The Journal of Supercomputing
https://doi.org/10.1007/s11227-019-02976-6

Hybrid scheduling to enhance reliability of real-time tasks running on reconfigurable devices

Abolfazl Ghavidel[1] · Yasser Sedaghat[1] · Mahmoud Naghibzadeh[1]

Abstract
Reconfigurable devices (RDs) are extremely advantageous when employed in real-time embedded systems. Nonetheless, they are susceptible to soft errors. In a broad sense, the present research addresses the challenge of improving the reliability of independent periodic real-time hardware tasks in RDs by utilizing hybrid fault-tolerant scheduling. The current paper combines static and dynamic real-time scheduling techniques to improve the reliability of the system. First, the proposed algorithm statically schedules primary tasks and preserves area and time for possible backup tasks on the RD. The overlapping of passive backup tasks is possible. Next, at the run time, event-triggered dispatcher dynamically determines which candidate backup copy should be selected for configuration on the overloaded preserved areas. Reliability, task deadline, and RD area limitations are the determining factors of backup overloading in the static phase. On the other hand, in the dynamic phase, the execution result of the primary tasks—in this case, success or failure—is the deciding factor based on which the dispatcher configures the true backup task on the preserved area. Experimental results show that the hybrid scheduling technique enhances the mean-time-to-failure of the system by an average factor of 1.22 in comparison with a similar state-of-the-art study.

Keywords Real-time scheduling · Fault tolerance · Mean-time-to-failure · Reconfigurable device

✉ Yasser Sedaghat
y_sedaghat@um.ac.ir

Abolfazl Ghavidel
ghavidel@mail.um.ac.ir

Mahmoud Naghibzadeh
naghibzadeh@um.ac.ir

[1] Dependable Distributed Embedded Systems (DDEmS) Laboratory, Department of Computer Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

Published online: 27 August 2019

1 Introduction

Reconfigurable devices (RDs) are general purpose hardware devices with the ability to be programmed and re-programmed over and over again [1]. Apart from flexibility, RDs eliminate the need to fetch, decode, and execute instructions since data are sent to the next stage on every clock cycle. Therefore, RDs offer the intrinsic feature of concurrency. This is especially the case for high-performance supercomputing applications that are usually implemented on general purpose multiprocessing computers/clusters. State-of-the-art RDs, currently, have been highly sophisticated that they can be employed for such applications which target both fine grain and coarse grain parallelism (called embedded supercomputing).

The most prominent, widespread, and well-known RDs are static memory-based (SRAM-based) field-programmable gate arrays (FPGAs) [2, 3]. Because of their reprogrammability and use of standard CMOS process technology, FPGAs provide a superior trade-off between the performance of dedicated hardware (e.g., application-specific integrated circuits) and flexibility [4, 5]. Moreover, FPGAs perform remarkably on stream processing and cryptographic problems [6, 7]. Such distinguishing features have led to the increased employment of SRAM-based FPGAs in broad applications as controllers and data processors, particularly in bioinformatics and safety critical embedded systems (e.g., avionics and space missions) [8, 9].

For configuring hardware tasks, SRAM-based FPGAs generally use an array of latches (called a configuration file or bitstream) to switch pass transistors ON or OFF. The bitstream contains SRAM bits to store configuration data for lookup tables, block RAMs, and flip-flops which occupy more than 95% of the FPGA area [8]. Similar to other electronic integrated circuits, SRAM-based FPGAs are well-known for their susceptibility to transient and intermittent faults[1] mainly caused by high-energy neutrons, protons, or heavy ions [10]. So-called single event effects (SEEs), such as single event upsets (SEUs), multiple bit upsets, and multiple cell upsets, affect bitstream memory cells by modifying their values [11] and thus possibly disrupt the normal operation of the FPGA. Hence, reliability and mean-time-to-failure (MTTF) are of particular concern to all critical embedded systems using SRAM-based FPGAs [8].

