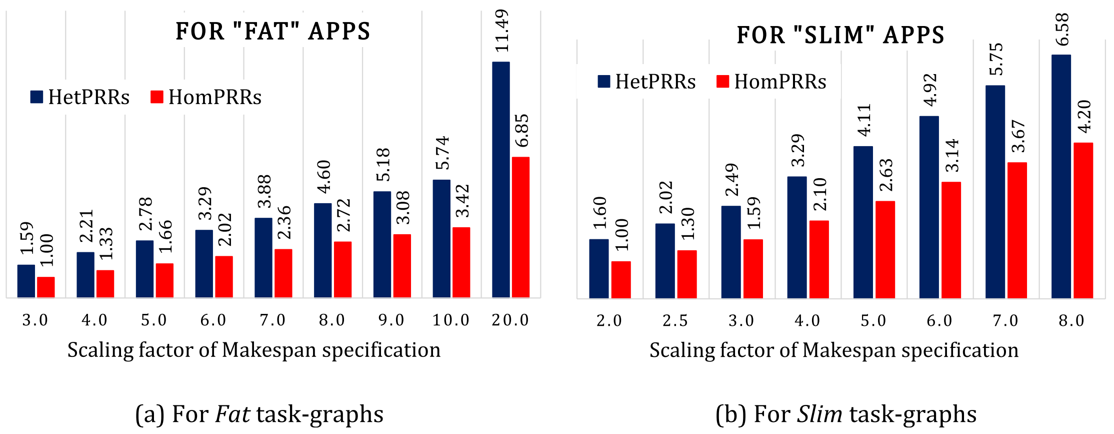


Fig. 13. Scaling with increasing $Spec_{MS}$.

used to accommodate the more stress-inducing PRMs, and show improvements in system MTTF for up to 13 PRRs. The flattening of performance beyond that was explained in Section 5.3.1.

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 the 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. 10a and 11a, 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 providing adequate parallelism. As seen in Figs. 10b and 11b, 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.

5.4. GA-based multi-objective system partitioning

Similar to the MILP-based optimization, evaluating the GA-based methodology involved comparing the performance of *HomPRR* and *HetPRR* based system partitioning with applications of two types – *Fat* and *Slim*– and increasing number of tasks – 10, 20, 30 … 100. To demonstrate the multi-objective DSE, the *SysMTTF* at different values of $Spec_{MS}$ was obtained to generate the Pareto front. The different values for $Spec_{MS}$ were obtained by scaling the *critical path length* of the application under test. The GA framework from Ref. [24] is used in this work to implement the proposed methods. The parameters used in the GA-based DSE are mentioned below:

Starting population: 500
Maximum generations: 100
Crossover probability: 0.7
Mutation probability: 0.3

Fig. 12 shows the average improvement in *HetPRR*-based *SysMTTF* over systems using *HomPRRs* across all experiments with different values of $Spec_{MS}$. Each *bar* in the figure shows the average ratio of *SysMTTF* with *HetPRRs* to that using *HomPRRs*, for applications of different type and having a varying number of tasks. Similar to our results from MILP-based system partitioning, *HetPRR*-based systems exhibit considerable improvements (denoted by values > 1.0) in *SysMTTF* for all cases, with up to $\sim 2.5x$ improvements in a *Fat* application with 30 tasks.

Fig. 13 shows the average performance scaling comparison for *HetPRR* and *HomPRR* based systems for each type of application under increasing values for the constraint *SpecMS*. The average improvements normalized w.r.t. the lowest valued *SysMTTF* across all Pareto fronts are shown in the figure. As evident from the figure, the *HetPRR*-based systems show better scaling with increasing relaxations in $Spec_{MS}$ constraints across both application types – *Fat* and *Slim*. The starting scaling factor for $Spec_{MS}$ relaxation was chosen for the case (type and number of tasks) that has at-least one feasible point for both types of PRR-based system.

The Pareto-front plots in Figs. A.14, and B.15 show the detailed optimization results for the trade-offs w.r.t. *SysMTTF* and *SysMS* for all application cases. The points marked with star (☆) on the curves represent the infeasible points w.r.t. the $Spec_{MS}$ constraint. In both cases of application types, the *HomPRR*-based systems show more of such infeasible points compared to the *HetPRR* systems. This can be explained by the lack of resources to generate the sufficient number of homogeneous PRRs required for the parallelism needed to complete application execution within the specified $Spec_{MS}$ constraint. Further, the Pareto-fronts obtained with *HetPRRs* are almost always better than that using *HomPRRs*, signifying more efficient multi-objective optimization.

