Reliability and Makespan Optimization of Hardware Task Graphs in Partially Reconfigurable Platforms

Reza Ramezani, Yasser Sedaghat, Mahmoud Naghibzadeh, *Fellow, IEEE,* and Juan Antonio Clemente

***Abstract*—This paper addresses the problem of reliability and makespan optimization of hardware task graphs in reconfigurable platforms by applying fault tolerance (FT) techniques to the running tasks based on the exploration of the Pareto set of solutions. In the presented solution, in contrast to existing approaches in the literature, task graph scheduling, tasks parallelism, reconfiguration delay, and FT requirements are taken into account altogether. This paper firstly presents a model for hardware task graphs, task prefetch and scheduling, reconfigurable computer and a fault model for reliability. Then, a mathematical model of an integer nonlinear multi-objective optimization problem is presented for improving the FT of hardware task graphs, scheduled in partially reconfigurable platforms. Experimental results show the positive impacts of choosing the FT techniques selected by the proposed solution, which is named *Pareto-based*. Thus, in comparison to non-fault-tolerant designs or other state-of-the-art FT approaches, without increasing makespan, about 850% Mean Time To Failure (MTTF) improvement is achieved and, without degrading reliability, makespan is improved by 25%. In addition, experiments in fault-varying environments have demonstrated that the presented approach outperforms existing state-of-the-art adaptive FT techniques in terms of both MTTF and makespan.**

***Index Terms*—Fault Tolerance, Optimization, Reconfigurable Platforms, Reliability, Scheduling.**

I. Introduction and Related Work

NEW-GENERATION safety- and mission-critical embedded systems, and especially space computing ones, demand high performance, reliability, efficiency, and flexibility [1]. In these systems, as the data collection rate surpasses the data communication rate, increasing systems' onboard data processing is required to mitigate data transmission bottlenecks. However, on-board computers are exposed to more radiations in comparison to those processing data at the Earth ground level, which leads to significant reliability degradations [2]. Hence, reliability and performance should be considered simultaneously for such systems [3].

Including hardware support in computations helps to speed up the execution and gain a better performance, which was traditionally done by using Application-Specific Integrated Circuits (ASICs). However, although ASICs are a very efficient option, they suffer from: fixed functionality, increased time-to-market of development, and expensive bug detection or functionality improvement. Dynamically Reconfigurable Hardware (DRH) is a key solution to address these drawbacks. The DRH paradigm [4] is a combination of the two traditional computing paradigms: Application Specific Computing (ASC) and General Purpose Computing (GPC). The most widespread reconfigurable computers are SRAM-based Field Programmable Gate Arrays (FPGAs), which offer a trade-off between performance and flexibility [5], [6]. These devices can be reconfigured multiple times at run-time, which makes a single FPGA capable of executing multiple functionalities in a time-multiplexed manner. This feature allows executing hardware task graphs whose size is bigger than the FPGA, by employing appropriate scheduling techniques [7].

Despite flexibility and good performance, as SRAM-based FPGAs are frequently implemented with memory cells susceptible to radiation-induced soft errors [8], they suffer from the so-called Single Event Effects (SEEs) [9], such as Single Event Upsets (SEUs), Multiple Bit Upsets (MBUs) and Multiple Cell Upsets (MCUs). These upsets can lead to the FPGA functionality failure, therefore some Fault Tolerance (FT) techniques are required to reduce the impacts of these upsets. However, FT techniques imply several overheads (e.g. performance, power consumption, hardware, and timing expenses) to the system. Hence the FT techniques should be chosen appropriately so that not only system reliability is increased, but also the corresponding overheads do not increase significantly.

Several studies have concentrated on improving the reliability and FT aspects of FPGAs without considering the scheduling problem. These studies can be categorized into two groups of mitigation techniques, namely: design-based methods and recovery techniques. Design-based methods involve redundancy and they can be applied at different granularities and different levels of design, independently of the final application [10], [11]. Recovery-based methods are especially designed for reconfigurable computers to prevent fault accumulation at run-time [12]. These methods employ scrubbing or task reallocation to periodically refresh the contents of the memory cells. In any case, most of the studies on FT techniques in reconfigurable computers assume that all the tasks can be placed simultaneously on the reconfigurable computer, and therefore they have not taken any scheduling strategy into account [13], [14]. Nevertheless, modern embedded systems with reconfigurable coprocessors have very limited resources availability [15]. Hence, scheduling techniques are required for

Manuscript received December 18, 2015; revised 31 May, 2016; accepted September 19, 2016.

This work was supported in part by the Ministry of Science, Research and Technology of Iran and by the Spanish MCINN project TIN2013-40968-P, *Corresponding author: Yasser Sedaghat*

R. Ramezani, Y. Sedaghat and M. Naghibzadeh are with the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran (e-mails: reza.ramezani@stu.um.ac.ir, y_sedaghat@um.ac.ir, naghibzadeh@um.ac.ir).

J. A. Clemente is with the Computer Architecture Department, Universidad Complutense de Madrid, Madrid 28040, Spain (e-mail: ja.clemente@fdi.ucm.es).

the running applications to achieve an acceptable performance in reconfigurable platforms.

However, very few studies have considered both scheduling and FT techniques simultaneously. These methods try to increase the reliability whereas a given performance is guaranteed. For instance, a *Primary/Backup* scheme is proposed in [16] in order to support FT while scheduling. In [17], a real-time fault-tolerant scheduling algorithm is presented so that, by using a sufficient schedulability test condition, it schedules hybrid hardware/software tasks that are able to tolerate f_i faults during task execution. Recently, we have investigated the application of different FT strategies on different real-time scheduling algorithms in reconfigurable platforms [18].

This paper presents an approach that enhances traditional scheduling methods by considering FT requirements of the running applications, which are represented as *Directed Acyclic Graphs* (DAGs) [19], (or simply Task Graphs (TG) in the remainder of this paper), and that will run on a reconfigurable computer. For that purpose, it applies active redundancy-based FT techniques to the hardware tasks, by applying a multi-objective optimization algorithm to improve both their reliability and total execution time (which is known as *makespan*). An *As Soon As Possible (ASAP)* scheduling algorithm is used to steer the reconfiguration and execution of the applications in the reconfigurable resources, by taking the data dependencies among tasks and the configuration overheads into account. Thus, this paper demonstrates how this ASAP-based scheduling approach can be greatly improved by considering both task execution time and applications reliability simultaneously. In the proposed solution, task replication is used for FT and task reconfiguration is employed as a recovery-based method. Experimental results will show the positive effects of the presented solution in comparison with other fixed FT techniques so that without degrading the task graphs makespan, their Mean Time to Failure (MTTF) can be improved significantly and vice versa. Supplementary experiments in fault-varying environments will demonstrate that the presented approach outperforms other state-of-the-art adaptive FT techniques [10].

The remainder of the paper is organized as follows. Section II describes the system model. Section III presents a motivational example. Next, Section IV presents the proposed methodology. Section V gives the experimental evaluations and the obtained results, and finally Section VI concludes the paper and provides suggestions for future work.

II. System Model

A. Task Model

In the proposed solution, hardware tasks are modeled as DAGs, whose nodes represent computational tasks, and whose edges indicate dependencies among the tasks. In this paper each hardware task τ_i is modeled as follows:

$$\tau_i = \{CT_i, CC_i, TS_i, SB_i\} \tag{1}$$

where CT_i is the computation time of Task τ_i, CC_i is task size in terms of Configurable Logic Blocks (CLBs) count [20], TS_i is task size in the configuration memory (number of configuration bits), and SB_i is the percent of sensitive bits of the task. Any upset in a sensitive bit will eventually affect the functionality of the corresponding task and leads to a failure [21]. The number of sensitive bits of a design can be estimated by means of fault injection, fault emulation or even radiation-ground experiments [22].