There are several well-known resilience techniques for FPGAs according to their technology, architecture, and operating environment. These generally fall into three categories [8]: (1) Fabrication process-based techniques, e.g., silicon-on-insulator, (2) design-based techniques, e.g., hardware/time redundancy, duplication with compare (DWC) and triple modular redundancy (TMR), and (3) recovery-based techniques, e.g., scrubbing. However, many studies do not consider fault-tolerant (FT) scheduling on FPGAs. Furthermore, critical embedded systems are quite often hard real-time (RT) [12]. That is, tasks have predefined specified time constraints, usually referred to as hard deadlines, and missing too many deadlines may cause, for example, a human or natural catastrophe or severe financial losses. Hence, to achieve a

[1] Throughout the remainder of the present paper, both transient and intermittent faults are referred to as soft errors (SEs).

high degree of reliability, both the timelines and the logical correctness of the output are crucial.

Critical embedded systems employing FPGAs as a processing element (PE), however, inherently have severe limitations on available resources and power capability [3]. Along with the marked tendency of utilizing FPGAs in critical RT embedded systems, these limitations have brought to light the need for a reliable scheduling algorithm design. The main goal of scheduling in computing is to efficiently use shared resources (e.g., PE) so as to make the system fair and fast. In contrast to the mentioned resilience techniques, the real advantage of FT scheduling is improving reliability on FPGAs—and the system—without the need for new additional external resources.

There is, however, a dearth of research aiming to employ scheduling techniques for fault tolerance in FPGAs. A few existing studies use, for the most part, the strongly static primary-backup (PB) scheduling technique with active redundancy [12, 13]. Although static scheduling has many benefits (e.g., synchronization and safeness for low-laxity tasks), only dynamic FT scheduling can efficiently utilize shared resources. Furthermore, the cost of increased power will be considerable with the use of active backup tasks, such as TMR. Thus, for critical embedded systems, where satisfaction with the energy constraint is absolutely vital, the focus should be on scheduling passive backups. To take advantage of both static and dynamic task scheduling, a hybrid scheduling technique can be used to manage and handle tasks and their active/passive backups under execution time constraints [14].

The present study proposes a novel hybrid RT scheduling technique to improve the MTTF of a system in the presence of soft errors (SEs). First, the proposed technique combines two traditional scheduling approaches, static and dynamic, for the FT scheduling of independent periodic RT hardware tasks on FPGA. For this purpose, some main principles are presented and proven. Next, all scheduled jobs are reordered according to the mentioned principles. The scheduler then applies both active and passive redundancies to the hardware tasks. Given that it is possible to overlap some passive backup copies in the schedule, the scheduler dynamically selects a candidate backup task to be configured in the preserved overloaded area at run time according to the proven principles. In comparison with the fully static fault tolerance study conducted by the active backup redundancy method [13], the experimental results show that the proposed hybrid scheduling approach enhances the MTTF of the system by an average factor of 1.22.

The rest of the current paper is organized as follows. Section 2 covers related work, while Sect. 3 describes the system model and provides notations. Section 4 formulates the problem in detail, for which Sect. 5 provides an illustrative example. Section 6 elaborates on the details of the proposed scheme and supports it by providing several lemmas and theorems. In Sect. 7, experimental results and a comparative study are presented. Finally, Sect. 8 concludes the paper and discusses potential future work.

2 Related work

In efforts to increase the reliability of FPGAs, several studies have introduced recovery-based and design-based techniques. Recovery-based techniques are specifically developed to prevent fault accumulation in SRAM memory cells [15]. For instance, to maximize the total reliability of the system, the authors in Ref. [16] established different scrubbing rates for different circuits based on the failure rate, so that the total reliability of the system is maximized. Periodical scrubbing permits the recovery from errors in SRAM cells induced by SEs periodically [17]. Another example is the scrubbing method presented in Ref. [18] which increases reliability by considering power optimization. Routing and re-routing techniques also improve FPGA reliability [19, 20]. Nevertheless, in order to achieve a high degree of reliability, such techniques should be employed in combination with design-based techniques [21].