6. Conclusion

In this work, we propose a novel lifetime-aware proactive methodology DPR-based system design. Our approach analyses 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. This information is used to build optimization problems for MILP-based solver and GA-based stochastic search. The PRRs of the resulting system are heterogeneous leading to more efficient usage of the FPGA resources for improving the system lifetime. Further, we propose a multi-objective DSE for system partitioning. Our experiments show that the heterogeneous systems can offer more than 2x lifetime improvement over homogeneous ones and show better scaling with increasing makespan specification. Currently, we are working on exploring the possibility of having more heterogeneous PRRs as redundancy resources to obtain appropriate trade-offs between aging mitigation and tolerance against external faults. Future research is required for adding more metrics to the multi-objective problem to enable increased application-specific design.

Acknowledgements

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 TU Dresden and the HiPEAC4 Network of Excellence.

Appendix A. Pareto plots for fat applications

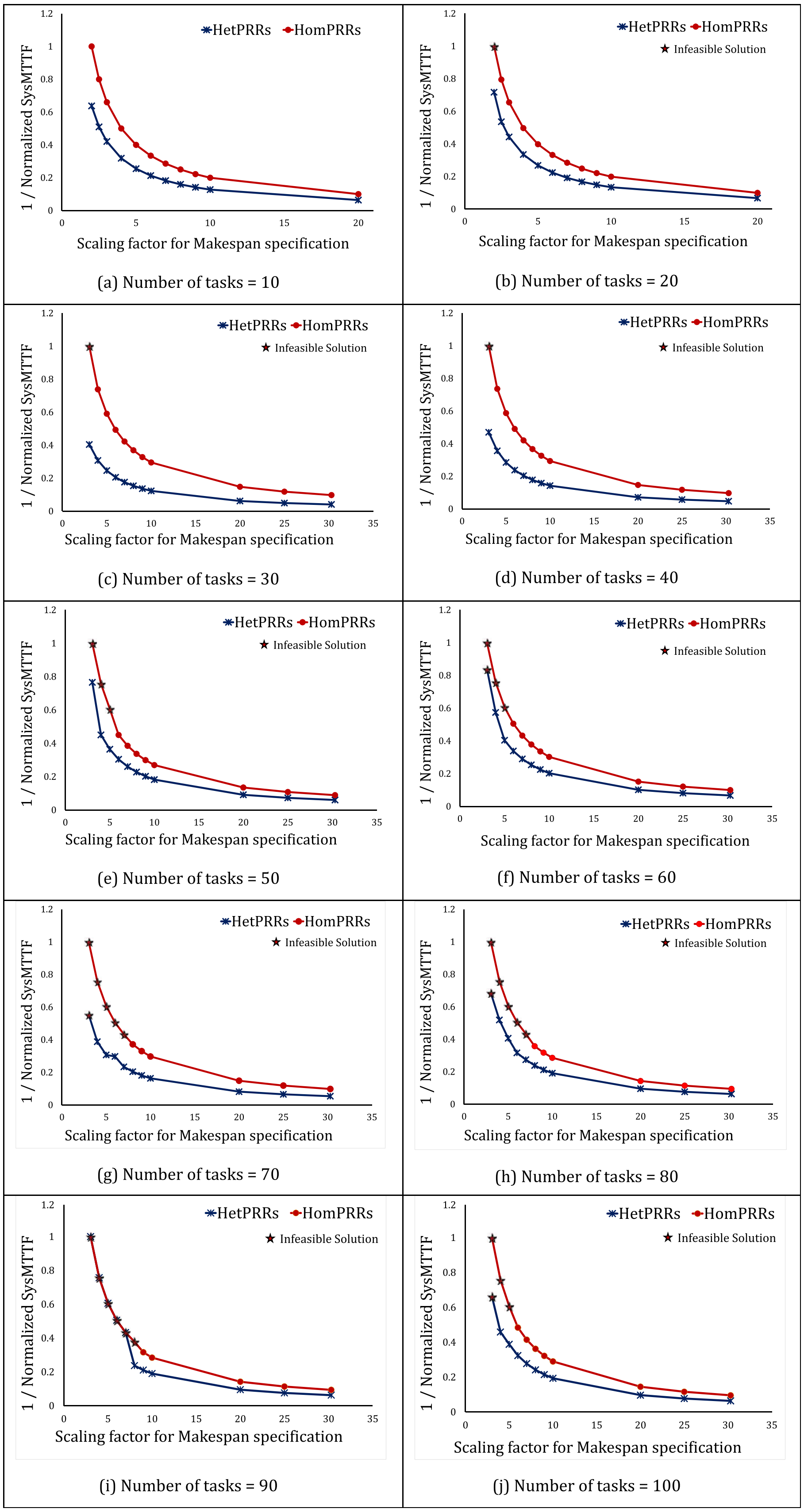


Fig. A.14. Pareto fronts for *F at* task-graphs.

Appendix B. Pareto plots for slim applications

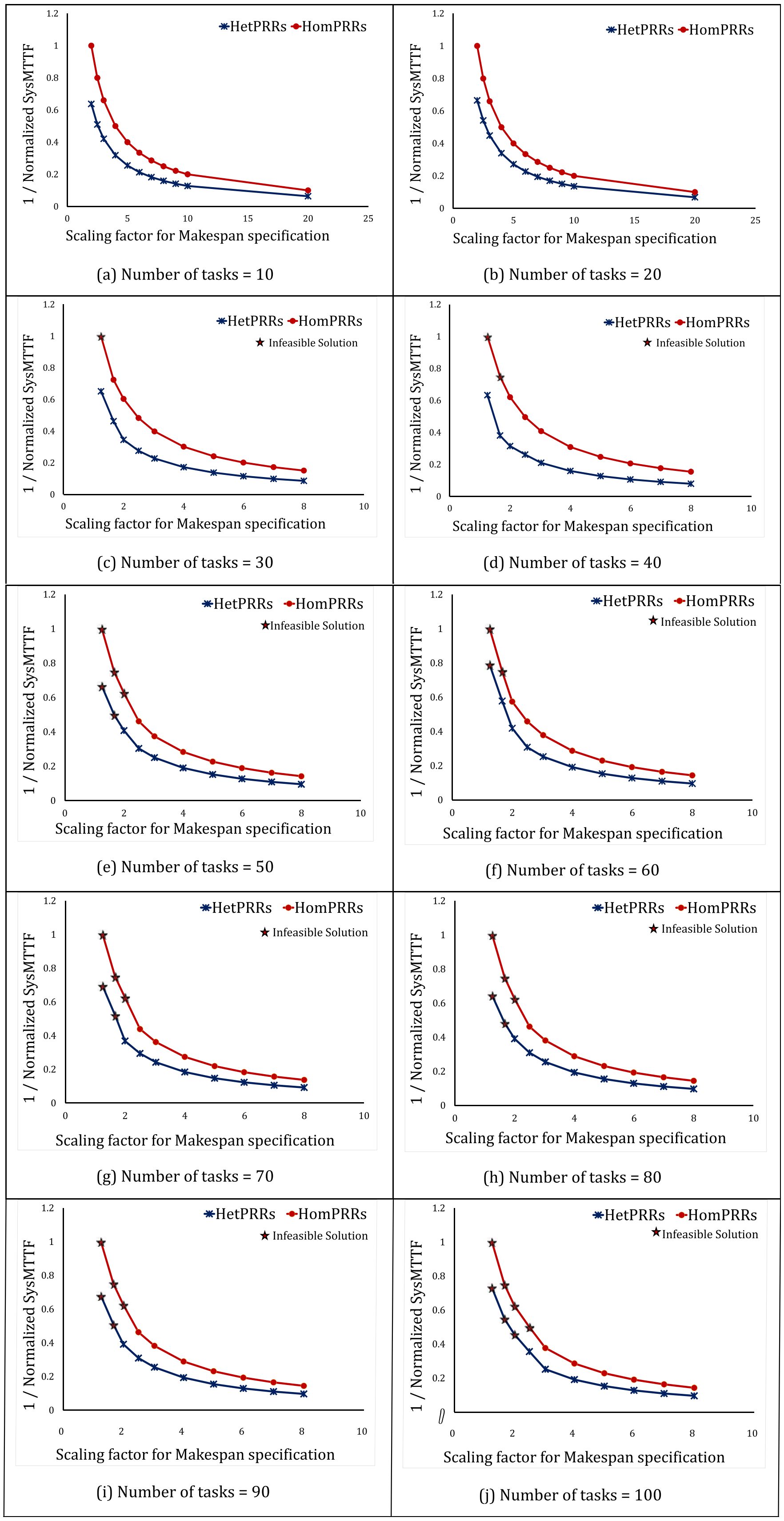


Fig. B.15. Pareto fronts for *Slim* task-graphs.