Since the validity of the presented model has been verified on a Virtex-5 FPGA, in this case the basic elements are the so-called *Configurable Logic Blocks* (CLBs). However, this idea can be easily ported to any other FPGA in the market, since all of them feature any sort of basic elements, such as, for instance, *Logic Elements* (LEs) in Altera architectures.

It is assumed that the task graph is executed successfully if all the tasks are executed correctly [23]. In addition, it is assumed that task graphs are soft real-time, that decreasing makespan leads to increasing Quality of Service (QoS) [7], such as the H.264 video encoder [24].

B. Reconfigurable Computer Model

The target FPGA features partial runtime reconfigurability and it includes an array of CLBs, such that hardware tasks are synthesized and mapped on a subset of them. Typically, in partially reconfigurable systems, a CLB is not the minimum addressable segment of the device that can be reconfigured separately, but a group of them, which is referred to in this paper as *CLB group*. Thus, the target partially reconfigurable computer RC is characterized as:

$$\mathrm{RC} = (RO, CO, GS, CD) \tag{2}$$

where RO and CO indicate row count and column count (in terms of CLBs) of the reconfigurable computer. GS stands for CLB group size (in terms of CLB count in each group), and finally CD is the configuration delay of a CLB group. The configurations are carried out in a serial manner [7].

For practical reasons, it has been assumed that the tasks are reconfigured in the target device following a 1D reconfiguration model. In other words, tasks span the whole height of the device, and occupy a number of columns of CLBs. Thus, the number of columns of CLBs needed to implement Task τ_i on the target reconfigurable device is shown as $ColumnCount_{\tau_i}$. For a given Task τ_i, its size CC_i (in terms of CLBs) and its $ColumnCount_{\tau_i}$ are related as follows:

$$ColumnCount_{\tau_i} = \left\lceil \frac{CC_i}{RO} \right\rceil \tag{3}$$

In the proposed solution, hardware tasks can be simultaneously placed on the reconfigurable computer as long as their total column count is less than or equal to the device's column count CO [25], i.e.:

$$\sum_{\tau_k \in Running\ Tasks} ColumnCount_{\tau_k} \leq CO \tag{4}$$

C. Reliability and Fault Model

As indicated by [26], different altitudes above the Earth surface have different soft error rates (SERs) that can be measured *per bit per time unit*. In this work, the reliability

model presented in [27] has been used. In this model, the hardware tasks unreliability can be calculated as $Q_{\tau_i} = 1 - R_{\tau_i} = P(F_{\tau_i})$ where R_{τ_i} is the task reliability and $P(F_{\tau_i})$ is the probability of failure of Task τ_i given j upsets during its execution, $j \geq 1$, i.e:

$$P(F_{\tau_i}) = \sum_{j=1}^{\infty} P(F_{i,j}) \tag{5}$$

As not all the bits of a task are sensitive, let SB_i indicate "upset in a sensitive bit of Task τ_i". Thus, $P(F_{i,j})$ is the conditional probability of at least one bit flip in the sensitive bits of Task τ_i, given j upsets:

$$P(F_{i,j}) = P(U_{i,j})\, P(SB_i \mid U_{i,j}) \tag{6}$$

where $P(SB_i \mid U_{i,j}) = 1 - (1 - SB_i)^j$. As SERs follow the Poisson distribution, Eq. (7) can be used to estimate the probability of j bit flips in Task τ_i when it is running:

$$P(U_{i,j}) = e^{-v_i} \frac{{v_i}^j}{j!} \tag{7}$$

where $v_i = \mu \times TS_i \times (RT_i + CT_i)$, in which μ is the SER expressed in *bit per time* unit [26], CT_i is task computation time, and RT_i is residency time of Task τ_i indicating the time elapsed from when it is configured until it starts its execution. The SER can be estimated by some modeling tools such as CREME96 [28].

Once the task unreliability Q_{τ_i} is known, the failure rate (number of failures per time unit per task) of a task during its execution can be estimated as follows [2]:

$$\lambda_{\tau_i} = \frac{Q_{\tau_i}}{CT_i} \tag{8}$$

And finally the Mean Time to Failure (MTTF) as an indicator of reliability is easily calculated as:

$$MTTF_{\tau_i} = \frac{1}{\lambda_{\tau_i}} = \frac{CT_i}{Q_{\tau_i}} \tag{9}$$

In this work, an active redundancy-based FT technique is used for increasing task reliability [29]. With this technique, by replicating Task τ_i for r_i times, using $1 - out - of - r_i$ scheme, the unreliability of the fault-tolerant task $\tau_{ft,i}$ is given by [30]:

$$Q_{\tau_{ft,i}} = 1 - R_{\tau_{ft,i}} = 1 - \sum_{k=1}^{r_i} \binom{r_i}{k} (1 - Q_{\tau_i})^k (Q_{\tau_i})^{r_i - k} \tag{10}$$

Hence the unreliability of task graph TG is obtained as:

$$Q_{TG} = 1 - R_{TG} = 1 - \prod_{\tau_{ft,k} \in TG} \left(1 - Q_{\tau_{ft,k}}\right) \tag{11}$$

where R_{TG} is reliability of the task graph after applying FT techniques. Finally MTTF of the task graph is easily calculated as:

$$MTTF_{TG} = \frac{MS_{TG}}{Q_{TG}} \tag{12}$$

where MS_{TG} is the makespan of TG.

D. Scheduling Model

To manage the execution of hardware task graphs, since list-based scheduling techniques need little computation time and provide sub-optimal but good schedules [7], the proposed solution uses a non-preemptible *As Soon As Possible (ASAP)* scheduler. Thus, first of all, the *Starting Configuration Time* ($SCT(\tau_i)$) of Task τ_i is determined taking into account the tasks' precedence constraints as follows:

$$SCT(\tau_i) = max\left\{AVL(RC,\ \tau_i),\ \max_{j \in PR(\tau_i)} \{FET(\tau_j)\}\right\} \tag{13}$$

where $AVL(RC, \tau_i)$ indicates the earliest time, after the last task configuration, that the reconfigurable computer RC is available and has free area to place all replicas of Task τ_i, $PR(\tau_i)$ stands for the set of predecessors of τ_i, and $FET(\tau_j)$ is the *Finishing Execution Time* of τ_j. As an active redundancy-based FT technique is used, and also the scheduling strategy is not preemptible, FET of task τ_i is obtained as:

$$FET(\tau_i) = SET(\tau_i) + CT_i \tag{14}$$

in which $SET(\tau_i)$ denotes the *Starting Execution Time* of τ_i ($SET(\tau_i)$). Since Task τ_i can start its execution once it is configured, we have:

$$SET(\tau_i) = FCT(\tau_i) \tag{15}$$

where $FCT(\tau_i)$ is *Finishing Configuration Time* of τ_i. As reconfigurations are carried out in a serial manner, $FCT(\tau_i)$ is calculated as:

$$FCT(\tau_i) = SCT(\tau_i) + (CD_i \times r_i) \tag{16}$$

where CD_i and r_i denote configuration delay and the number of redundancies of Task τ_i, respectively. Finally, the makespan of the task graph is easily obtained as:

$$MS_{TG} = \max_{\tau_i \in TG} FET(\tau_i) \tag{17}$$

As it has been illustrated through Equations 13-16, in order to allocate a new task in the reconfigurable device, a partial reconfiguration process is required prior to its execution. This is a time-consuming process, especially in case of replicated tasks. This problem can be alleviated by applying a well-known technique named *task prefetch* [15], [24], which consists in configuring the tasks before they are ready to be executed, in such a way that their configuration delay overlaps with the execution time of precedent tasks. This technique leads to achieve a better makespan which gives the opportunity of using more FT techniques.