To mitigate SEs, design-based techniques are generally dependent on resource redundancy [8]. These techniques utilize different replications, at different granularities, to increase system reliability [22]. Redundancy-based techniques have been investigated in several reliability studies in space mission systems [23], the cloud computing environment [24], and the multiprocessor platforms [25]. Numerous similar and dissimilar possible combinations of hardware, information, software, and time redundancy have been employed to recover failures by performing computations two- or more times [12, 26]. Nonetheless, most studies in the literature address only pure hardware redundancy techniques on FPGAs without considering scheduling. The main drawback is the throwing of too much redundant hardware, which is often impossible because of the severe limitations on the resources, costs, and power availability of embedded systems. Moreover, the overwhelming majority of faults in PEs are non-permanent in nature [12], and consequently, there is a need for a mechanism to protect the system from SEs.

As the term implies, FT scheduling complements enormous redundancy to achieve higher reliability. Irrespective of algorithms, all FT scheduling procedures are basically similar in that their main goal is to ensure the successful completion of tasks in spite of PE failure. Toward that aim, FT scheduling procedures, by and large, utilize time and PE redundancy with the PB approach. In the PB approach, at least two versions (primary and backup) of a task are scheduled. If the primary version of the task fails, then the backup copy offers another chance to complete the execution. The need for some fault detection mechanisms, such as acceptance test in the system should be noted, to detect task failure [12]. For further information about FT scheduling and RT systems, Refs. [12, 26] may be consulted.

For FT scheduling on the multiprocessor platform, Pathan in Ref. [27] introduced an FT perspective that executes backup tasks when faults are detected. A global PB scheduling algorithm was also proposed by employing both active and passive backups of RT sporadic tasks on a multi-core system [25]. In another work [28], researchers assumed only active backups and considered tolerating processor failures. Furthermore, authors in Ref. [29] utilized the PB approach for

FT scheduling in clouds. On the whole, it is possible to claim that a mature body of literature, currently, exists on the subject of FT RT scheduling in multiprocessor systems [12, 30]. Despite this, very few studies, in fact, have considered implementing both scheduling and fault tolerance techniques on FPGAs.

Not surprisingly, fault tolerance is a principal avenue in SRAM-based FPGA research. Employing FT scheduling techniques can be effective in mitigating SEs in FPGA-based critical embedded systems. For instance, [31] employed the PB approach with the aim of minimizing overlap time. Similarly, Ref. [32] presented an FT algorithm for scheduling hybrid tasks. The authors of a related research [13] introduced Pareto-based optimization methods to increase the reliability of dependent tasks running on SRAM-based FPGAs without deteriorating the total makespan. This Pareto-based technique conducts an exhaustive search to find the optimal scheduling table (with the highest reliability) of all tasks coupled with their active backups at design time. The present study considers, to the best of its knowledge, the Pareto-based approach to be the work most closely related to its own. Shortly after their research in [13], the same authors proposed another concept well-suited for scheduling periodic tasks, which is a technique for increasing the reliability of hardware task graphs called the configuration early fetch [33].

Although the studies discussed above focus on the FT scheduling of hardware tasks in FPGA, most do not consider RT constraints. Critical embedded systems, however, are quite often RT, as mentioned earlier. In other words, critical embedded systems must be based on guaranteed timeliness to meet predefined hard deadlines. Actually, however, most activities performed by RT systems are periodic in nature, e.g., regularly monitoring special conditions/cases or frequently receiving data from sensor nodes. The current paper presents a hybrid static-dynamic scheduling technique aiming to improve the total MTTF of periodic RT hardware tasks (represented as a bag-of-tasks) on SRAM-based FPGAs, without any need for extra external hardware redundancy.

3 Models and preliminaries

This section provides models and preliminaries. Section 3.1 introduces task model. In Sect. 3.2, reconfigurable device model is described. After that, reliability and fault models are presented in Sect. 3.3. Furthermore, to facilitate easy reading, some variables and symbols that are frequently used in the current paper are summarized in Table 1.