References

[1] M. Glesner, H. Hinkelmann, T. Hollstein, L.S. Indrusiak, T. Murgan, A.M. Obeid, M. Petrov, T. Pionteck, P. Zipf, Reconfigurable embedded systems: an application-oriented perspective on architectures and design techniques, in: T.D. Hämäläinen, A.D. Pimentel, J. Takala, S. Vassiliadis (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 12–21.

[2] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, B. He, COMBA: a comprehensive model-based analysis framework for high level synthesis of real applications, in: 2017 IEEE/ACM International Conference on Computer-aided Design (ICCAD), 2017, pp. 430–437.

[3] S. Wang, Y. Liang, W. Zhang, FlexCL: an analytical performance model for OpenCL workloads on flexible FPGAs, in: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

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

[5] C. Bird, V.-P. Ranganath, T. Zimmermann, N. Nagappan, A. Zeller, Extrinsic influence factors in software reliability: a study of 200,000 windows machines, in: Companion Proceedings of the 36th International Conference on Software Engineering, ICSE Companion 2014, ACM, New York, NY, USA, 2014, pp. 205–214. http://doi.acm.org/10.1145/2591062.2591173.

[6] S.S. Sahoo, T.D.A. Nguyen, B. Veeravalli, A. Kumar, Lifetime-aware design methodology for dynamic partially reconfigurable systems, in: 23rd Asia and South Pacific Design Automation Conference, ASP-DAC 2018, Jeju, Korea (South), January 22-25, 2018, 2018, pp. 393–398. https://doi.org/10.1109/ASPDAC.2018.8297355.

[7] E.A. Stott, J.S. Wong, P. Sedcole, P.Y. Cheung, Degradation in FPGAs: measurement and modelling, in: Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '10, ACM, New York, NY, USA, 2010, pp. 229–238. http://doi.acm.org/10.1145/1723112.1723152.

[8] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M.J. Irwin, K. Sarpatwari, Toward increasing FPGA lifetime, IEEE Trans. Dependable Secure Comput. 5 (2) (2008) 115–127, https://doi.org/10.1109/TDSC.2007. 70235.

[9] E. Stott, P.Y.K. Cheung, Improving FPGA reliability with wear-levelling, in: 2011 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 323–328, https://doi.org/10.1109/FPL.2011.65.

[10] Z. Ghaderi, E. Bozorgzadeh, Aging-aware high-level physical planning for reconfigurable systems, in: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 631–636, https://doi.org/10.1109/ASPDAC.2016.7428082.

[11] H. Zhang, L. Bauer, M.A. Kochte, E. Schneider, C. Braun, M.E. Imhof, H.J. Wunderlich, J. Henkel, Module diversification: fault tolerance and aging mitigation for runtime reconfigurable architectures, in: 2013 IEEE International Test Conference (ITC), 2013, pp. 1–10, https://doi.org/10.1109/TEST.2013.6651926.

[12] H. Zhang, M.A. Kochte, E. Schneider, L. Bauer, H.J. Wunderlich, J. Henkel, STRAP: stress-aware placement for aging mitigation in runtime reconfigurable architectures, in: 2015 IEEE/ACM International Conference on Computer-aided Design (ICCAD), 2015, pp. 38–45, https://doi.org/10.1109/ICCAD.2015.7372547.

[13] J. Angermeier, D. Ziener, M. Gla, J. Teich, Stress-aware module placement on reconfigurable devices, in: 2011 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 277–281, https://doi.org/10.1109/FPL.2011. 56.

[14] V. Dumitriu, L. Kirischian, V. Kirischian, Run-time recovery mechanism for transient and permanent hardware faults based on distributed, self-organized dynamic partially reconfigurable systems, IEEE Trans. Comput. 65 (9) (2016) 2835–2847, https://doi.org/10.1109/TC.2015.2506558.