Thus, by taking task prefetch into account, $SCT(\tau_i)$ and $SET(\tau_i)$ are reformulated as:

$$SCT(\tau_i) = AVL(RC,\ \tau_i) \tag{18}$$

$$SET(\tau_i) = max\left\{FCT(\tau_i),\ \max_{j \in PR(\tau_i)} \{FET(\tau_j)\}\right\} \tag{19}$$

$FCT(\tau_i)$, $FET(\tau_i)$ and MS_{TG} remain unchanged.

These ideas have been gathered in Algorithm 1. It determines the starting and finishing configuration and execution

Algorithm 1 Configuration prefetch-aware ASAP scheduling strategy algorithm

1: **input** TG *//Task graph*
2: **input** RC *//Reconfigurable computer,* $RC = (RO, CO, GS, CD)$
3: **input** $Prefetch$ *//Boolean, indicating task prefetch*
4: $\tau s = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *//Sort tasks of TG based on ASAP strategy*
5: **while** $(\tau s \neq \emptyset)$ **do**
6: $\quad \tau_i = FirstTask\,(\tau s)$
7: $\quad$ **if** $Prefetch$ **then**
8: $\quad\quad SCT\,(\tau_i) = AVL\,(RC,\ \tau_i)$
9: $\quad\quad FCT\,(\tau_i) = SCT\,(\tau_i) + (CD_i \times r_i)$
10: $\quad\quad SET\,(\tau_i) = max\{FCT\,(\tau_i)\,,\ \max_{j \in PR(\tau_i)}\{FET\,(\tau_j)\}\}$
11: $\quad$ **else**
12: $\quad\quad SCT\,(\tau_i) = max\{AVL\,(RC,\ \tau_i)\,,\ max_{j \in PR(\tau_i)}\{FET\,(\tau_j)\}\}$
13: $\quad\quad SET\,(\tau_i) = FCT\,(\tau_i) = SCT\,(\tau_i) + (CD_i \times r_i)$
14: $\quad$ **end if**
15: $\quad FET\,(\tau_i) = SET\,(\tau_i) + CT_i$
16: $\quad \tau s = \tau s \,/\, \tau_i$
17: **end while**
18: **return** $\max_{\tau_i \in \tau s}\{FET\,(\tau_i)\}$

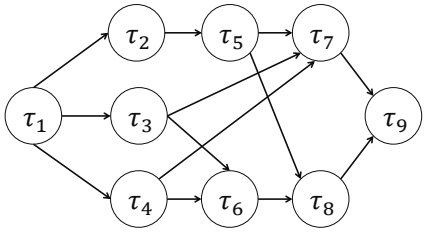


Fig. 1. A sample task graph

times of the hardware tasks scheduled with the prefetch-aware ASAP scheduling strategy. The input parameter $Prefetch$ indicates whether task prefetch will be applied or not.

In Algorithm 1, τs stores the set of unscheduled tasks, sorted by the ASAP strategy. The values of *SCT*, *FCT*, and *SET* of the tasks have been calculated in Lines 8-10, assuming task prefetch. The same values for the case of having no task prefetch have been obtained in Lines 12-13. Then *FET* of tasks has been calculated in Line 15. Finally, the returned value is the makespan of the task graph (Line 18).

III. Motivational Example

To clarify the proposed models and solutions, an hardware task graph with 9 tasks has been depicted in Figure 1. The characteristics of the tasks have been tabulated in Table I and those of the reconfigurable computer have been extracted from the Xilinx^{TM} Virtex-5 XUPV5LX110T FPGA. Details of the reconfigurable computer, and the measures of soft error rates (SERs) are further elaborated in Subsection V-A. In addition the unreliability and MTTF of the tasks in Table I have been obtained from Eq. (5) and Eq. (9) respectively. On the other hand Eq. (12) and Eq. (17) have been used to obtain the MTTF and the makespan of the task graph, respectively.

As a first example, the presented task graph has been scheduled by the ASAP strategy on the aforementioned reconfigurable computer without any FT techniques (Figure 2). Then, in Figure 3 the same task graph featuring a Duplication With Compare (DWC) FT technique is displayed. As Figure

TABLE I
Characteristics of the task graph depicted in Figure 1

Task	Comp. Time (*ms*)	CLB Count	Config. Delay (*ms*)	Unreliability	MTTF (*ms*)
τ_1	121	242	46	9.08×10^{-4}	1.33×10^{5}
τ_2	21	59	11	3.64×10^{-5}	5.77×10^{5}
τ_3	151	120	22	5.23×10^{-4}	2.89×10^{5}
τ_4	319	189	36	1.84×10^{-3}	1.73×10^{5}
τ_5	361	588	106	6.23×10^{-3}	5.79×10^{4}
τ_6	274	653	117	5.21×10^{-3}	5.26×10^{4}
τ_7	82	215	39	5.21×10^{-4}	1.57×10^{5}
τ_8	96	409	75	1.16×10^{-3}	8.25×10^{4}
τ_9	200	344	64	2.08×10^{-3}	9.63×10^{4}

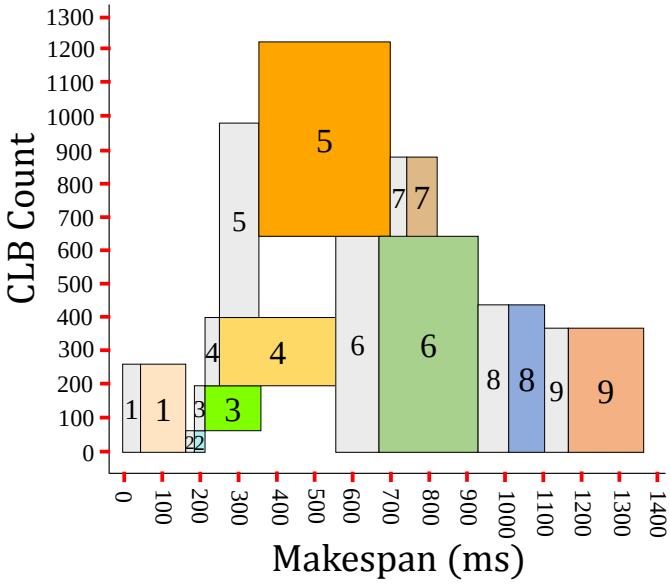


Fig. 2. Schedule of the sample task graph without any FT techniques

2 shows, the makespan of this task graph is 1381 *ms*, which leads to a MTTF of 7.51×10^4 *ms*. By using DWC (Figure 3), the MTTF is improved to 2.29×10^7 *ms*, but the makespan is increased to 1752 *ms*. As these two figures show, each task requires some reconfiguration delay before execution, which is in gray color in the figures.

Figure 3 has shown that applying fixed FT techniques to all the tasks has positive effects on MTTF, but a negative impact on makespan of the whole task graph. The methodology presented in this paper aims at selecting different redundancy-based FT techniques for the tasks, in order to find solutions that improve MTTF, but at the cost of paying different (and yet acceptable) levels of makespan degradation. It is noteworthy to state that, since MTTF is an indicator of reliability, and following Eq. (12) it depends on both reliability and makespan, MTTF can be used to show reliability improvement only when the makespan does not change. The following examples will illustrate the presented methodology and will show the potential benefits that can be obtained by using it.