3.1 Task model

The present research assumes that the system is critical hard real-time consisting of n independent periodic hardware tasks with no interdependence constraints and represented as the bag-of-tasks $\Gamma = \left\{\tau_1, \tau_2, \dots, \tau_n\right\}$ scheduled on an SRAM-based partially run-time reconfigurable FPGA. As the name indicates, each periodic task $\tau_i \in \Gamma$ is

Table 1 A summary of frequent notations used in this paper

Notation	Description
τ_i	Real-time hardware task i
Γ	Set of all independent periodic real-time tasks: $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$
C_i	Worst-case computation time of τ_i
W_i	Size of τ_i in terms of configuration logic blocks (CLBs) count
B_i	Number of configuration bits of task τ_i in the bitstream
S_i	Number of sensitive bits in B_i expressed as a percentage
D_i	Relative hard deadline parameter
T_i	Period of τ_i which is equal to the deadline in the current paper
$\mathcal{T}$	Hyper period of all tasks in Γ: $\mathcal{T} = \mathrm{LCM}(T_1, T_2, \dots, T_n)$
S_R, S_C	Count of rows and columns in reconfigurable device
CD_G	Configuration delay time of a CLB group
CD_i	Configuration delay time for configuring task τ_i on FPGA
$r_{i,j}$	Residency time of $\tau_{i,j}$
$R(t)_{\tau_{i,j}}$	Reliability of the jth job of task τ_i which operates during t time units
λ_i	Failure rate of task τ_i in a given time interval
RD	Reconfigurable device
SEE, SEU	Single event effect, single event upset
SE	Soft error

made up of an infinite number of jobs (instances) and every job is issued exactly once every period of T_i time units. τ_i is characterized as follows:

$$\forall \tau_i \in \Gamma,\ \tau_i = (C_i, W_i, B_i, S_i, D_i, T_i) \tag{1}$$

where C_i is the worst-case computation time (or simply computation time throughout the present paper) of τ_i, and W_i represents the size of the task in terms of configurable logic blocks (CLBs) count [34]. B_i denotes the number of configuration bits of the task in the bitstream, and S_i is the number of sensitive bits in B_i expressed as a percentage. Any change in the value of sensitive bits affects the functionality of the task and will lead to a task failure [35]. The exact amount of sensitive bits of a hardware task is determined by experiments, such as radiation ground, fault injection, and emulation tests [13]. To the best of the present study's knowledge, the amount of sensitive bits reported in the literature has been no more than 35% [36], a figure taken into consideration by the current research. Finally, D_i and T_i are the relative hard deadline parameter or deadline (for the sake of simplicity in the remainder of the current paper) and the period of time, respectively.

In the majority of RT systems, the periodic tasks are synchronous and the relative deadline is equal to the period [12, 26]. The present study also assumes that D_i, which is called the implicit deadline, also represents the period ($D_i = T_i$). Furthermore, the scheduling table is made only for the smallest time interval called hyperperiod ($\mathcal{T}$) at design time; after that, it is repeated. Hyperperiod is equal to the least common multiple of all periods of the tasks.

$$\mathcal{T} = \mathrm{LCM}\ (T_1, T_2, \dots, T_n) \tag{2}$$

Therefore, each synchronous task τ_i has a total of ${}^{\mathcal{T}}\!/_{T_i}$ jobs in the hyperperiod, each of which is released exactly at the beginning of the period. This can be shown by the quantification:

$$\forall \tau_i \in \Gamma,\ \exists \tau_{i,j} : \tau_{i,j} \in \left\{ \tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,{}^{\mathcal{T}}\!/_{T_i}} \right\} \tag{3}$$

The current paper shows each backup copy with a superscript (an integer number just top-right of the job) indicating the backup number ($\tau_{i,j}^{(m)}$). For example, $\tau_{3,1}^{(2)}$ represents the second backup of the first job of task τ_3. Without loss of generality, all backup tasks are assumed to be clones of their primaries. That is, all backup task characteristics, such as computation time and deadline, are exactly equal to those of the primary version ($\tau_{i,j} = \tau_{i,j}^{(1)} = \tau_{i,j}^{(2)} = \cdots$). For fully utilizing PE to increase system reliability by responding to multiple task failures, it is also assumed that, except for resource availability, there is no limit to the number of backups for each job.