[15] T.D.A. Nguyen, A. Kumar, PR-HMPSoC: a versatile partially reconfigurable heterogeneous Multiprocessor System-on-Chip for dynamic FPGA-based embedded systems, in: 2014 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 1–6, https://doi.org/10.1109/FPL.2014.6927492.

[16] Y. Xiang, T. Chantem, R.P. Dick, X.S. Hu, L. Shang, System-level reliability modeling for MPSoCs, in: 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010, pp. 297–306.

[17] T.D. Nguyen, A. Kumar, PRFloor: an Automatic Floorplanner for Partially Reconfigurable FPGA Systems, 2016, pp. 149–158. http://doi.acm.org/10.1145/2847263.2847270.

[18] Gurobi, Gurobi Optimization Version 6.0.2, 2017. www.gurobi.com.

[19] R.P. Dick, D.L. Rhodes, W. Wolf, TGFF: task graphs for free, in: Hardware/software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on, 1998, pp. 97–101, https://doi.org/10.1109/HSC.1998.666245.

[20] Y. Hara, H. Tomiyama, S. Honda, H. Takada, Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis, J. Inf. Proces. 17 (2009) 242–254, https://doi.org/10.2197/ipsjjip.17.242. Released October 07, 2009, Online ISSN 1882-6652.

[21] OpenCores, 2017. www.opencores.org.

[22] EPFL, Combinational Benchmark Suite, lsi.epfl.ch/benchmarks.

[23] Xilinx, Intellectual Property, 2017. www.xilinx.com/products/intellectualproperty.html.

[24] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: evolutionary algorithms made easy, J. Mach. Learn. Res. 13 (2012) 2171–2175.

Siva Satyendra Sahoo received his B.Tech. degree in Instrumentation and Electronics Engineering from College of Engineering and Technology, Bhubaneswar, India, in 2008, and the M.Tech. degree in Electronics System Design from the Indian Institute of Science, Bangalore, India in 2012. He is currently pursuing the PhD degree in fault-tolerant embedded system design with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests are fault-tolerance in heterogeneous embedded systems, security design in embedded systems and reconfigurable computing.

Tuan Duy Anh Nguyen has obtained the B.Eng. in computer engineering from Ho Chi Minh City University of Technology, Vietnam, in 2011, and the PhD degree in computer engineering from National University of Singapore, Singapore, in 2018. His interests are in Partially Reconfigurable Heterogeneous Multiprocessor System-on-Chip. He has published research works in major FPGA conferences such as the International Conference on Field Programmable Logic and Applications (FPL), International Symposium on Field-Programmable Gate Arrays (FPGA) and Asia and South Pacific Design Automation Conference (ASP-DAC). His work was also nominated for the best paper award at the FPL 2014. He is currently with the Technische Universität Dresden, Dresden, Germany, where he works as PostDoctoral researcher at Chair for Processor Design.

Bharadwaj Veeravalli received his BSc degree in physics, from Madurai-Kamaraj University, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science, Bangalore, India in 1991, and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He is currently with the Department of Electrical and Computer Engineering, at The National University of Singapore, Singapore, as a tenured associate professor. His main stream research interests include, cloud/grid/cluster computing, scheduling in parallel and distributed systems, bioinformatics and computational biology, and multimedia computing. He had successfully secured several externally funded projects and published more than 160 papers in high-quality International Journals and Conferences and three research monographs.

Akash Kumar received his B.Eng. degree in computer engineering from the National University of Singapore (NUS), Singapore, in 2002, the masters degree in technological design with a minor in embedded systems jointly from NUS and the Eindhoven University of Technology (TUe), Eindhoven, The Netherlands, in 2004, and the Ph.D. degree in electrical engineering with a minor in embedded systems jointly from TUe and NUS, in 2009. He is currently with the Technische Universität Dresden, Dresden, Germany, where he directs the Chair for Processor Design. He has authored over 100 papers in leading international electronic design automation journals and conferences in his research areas. His current research interests include design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems. Dr. Kumar was a recipient of the best paper award nominations, including the Conference on Field Programmable Logic and Applications (FPL) in 2014, GLSVLSI in 2014, SC in 2015, and the Design, Automation & Test in Europe Conference (DATE) in 2015.