To illustrate a case of MTTF improvement without deteriorating makespan, selective FT techniques have been chosen for the tasks of Figure 2 and the result has been depicted

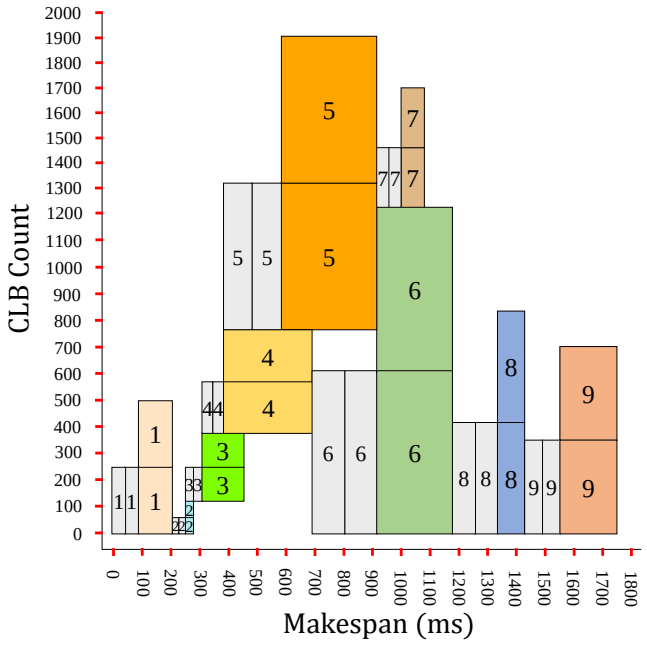


Fig. 3. Schedule of the sample task graph with duplicated tasks. A great reliability improvement is achieved, at the cost of deteriorating the makespan

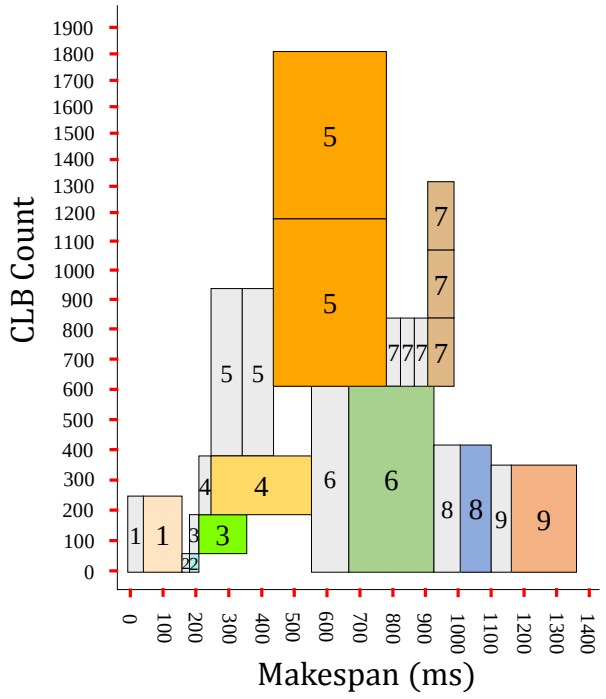


Fig. 4. Applying selective FT techniques to schedule of Figure 2. A good reliability improvement is achieved, but without deteriorating makespan

in Figure 4. This solution has the same makespan of Figure 2 (1381 *ms*), but as tasks τ_5 and τ_7 use double and triple redundancy respectively, following Eq. (12), the MTTF of the task graph is increased from 7.51×10^4 *ms* to 1.18×10^5 *ms*.

These kind of redundancy-based FT techniques have been widely studied in the literature. Thus, on one hand, triple redundancy, also known as Triple Modular Redundancy (TMR) or n-redundancy are well-known solutions to increase the

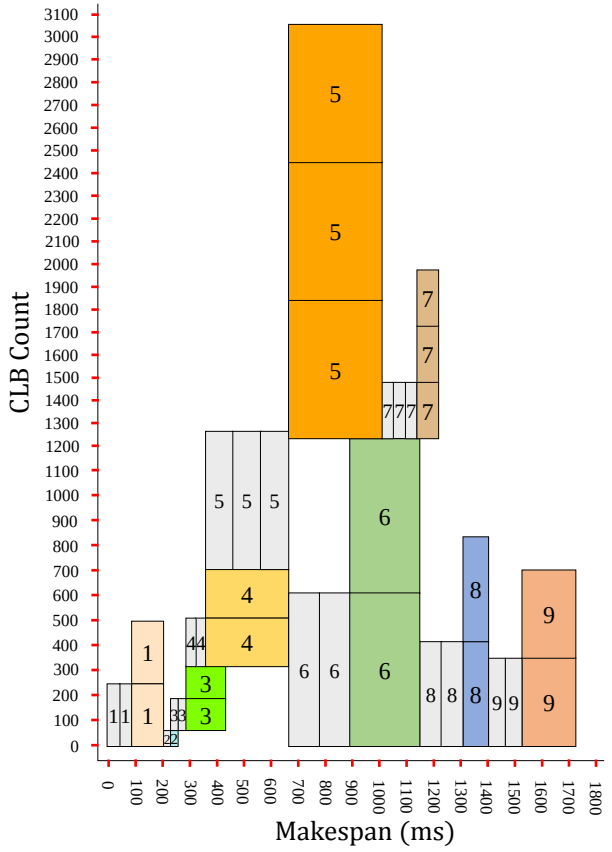


Fig. 5. Applying selective FT techniques to schedule of Figure 3. A good makespan improvement is achieved, but without deteriorating reliability

reliability of embedded systems, and it consists of instantiating n copies of a task and then, voting the correct output. Since this can be implemented in multiple ways, such as adding an additional voting circuit [31] or in a time multiplexed manner [32] (with different costs), in order not to lose generality, the approach in this paper will assume that the cost of n-redundancy is the hardware cost of the $n-1$ additional replicas of the task. On the other hand, double redundancy, also known as DWC, has also been investigated as an alternative to TMR since it reduces the area overhead of triplication [33], [34].

Finally, the example of Figure 5 achieves a reliability improvement and a makespan degradation between those of Figure 3 and Figure 4. In this case, Tasks τ_5 and τ_7 are triplicated, whereas Tasks τ_1, τ_3, τ_4, τ_6, τ_8 and τ_9 are duplicated. Thus, in this example not only reliability improves, but also the makespan decreases from 1752 *ms* to 1741 *ms* in comparison with Figure 3. These examples have illustrated how selective redundancy-based FT techniques can be used in different manners to obtain a solution that satisfies the users' needs. Thus, some solutions can be acceptable for hard real-time systems, whereas others can be used to improve the Quality-of-Service (QoS) of the applications.

For the sake of simplicity, all these examples have been obtained assuming a scheduler that does not implement configuration prefetch. This technique makes possible to achieve even better results, as it will be discussed in Section V.

IV. The Proposed Methodology

A. Problem Formulation

The proposed solution aims at selecting different FT techniques for hardware tasks, such that, for each one of these combinations, the reliability of the task graph is maximized whereas its makespan is degraded as less as possible. The explored solutions range between applying no FT techniques to the tasks (minimum reliability and makespan) and applying N_{Max} redundancy to all of them (maximum reliability and makespan), being N_{Max} a fixed parameter known in advance. The redundancy level of Task τ_i is then a *Decision Variable* of the optimization problem and, in this paper, it is denoted as r_i. These variables take integer values indicating different FT techniques. For instance, $r_i = 1$ indicates no FT technique for τ_i, $r_i = 2$ denotes DWC and $r_i = 3$ devotes to TMR.