Backup tasks can be scheduled as passive or active. In the passive mode, backups start execution only after the completion time of their primary versions. On the other hand, in the active mode, backups and primaries are executed at the same time. Regardless of the backup mode, scheduling techniques are either static or dynamic. If the schedule is created at design time (i.e., static scheduling), all tasks priorities are determined before the system begins to run. In contrast, a dynamic scheduler determines tasks priorities as it executes. The present study utilizes a hybrid (static-dynamic) technique to schedule both active and passive backup tasks.

Generally, schedulers can be either preemptive or non-preemptive. Inasmuch as interrupting hardware tasks requires additional time for saving the current state of the task, and later probably considerable overhead for re-configuring the preempted task on the FPGA, the current work presents its technique with the assumption that there is no preemption in the run-time of the hardware tasks [13, 33, 37]. Thus, under a non-preemptive scheduling regime, once a hardware task begins its execution, it completes its computation at all times without any interruption. Nevertheless, the proposed hybrid scheduling technique can easily be ported to systems with a preemptive scheduler.

3.2 RD model

The target RD is an SRAM-based FPGA with partial run-time re-configurability which includes an array of CLBs. All hardware tasks in Γ are configured and run on a subset of this array, but the minimum addressable content in the FPGA is a group of CLBs referred to as a CLB group in the present paper. Thus, the target RD is characterized as follows:

$$\mathrm{RD} = \left(S_{\mathrm{R}}, S_{\mathrm{C}}, S_{\mathrm{G}}, \mathrm{CD}_{\mathrm{G}}\right) \tag{4}$$

where S_{R} and S_{C} represent the count of rows and columns, respectively. S_{G} indicates the size of a CLB group in terms of CLBs, and CD_{G} is the configuration delay time of a CLB group.

Since SRAM is volatile, the FPGA, upon starting, must read the initial configuration data from a non-volatile memory. It is assumed that this initial data is protected against SEs. According to the inherent limitation of configuring tasks on an FPGA, the task configuration process is performed in a serial manner [13, 38]. It is also assumed that hardware task τ_i can be configured on any location of the device if there is enough space for at least W_i [13, 33]. Finally, the FPGA utilization factor (U_Γ), which is the unitless fraction of time and area spent in the execution of the task set during a hyperperiod on the FPGA, is calculated as follows:

$$U_\Gamma = \frac{\sum_{i=1}^{n} \frac{\mathcal{T}}{T_i} \times (C_i + \mathrm{CD}_i) \times W_i}{S_\mathrm{R} \times S_\mathrm{C} \times \mathcal{T}} = \frac{1}{S_\mathrm{R} \times S_\mathrm{C}} \sum_{i=1}^{n} (C_i + \mathrm{CD}_i) \times W_i / T_i \tag{5}$$

where $S_\mathrm{R} \times S_\mathrm{C}$ is the FPGA size in terms of the CLB and CD_i represents the configuration delay time of task τ_i, which is easily calculated as follows:

$$\mathrm{CD}_i = \frac{W_i}{S_\mathrm{G}} \times \mathrm{CD}_\mathrm{G} \tag{6}$$

In other words, U_Γ is the fraction of total time–space area used by all tasks and their jobs relative to the entire time–space area in a hyperperiod.

3.3 Reliability and fault models

Of all fault types in digital systems, the current paper addresses SEs (intermittent faults and single/multiple transient faults) and assumes that permanent faults are dealt with by manufacturing testing or field testing methods [39]. A fault detection mechanism is also considered to be in place, such as an acceptance test, executable assertions, or a fail signal that detects job failure in the system [12]. Failure is detected exactly at the scheduled completion time of the job. Moreover, the time overhead for detecting job failure is included in the worst-case computation time (C_i).