As there are two optimization objectives (reliability and makespan), a multi-objective optimization approach is used to improve both objectives while respecting the task order and device area constraints (the task order is determined in advance by the underlying scheduling algorithm). Hence the optimization problem is formulated as:

$$\begin{aligned} &\text{Maximize } R_{TG_{ft}} \\ &\text{Minimize } MS_{TG_{ft}} \\ &\text{Subject to} \\ &\sum_{\tau_k \in Running\ Tasks} ColumnCount_{\tau_k} \times r_k \leq CO \\ &r_k \in \left\{1, 2, \ldots, \left\lfloor \frac{CO}{ColumnCount_{\tau_k}} \right\rfloor \right\} \end{aligned} \tag{20}$$

When dealing with optimization problems, it is easy and straightforward to find an optimal solution in single-objective optimization problems, so a decision variable vector that minimizes (or maximizes) the sole objective function and holds the constraints is considered as the optimization solution. As the introduced problem is multi-objective, the concept of optimality of single objective approaches cannot be directly used, therefore a classification of the solutions is presented in terms of Pareto optimality [35]. In the general case, if there are k decision variables and b objective functions, we have:

- $X = (r_1, r_2, \ldots, r_k)$ is a vector of decision variables.
- $f(X) = \{f_1(X), f_2(X), \ldots, f_b(X)\}$ is a vector of objective functions.

In terms of minimization, the following definitions of Pareto optimality exist [36]:

1) **Definition 1** (Pareto Optimal): A solution vector $x^* \in X$ is a Pareto optimal solution if there is not any other point $x \in X$ such that $f_t(x) \leq f_t(x^*)$ for all $t = 1,\ 2, \ldots, b$ and $f_s(x) < f_s(x^*)$ for at least one s. Such solution is called *true Pareto optimal* solution.
2) **Definition 2** (Pareto Dominance): A dominance x, dominates y (denoted as $x \succ y$) iff $f_t(x) \leq f_t(y)\ and\ \exists q\ s.t.\ f_q(x) < f_q(y); t, q \in \{1,\ 2, \ldots, b\}$; otherwise x and y are not better than each other. If there are no solutions that dominate x, then x is a *non-dominated solution*.
3) **Definition 3** (Pareto Set): A set of all non-dominated solutions $\{x^* | \nexists x : x \succ x^*\}$ is called a Pareto set.
4) **Definition 4** (Pareto Front): The set of vectors in the objective space that are pictures of elements of a Pareto set, i.e. $\{f(x^*) | \nexists x : x \succ x^*\}$.

The solution that the proposed methodology returns is selected as the one among solutions of the Pareto set that best complies with the user's needs. This is indicated by means of an input weight named α, $0 \leq \alpha \leq 1$. $\alpha = 1$ stands for maximum reliability optimization, whereas $\alpha = 0$ means minimum makespan degradation. Any value between 0 and 1 indicates an intermediate solution between these two ends.

B. Outline of the Proposed Solution

Based on the problem formulation and the aforementioned definitions, the proposed Pareto-based technique, applies a multi-objective optimization algorithm in order to obtain the Pareto set of solutions. There are many approaches to convert a multi-objective optimization problem to a single-objective one. For example, *ε-Constraint Approach* and *Weighted-Sum Approach* are two commonly used classical methods [35]. The *Weighted-Sum Approach* converts a set of objectives into a single one by multiplying each objective to the user-defined weights. In spite that the latter approach is straightforward, it has some drawbacks, e.g. choosing weight vectors uniformly does not necessarily lead to a uniform set of Pareto optimal solutions. For this reason, the *ε-Constraint Approach* has been used to implement the proposed methodology. Thus, the multi-objective optimization problem has been re-formulated by holding one of the objectives and restricting the remaining one as constraints. The general body of this approach is as follows:

$$\begin{aligned} &\text{Minimize } f_\mu(X) \\ &\text{Subject to} \\ &\quad f_i(X) \leq \varepsilon_i, \quad i = 1,\ 2,\ \ldots, b\ and\ i \neq \mu \\ &\quad Other\ Constraints \ldots \end{aligned} \tag{21}$$

In this formulation, the parameter ε_i denotes an upper bound of the value of function f_i. The pseudo-code of the proposed methodology has been depicted in Algorithm 2. It is important to note that, since the underlying scheduling strategy (ASAP) is sub-optimal, the solution that this methodology returns is sub-optimal as well.

In the presented technique, reliability is selected as the single objective of the optimization problem and makespan is considered as a constraint. It receives as inputs the task graph TG, the reconfigurable computer RC, and the α parameter discussed before. Then, the minimum and maximum values of the makespan are obtained in Lines 7 and 8, respectively, in order to find its lower and upper bounds. MS_{min} is obtained by scheduling the input task graph without any FT techniques and, in a similar way, MS_{max} is obtained by applying the maximum redundancy level to the tasks (N_{Max}).

In each iteration of the loop in Lines 10-19, a single objective optimization problem (defined in lines 11-16) with different constraints is solved, and the obtained result is stored

Algorithm 2 Reliability and Makespan Optimization Algorithm

1: **input** TG *//Task graph*
2: **input** RC *//Reconfigurable computer,* $RC = (RO,\ CO,\ GS,\ CD)$
3: **input** α *//Degree of user's preference for reliability improvement and makespan degradation*
4: $DV \leftarrow$ Decision variables set $(r_1,\ r_2, \ldots,\ r_n)$*//initially* $DV = \emptyset$
5: $PS \leftarrow$ Pareto set, initially $PS = \emptyset$
6: $x^* \leftarrow$ An optimization solution, initially $x^* = \emptyset$
7: $MS_{min} \leftarrow$ **Schedule**$(TG_{noFT},\ RC)$
8: $MS_{max} \leftarrow$ **Schedule**$(TG_{N_{Max}FT},\ RC)$
9: $\varepsilon_i = MS_{max}$
10: **while** $(\varepsilon_i \geq MS_{min})$ **do**
11: *//DV = solution of the following single objective optimization problem*
12: $Maximize\ R_{TG_{ft}}$
12: Subject to
13: $MS_{TG_{ft}} \leq \varepsilon_i$
14: $r_k \geq 1$
15: $r_k \leq \left\lfloor \frac{CO}{ColumnCount_{\tau_k}} \right\rfloor$
16: $\sum_{\tau_k \in Running\ Tasks} ColumnCount_{\tau_k} \times r_k \leq CO$
17: $\varepsilon_i = MS_{TG_{ft}}(DV) - 1$
18: $PS \leftarrow$ **CompareParetoSets**$(PS,\ DV)$
19: **end while**
20: $x^* \leftarrow$ **SelectPreferredSolution**$(PS,\ \alpha)$
21: **return** x^*

in DV. After solving each problem, the **CompareParetoSets** function updates the *Pareto set*, based on *Definitions 1-3*, by comparing the previously found non-dominated solutions with the current decision variable set (DV). Finally, the most desired user's solution is selected by the **SelectPreferred-Solution** function (Line 20). This function uses the α input parameter to select the solution $x^* \in Pareto\ Set$ such that:

$$R(x^*) = max\{R(\mathrm{x})\,;x \in Pareto\ Set\ \&$$

$$R(x) \leq (R_{Min} + ((R_{Max} - R_{Min}) \times \alpha))\} \qquad (22)$$

where $R(x^*)$ indicates the reliability of solution x^*, and R_{Min} and R_{Max} denote the minimum and maximum reliability of solutions, indicated in the Pareto front, respectively. The complexity of Algorithm 2 is governed by that of the optimization algorithm used to solve the problem presented in Lines 11-16. Since the decision variables are the redundancy level of tasks (r_i) and they take integer values, the presented optimization problem is integer nonlinear. It has been proven that there is no polynomial-time optimal solution for optimization problems with integer variables [37]. There exist many techniques to solve the presented optimization problem, such as *Outer approximation*, *Generalized Benders decomposition*, *Extended cutting-plane method, Branch and bound*, and *Branch and cut*. In the worst-case scenario, these methods take an exponential number of iterations, but the properties of practical problems are such that in practice these optimization methods often work efficiently [38]. The experiments that have been carried out (which are described in the next section) showed that the proposed Pareto-based approach worked efficiently for the task graphs that were evaluated. Thus, on average it requires totally 688 *ms* to generate the Pareto set of solutions. The experiments have been run on a computer with an Intel® Core i5 2.4GHz processor and a Windows 8.1 operating system.

V. Experimental Results

A. Experimental Setup

This section presents experimental results on realistic task graphs in order to evaluate the proposed Pareto-based approach. In these experiments, the task graphs have been generated using *P-Method* [39], which is based upon the probabilistic construction of a task graph adjacency matrix. In order to generate appropriate task graphs for the target reconfigurable computer, some changes have been made to the P-Method, so that the parameters introduced in Eq. (1) are considered in the task graph generation.

In the experiments, a realistic partial reconfigurable computer is modeled and several real-world inspired task graphs are randomly generated to be emulated on it. In this case, the Xilinx™ Virtex-5 XUPV5LX110T FPGA [40] has been modeled, which features 160 rows and 54 columns of CLBs. In this device, each CLB group has 20 CLBs and there are 8 CLB groups per column. According to experimental measurements made by our research group, each CLB group takes 3.53 *ms* to be configured. Therefore $RC = (160, 54, 20, 3.53)$.

The evaluated task graphs contain 10 tasks (to mimic H.264 and MP3 task graphs [24]), each of which is attributed with realistic characteristics. The computation times are selected between [10...500] *ms* [41]. Each task has a width and height in the range of [7...42] CLBs to model hardware tasks between 49 and 1764 CLBs, such as UART reconfigurable core and the Discrete Wavelet Transform reconfigurable core [42].

Task size TS depends on the number of occupied CLB groups. For the Xilinx™ Virtex-5 XUPV5LX110T, each CLB group contains 36 frames and each frame has 1280 bits. Sensitive bits are stated as a percentage of the task size. To mimic real-world tasks, this parameter takes values ranging between 7% [43] to upmost 35% [44] of the task size.

Based on the values of soft errors reported in [26], for difference altitudes of harsh environments, SERs take values ranging between 1.0×10^{-7} to 5.0×10^{-3} upsets *per bit per day*. By having a task graph and the SER, MTTF of tasks and task graphs before and after applying FT techniques can be easily obtained from Eqs. (9) and (12). The results are presented as an average value of 100 independent runs of different task graphs. The positive impacts of applying the presented methodology are demonstrated in the next subsections.

B. MTTF Improvement

In the first experiment, the task graphs have been scheduled without any FT techniques in order to obtain their MTTF and makespan. Then, Algorithm 2 has been applied with and without considering task prefetch (illustrated in Algorithm 1). From all the obtained points of the Pareto set, the solutions with a makespan equal to the case with no FT technique but a higher MTTF has been selected.

Different task characteristics have different impacts on MTTF of the task graph. In this experiment, the effects of task CLB count and task computation time on the MTTF improvement are examined. It is noteworthy to state that, since

TABLE II
RELIABILITY IMPROVEMENT ACHIEVED FOR TASK GRAPHS WHOSE TASKS FEATURE DIFFERENT CLB COUNTS (FPGA SIZE: 8640 CLBS)

		Without Task Prefetch		**With Task Prefetch**	
# CLBs (Size Ratio)	Basic MTTF	Optimal MTTF	MTTF Improve	Optimal MTTF	MTTF Improve
50 (0.006)	8.26×10^5	1.02×10^6	+23.07%	7.85×10^6	+849.76%
100 (0.012)	5.20×10^5	6.25×10^5	+20.10%	4.36×10^6	+739.02%
200 (0.023)	2.73×10^5	3.32×10^5	+21.67%	1.76×10^6	+545.15%
500 (0.057)	1.39×10^5	1.55×10^5	+12.08%	4.67×10^5	+236.61%
1000 (0.116)	8.84×10^4	9.65×10^4	+9.15%	1.17×10^5	+31.99%
1500 (0.174)	7.26×10^4	7.77×10^4	+7.06%	8.13×10^4	+11.91%

TABLE III
THE RELIABILITY IMPROVEMENT ACHIEVED FOR TASK GRAPHS WHOSE TASKS FEATURE DIFFERENT COMPUTATION TIMES

		Without Task Prefetch		**With Task Prefetch**	
Cmpt. Time	Basic MTTF	Optimal MTTF	MTTF Improve	Optimal MTTF	MTTF Improve
50	2.86×10^5	2.91×10^5	+1.65%	2.91×10^5	+1.85%
100	1.88×10^5	1.94×10^5	+3.18%	1.96×10^5	+4.17%
200	1.19×10^5	1.25×10^5	+4.93%	1.40×10^5	+17.44%
300	9.72×10^4	1.05×10^5	+7.79%	1.59×10^5	+63.29%
400	8.80×10^4	9.62×10^4	+9.33%	2.34×10^5	+166.35%
500	8.15×10^4	9.16×10^4	+12.35%	3.67×10^5	+350.56%

task size and configuration delay depends on task CLB count, their effects are not experimented separately.

Task graphs whose tasks feature sizes ranging from 50 to 1500 CLBs have been evaluated (Table II). This table also presents the tasks' *Size Ratio,* which categorizes different relative task sizes with respect to the device CLB count (8640 CLBs). Thus, by applying the proposed solution and when the scheduler does not apply task configuration prefetch, without deteriorating makespan, the MTTF improvement varies from 7.06% to 23.07% (Columns 3-4). Applying prefetch yields even better results (Columns 5-6). In this case, MTTF improvement varies from 11.91% to 849.76% which is, on average, 26 times higher. Thus, by applying the proposed solution, the makespan of task graphs could be held constant while its MTTF is substantially improved. In addition, this improvement is more significant for tasks with lower size ratio.

Another experiment has been performed in which task computation time changes from 50 *ms* to 500 *ms* (Table III). This table shows moderate improvements when no task prefetch is used, but very good ones when using this technique on the scheduler, especially for large task computation times. Although these MTTF improvements are achieved without deteriorating makespan, the experiments show that the configuration delay is a serious limitation to the MTTF improvement and when the configuration delay decreases due to task prefetch, the MTTF improvement is more significant. The experiments with task prefetch show a MTTF improvement that is, on average, 15.38 times higher than their counterparts without using task prefetch.

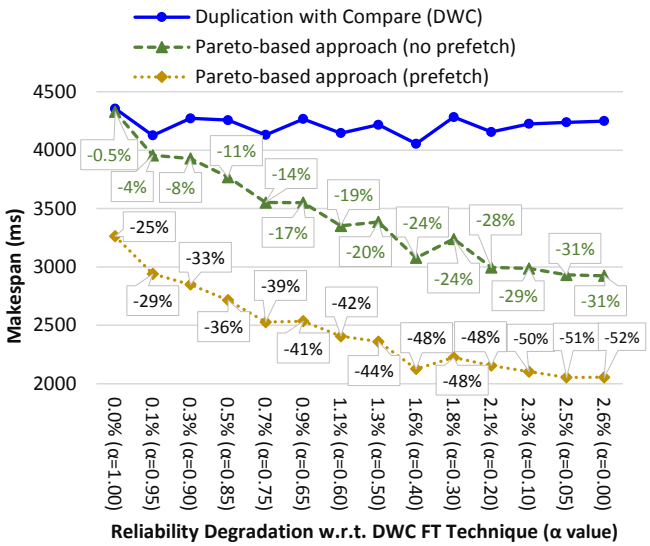


Fig. 6. Comparison of MTTF and makespan improvement with the fixed FT technique DWC

C. Comparison with fixed FT technique: DWC

In the second experiment (Figure 6), all the task graphs have been enhanced with the DWC FT technique [45] and then their reliability and makespan have been compared with the proposed technique. The objective is to observe how the Pareto-based solutions, which feature different trade-offs between reliability improvement and makespan degradation for different values of the α parameter, outperform DWC. Similarly as in previous experiments, two cases have been examined: scheduling with and without task prefetch.