In a recent work [40], researchers at the DDEmS laboratory introduced a reliability model for hardware tasks running on an FPGA to SEs. To validate the presented model, a complete set of practical experiments on real hardware tasks were conducted. Results confirmed the high accuracy of the model, especially in harsh environments in which only a 0.5% discrepancy was recorded between the true experimental results and estimated values. The present study also employs this reliability model which follows Poisson probability distribution. That is, faults are supposed to independently occur in an FPGA at a constant rate [13, 41]. Therefore, the reliability of the jth job of task τ_i, which operates during t time units, is calculated as follows:

$$R\big(t = C_i + r_{i,j}\big)_{\tau_{i,j}} = \mathrm{e}^{-\lambda_i (C_i + r_{i,j})},\ i \in \{1, 2, \ldots, n\}, j \in \left\{1, 2, \ldots, \mathcal{T}/T_i\right\} \tag{7}$$

where $r_{i,j}$ is the residency time of $\tau_{i,j}$ indicating the time elapsed from when a job is configured on the FPGA until its execution begins. If $\tau_{i,j}$ immediately starts its execution after the configuration, $r_{i,j}$ is equal to zero. However, as seen in the next section, it is sometimes possible to configure $\tau_{i,j}$ ahead of its arrival time. In such a circumstance, $r_{i,j}$ is added to the computation time. λ_i is the failure rate of task τ_i in a given time interval, which depends on the soft error rate (SER) of the environment, the number of configuration bits of the task in the bitstream (B_i), and the number of sensitive bits (S_i). λ_i is calculated as follows:

$$\lambda_i = \Lambda \times B_i \times S_i \tag{8}$$

where Λ indicates the bit flip rate in one bit of the bitstream and is a representative of the environment's SER, which is exponentially related to the critical charge of the circuit and radiation intensity (neutron flux) [42]. It should be noted that λ_i does not vary with time and is a constant rate for all τ_i jobs. Once the constant failure rate of the task is achieved, the task MTTF is easily calculated as follows [43]:

$$\mathrm{MTTF}_{\tau_i} = \frac{1}{\lambda_i} = \frac{C_i}{1 - R\left(C_i\right)_{\tau_i}} \tag{9}$$

It is assumed that the system is properly executed if and only if all independent tasks correctly finish execution before their individual deadlines. In other words, the system runs as a series-parallel system. To calculate the reliability of the series-parallel system, the current paper utilizes the reliability block diagram (RBD), which is an inductive model for analyzing the reliability of complex systems [37, 44, 45]. Hence, the total reliability of the system in the hyperperiod is calculated as follows:

$$R(\mathcal{T})_{\mathrm{Sys}} = \prod_{i=1}^{n} \prod_{j=1}^{\mathcal{T}/T_i} R\left(T_i\right)_{\tau_{i,j}} \tag{10}$$

It is worth mentioning that every job of a task may have several backups. Since only one instance of a job (the primary or one of its backups) is needed to successfully complete its execution, the reliability of $\tau_{i,j}$ with m backup copies is calculated as a *consecutive-one-out-of-m* system. Therefore, the reliability of every job in one period is calculated as follows:

$$R\left(T_i\right)_{\tau_{i,j}} = 1 - \left(1 - R\left(C_i + r_{i,j}\right)_{\tau_{i,j}}\right)\left(1 - R\left(C_i + r_{i,j}^{(1)}\right)_{\tau_{i,j}^{(1)}}\right) \dots \left(1 - R\left(C_i + r_{i,j}^{(m)}\right)_{\tau_{i,j}^{(m)}}\right) \tag{11}$$

Finally, the total MTTF of the system is evaluated by:

$$\mathrm{MTTF}_{\mathrm{Sys}} = \frac{\mathcal{T}}{1 - R(\mathcal{T})_{\mathrm{Sys}}} \tag{12}$$