As Figure 6 shows, by decreasing α from 1 to 0, the reliability is degraded from 0% to 2.6%. The more the reliability is degraded with respect to DWC (lower α value), the better makespan improvement is achieved and this improvement is more significant when task prefetch is applied. With no prefetch, the makespan improvement ranges from 27 to 1325 *ms* (from 0.5% to 31%) whereas by applying task prefetch, a makespan improvement from 1094 to 2195 *ms* is achieved (from 25% to 52%). This experiment clearly shows how different degrees of makespan improvement can be achieved based on the user's preferences.

D. Comparison with Adaptive Fault Tolerance

Recently Jacobs et al. in [10] have introduced a comprehensive framework for adaptive fault tolerance in space computing and SER-varying environments. Their approach, named "*three-mode adaptive strategy*", employs different FT techniques for different ranges of SERs, and in each SER, a fixed FT technique is used for all the tasks. In this approach, single tasks are executed when the SER is lower than 10% of the expected fault rates, triplicated tasks are executed when the SER is above 50% of the expected fault rates, and tasks are duplicated otherwise. This technique assumes that all the running tasks fit in the reconfigurable computer and hence it does not implement any scheduling strategy.

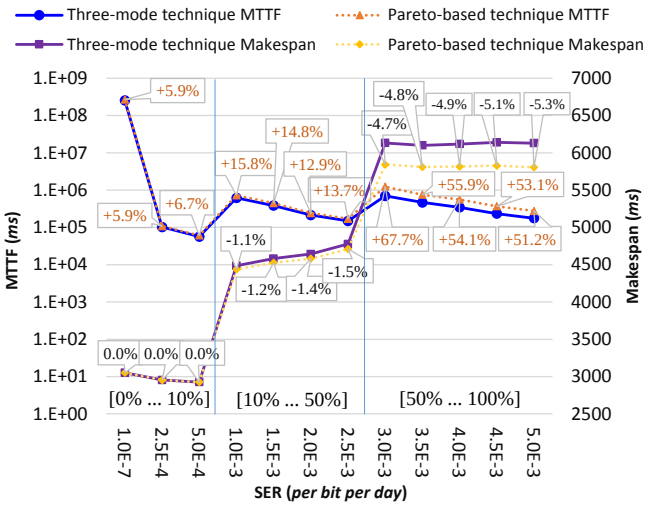


Fig. 7. Comparison of MTTF and makespan improvement with an adaptive FT approach [10]. Note that the left y-axis scale is logarithmic.

To make a fair comparison with Jacobs's strategy, our ASAP scheduling technique without task prefetch has been added to the Jacobs' approach. To do the comparison, after applying the proposed Pareto-based technique to the input task graphs, two solutions have been chosen: 1) The one that most improves the MTTF without deteriorating makespan, and 2) The one that most improves makespan without degrading reliability. The results are illustrated in Figure 7. As this figure shows, in the first SER range ($0\% \rightarrow 10\%$), Jacobs uses a single module technique, therefore even by using our approach, the makespan could not be improved, but the MTTF can be improved easily. For higher SERs, both makespan and MTTF are improved without degrading the other metric. In addition, within the same range, the MTTF and makespan improvements are more significant as the SER increases. The obtained results demonstrate that, although adaptive FT techniques in SER-varying environments have positive impact on the system performance, our technique achieves even better results.

VI. Conclusions and Future Work

This paper has addressed the problem of reliability and makespan optimization of hardware task graphs in reconfigurable platforms, by applying redundancy-based FT techniques to the running tasks. The proposed problem has been formulated as a multi-objective integer nonlinear optimization problem, which uses an ASAP-based scheduler with task configuration prefetch to steer the execution of the tasks in the device. It involves the exploration of the solutions space in order to find a set of solutions that offer different trade-offs between reliability improvement and makespan degradation.

Experimental results have demonstrated the positive effects of the proposed technique. Experiments have been performed on real-world-inspired hardware task graphs. Thus, by using the proposed technique, the reliability of task graphs can be improved without any makespan degradation, and vice-versa. Applying task prefetch to the underlying scheduling strategy leads to even more significant improvements. The presented technique also outperforms the adaptive FT strategy described in [10], for fault-varying environments.

For future work, we want to extend this approach by adding a placement-aware optimization approach and to study different reconfiguration area models.

References

[1] S. Engelen, E. Gill, and C. Verhoeven, "On the reliability, availability, and throughput of satellite swarms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 2, pp. 1027–1037, 2014.

[2] P. S. Ostler, M. P. Caffrey, D. S. Gibelyou, P. S. Graham, K. S. Morgan, B. H. Pratt, H. M. Quinn, and M. J. Wirthlin, "SRAM FPGA reliability analysis for harsh radiation environments," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3519–3526, 2009.

[3] Y. Zhang and J. Jiang, "Fault tolerant control system design with explicit consideration of performance degradation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 3, pp. 838–848, 2003.

[4] J. Gray and T. Kean, "Configurable hardware: a new paradigm for computation," in *Proc. Dicennial CalTech Conference*, 1989.

[5] T. Kean and I. Buchanan, "The use of FPGAs in a novel computing subsystem," in *Proc. 1st Intl. ACM/SIGDA Workshop on FPGAs*, 1992.

[6] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[7] J. A. Clemente, J. Resano, C. González, and D. Mozos, "A hardware implementation of a run-time scheduler for reconfigurable systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1263–1276, 2011.

[8] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 37, 2015.

[9] M. Niknahad, O. Sander, and J. Becker, "Fine grain fault tolerancea key to high reliability for fpgas in space," in *Aerospace Conference, 2012 IEEE*. IEEE, 2012, pp. 1–10.

[10] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 21, 2012.

[11] B. Pratt, M. Fuller, M. Rice, and M. Wirthlin, "Reduced-precision redundancy for reliable FPGA communications systems in high-radiation environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 369–380, 2013.

[12] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for Xilinx Virtex FPGA configuration memory scrubbers," *IEEE Transactions on Nuclear Science*, vol. 60, no. 1, pp. 376–385, 2013.

[13] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, 2007.

[14] K. Xia, A. Shen, Y. Fu, T. Liu, and J. Jiang, "A high performance DSP system with fault tolerant for space missions," in *CCF National Conference on Computer Engineering and Technology*. Springer, 2012, pp. 111–120.

[15] J. A. Clemente, J. Resano, and D. Mozos, "An Approach to Manage Reconfigurations and Reduce Area Cost in Hard Real-Time Reconfigurable Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 90, 2014.

[16] J.-Y. Yin, G.-C. Guo, and Y.-X. Wu, "A hybrid fault-tolerant scheduling algorithm of periodic and aperiodic real-time tasks to partially reconfigurable FPGAs," in *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*. IEEE, 2009, pp. 1–5.