4 Problem formulation

Given the following as inputs:

1. A task set $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$ of independent periodic RT hardware tasks, where τ_i is modeled by a 6-tuple $\tau_i = (C_i, W_i, B_i, S_i, D_i, T_i)$
2. The target RD given by $\mathrm{RD} = (S_\mathrm{R}, S_\mathrm{C}, S_\mathrm{G}, \mathrm{CD}_\mathrm{G})$

Determine a scheduling S for all tasks and their jobs so as to find

possible slack times and areas to preserve for backup copies, and
candidate backup tasks to overload preserved areas

such that while SEs occur:

1. Candidate tasks have another chance of meeting their individual deadlines, and
2. The MTTF of the system is increased.

Quite apart from algorithms, the PB approach for FT scheduling is, perhaps, the most important approach in the literature [12]. As SEs occur, jobs affected by the faults need to be re-configured (backup jobs) so as to not to miss their individual deadlines. Due to severe limitations on available resources and power capability, determining candidate jobs for true backup overloading on preserved areas on the FPGA must be carefully considered. In addition, as seen later, selecting true candidate backups can dramatically increase the total MTTF. Methods, such as [13, 37, 44], utilize the fully static FT scheduling of hardware tasks to improve reliability. However, static scheduling imposes a penalty on the perfect usage of slack times and areas. The present paper addresses these challenges by benefiting from a hybrid static-dynamic scheduling technique.

5 Illustrative example

This section presents a simple yet comprehensive motivational example of an RT task set composed of three hardware tasks. In the following example, it is assumed that the amount of sensitive bits is the same for all tasks and, hence, the differences between the reliabilities of the tasks depend on computation time, residency time, and task size.

$\Gamma_1 = \{\tau_1, \tau_2, \tau_3\}$ shall be a task set and RD is assumed to be a simple FPGA. Each task $\tau_i = (C_i, W_i, \mathrm{CD}_i, D_i)$, where C_i is the computation time, W_i is the size of the task in terms of CLBs, CD_i is the configuration delay time of the task, and D_i is the relative deadline equal to the period ($D_i = T_i$). All three tasks in Γ_1 have the same size, deadline, and configuration delay time: $W_1 = W_2 = W_3$, $D_1 = D_2 = D_3 = 12$ s, $\mathrm{CD}_1 = \mathrm{CD}_2 = \mathrm{CD}_3 = 1$ s, and C_1, C_2, C_3 are 3, 4, and 5 s, respectively. Therefore,

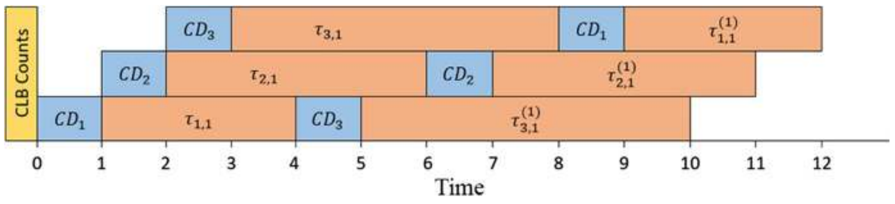

Fig. 1 A simple example of the strong static scheduling of task set Γ_1, which is composed of three tasks with the PB approach

each task has exactly one job in the hyperperiod ($\mathcal{T}$= 12 s). Since all tasks have the same size, their failure rate (λ_i) is also constant and equal to one another. For this example, it is supposed that the average number of task failures is 2 in 1000 s of execution of the task ($\lambda_i = 0.002$).