[17] J. Yin, B. Zheng, and Z. Sun, "A hybrid real-time fault-tolerant scheduling algorithm for partial reconfigurable system," *Journal of Computers*, vol. 7, no. 11, pp. 2773–2780, 2012.

[18] R. Ramezani and Y. Sedaghat, "Scheduling periodic real-time hardware tasks on dynamic partial reconfigurable devices subject to fault tolerance," in *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*. IEEE, 2014, pp. 479–484.

[19] M. Naghibzadeh, "Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud," *Future Generation Computer Systems*, pp. 33 – 45, 2016.

[20] F. Say and C. F. Bazlamaçcı, "A reconfigurable computing platform for real time embedded applications," *Microprocessors and Microsystems*, vol. 36, no. 1, pp. 13–32, 2012.

[21] I. Herrera-Alzu and M. Lopez-Vallejo, "System design framework and methodology for Xilinx Virtex FPGA configuration scrubbers," *IEEE Transactions on Nuclear Science*, vol. 61, no. 1, pp. 619–629, 2014.

[22] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2147–2157, 2003.

[23] G. Liu, Y. Zeng, D. Li, and Y. Chen, "Schedule length and reliability-oriented multi-objective scheduling for distributed computing," *Soft Computing*, vol. 19, no. 6, pp. 1727–1737, 2015.

[24] J. A. Clemente, I. Beretta, V. Rana, D. Atienza, and D. Sciuto, "A Mapping-Scheduling Algorithm for Hardware Acceleration on Reconfigurable Platforms," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 9, 2014.

[25] N. Guan, Q. Deng, Z. Gu, W. Xu, and G. Yu, "Schedulability analysis of preemptive and nonpreemptive EDF on partial runtime-reconfigurable FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 4, p. 56, 2008.

[26] J. Zhang, Y. Guan, and C. Mao, "Optimal partial reconfiguration for permanent fault recovery on SRAM-based FPGAs in space mission," *Advances in Mechanical Engineering*, vol. 5, p. 783673, 2013.

[27] R. Ramezani, J. A. Clemente, Y. Sedaghat, and H. Mecha, "Estimation of Hardware Task Reliability on Partially Reconfigurable FPGAs," in *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*. IEEE, 2016, pp. 1–4.

[28] A. J. Tylka, J. Adams, P. R. Boberg, B. Brownstein, W. F. Dietrich, E. O. Flueckiger, E. L. Petersen, M. A. Shea, D. F. Smart, and E. C. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, 1997.

[29] R. Ramezani and Y. Sedaghat, "An overview of fault tolerance techniques for real-time operating systems," in *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*. IEEE, 2013, pp. 1–6.

[30] M. L. Shooman, *Reliability of computer systems and networks: fault tolerance, analysis, and design*. John Wiley & Sons, 2003.

[31] H. Quinn, P. Graham, and B. Pratt, "An automated approach to estimating hardness assurance issues in triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3070–3076, 2008.

[32] X. She and K. McElvain, "Time multiplexed triple modular redundancy for single event upset mitigation," *IEEE Transactions on Nuclear Science*, vol. 4, no. 56, pp. 2443–2448, 2009.

[33] J.-P. Anderson, B. Nelson, and M. Wirthlin, "Using statistical models with duplication and compare for reduced cost FPGA reliability," in *Aerospace Conference, 2010 IEEE*. IEEE, 2010, pp. 1–8.

[34] H. Asadi and M. B. Tahoori, "Analytical techniques for soft error rate modeling and mitigation of FPGA-based designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 12, pp. 1320–1331, 2007.

[35] K. Deb, "Multi-objective optimization," in *Search methodologies*. Springer, 2014, pp. 403–449.

[36] H. Garg and S. Sharma, "Multi-objective reliability-redundancy allocation problem using particle swarm optimization," *Computers & Industrial Engineering*, vol. 64, no. 1, pp. 247–255, 2013.

[37] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[38] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," *Acta Numerica*, vol. 22, pp. 1–131, 2013.

[39] S. Al-Sharaeh and B. E. Wells, "A comparison of heuristics for list schedules using the Box-method and P-method for random digraph generation," in *System Theory, 1996., Proceedings of the Twenty-Eighth Southeastern Symposium on*. IEEE, 1996, pp. 467–471.

[40] Xilinx, "Virtex-5 FPGA Configuration User Guide, UG191(V3.10)," 2012.

[41] K. Danne and M. Platzner, "An EDF schedulability test for periodic tasks on reconfigurable hardware devices," in *ACM SIGPLAN Notices*, vol. 41, no. 7. ACM, 2006, pp. 93–102.

[42] C. Steiger, H. Walder, M. Platzner, and L. Thiele, "Online scheduling and placement of real-time tasks to partially reconfigurable devices," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 224–225.

[43] D. E. Johnson, K. S. Morgan, M. J. Wirthlin, M. P. Caffrey, and P. S. Graham, "Detection of configuration memory upsets causing persistent errors in SRAM-based FPGAs," 2004.

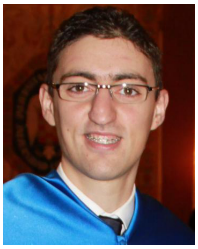
[44] J. S. Monson, M. Wirthlin, and B. Hutchings, "A fault injection analysis of Linux operating on an FPGA-embedded platform," *International Journal of Reconfigurable Computing*, vol. 2012, p. 7, 2012.

[45] D. Bozdag, F. Ozguner, and U. V. Catalyurek, "Compaction of schedules and a two-stage approach for duplication-based DAG scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857–871, 2009.

Reza Ramezani has received the B.S.c and M.Sc. degrees in computer engineering from Shiraz Faculty of Engineering and Isfahan University of Technology, Iran, in 2010 and 2012, respectively. He is currently a Ph.D. candidate of Software Engineering, with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran. Since 2014, he collaborates with the Departamento de Computadores y Automatica (DACyA), at the Universidad Complutense de Madrid (UCM), Madrid, Spain. His main research area includes reconfigurable computing, real-time systems, task scheduling, and fault-tolerant design.

Yasser Sedaghat received the M.Sc. and Ph.D. degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 2006 and 2011, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM), Mashhad, Iran. He has established and has been the chair of the Dependable Distributed Embedded Systems (DDEmS) Laboratory, FUM, since 2012. His current research interests include dependable embedded systems and networks, reliable software design, dependable embedded operating systems, and dependable FPGA-based designs.

Mahmoud Naghibzadeh has received the MS and PhD degrees in Computer Science and Computer Engineering in 1977 and 1980, respectively, from University of Southern California, USA. Now, he is a full professor at the Department of Computer Engineering and the director of the Knowledge Technology Laboratory, Ferdowsi University of Mashhad, Mashhad, Iran. His research interests include the scheduling aspects of real-time systems, Grid, Cloud, Multiprocessors, Multicores, and GPGPUs. Currently, he is the chief editor of the Computer and Knowledge Engineering (CKE) journal.



Juan Antonio Clemente received his computer science degree from Universidad Complutense de Madrid (UCM), Madrid, Spain, in 2007; and his Ph.D. in 2011. He is Assistant Professor and Researcher with the GHADIR research Group. Since 2012, he collaborates with the TIMA Laboratory, in Université Grenoble-Alpes, Grenoble, France.

His research interests are dynamically reconfigurable hardware, FPGA design and task scheduling. Also, his research is focused on the study of Single Event Effects (SEE) tolerance of digital circuits, and on conducting experiments evaluating the robustness of memories face to neutrons with TIMA Labs and the Laboratoire de Physique Subatomique et de Cosmologie (LPSC), at CNRS-IN2P3 research center.