Figure 1 shows one possible static scheduling of the three tasks. All tasks arrive simultaneously at time 0 (i.e., *critical instant scenario*). As seen, each job has one backup. Thus, this system has the ability to tolerate a total of one job failure. It should be noted that the backups of $\tau_{1,1}$ and $\tau_{2,1}$ are configured and executed only when their primaries fail, as indicated by the acceptance test flags, and this embodies the main concept of the passive backup mode. On the other hand, the backup of $\tau_{3,1}$ is configured and executed before its primary finishes execution. In other words, passive and active backups are employed together in order to increase reliability. According to Eq. (11), the reliability of each job, as a *consecutive-one-out-of-two* system, is calculated as follows:

$$R(t=12)_{\tau_{i,j}} = 1 - \left(1 - R\left(t = C_i + r_{i,j}\right)_{\tau_{i,j}}\right)\left(1 - R\left(t = C_i + r_{i,j}^{(1)}\right)_{\tau_{i,j}^{(1)}}\right)$$

By studying Fig. 1, one recognizes that the residency time is zero for all jobs and their backups. Consequently, according to Eq. (7), $R\left(t = C_i\right)_{\tau_{i,j}} = \mathrm{e}^{-\lambda_i C_i}$. Therefore:

$$R(12)_{\tau_{1,1}} = 1 - \left(1 - \mathrm{e}^{-0.002\times3}\right)^2,\ R(12)_{\tau_{2,1}} = 1 - \left(1 - \mathrm{e}^{-0.002\times4}\right)^2,$$
$$R(12)_{\tau_{3,1}} = 1 - \left(1 - \mathrm{e}^{-0.002\times5}\right)^2$$

As a result, the reliability and MTTF of the system are evaluated by the aid of Eqs. (10) and (12):

$$R(\mathcal{T}=12)_{\mathrm{Sys}} = R(12)_{\tau_{1,1}} \times R(12)_{\tau_{2,1}} \times R(12)_{\tau_{3,1}} \approx 0.999801731$$
$$\mathrm{MTTF}_{\mathrm{Sys}} = \frac{\mathcal{T}}{1 - R(\mathcal{T})_{\mathrm{Sys}}} \approx 5043\mathcal{T}$$

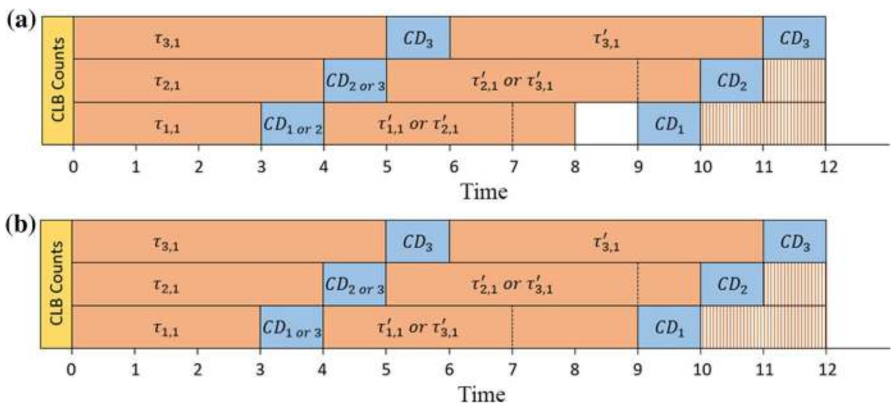

Fig. 2 Two possible hybrid scheduling tables for task set Γ_1

To put it simply, the MTTF_{Sys} value means it is expected that the system correctly operates for 5043 hyperperiods (or 60516 s). The MTTF (and reliability) of this system is also estimated by our simulation program. To better simulate scheduling, the current work chooses *true* random numbers—generated via atmospheric noise [46]—over pseudo-random numbers which are typically generated by computer programs in a predictable fashion via a mathematical formula. The slight difference of about 3.18E−06% between the exact value of reliability and its simulated result proves the accuracy of the present study's simulation.

Now, by changing the scheduling technique, the current paper presents another possible scheduling of task set Γ_1 (Fig. 2a). In this hybrid scheduling, jobs and their backups are first statically scheduled in such way that some preserved slack areas on the FPGA are overloaded with different backup jobs; backups are allowed to overlap other backup copies. Then, at run time, the scheduler dynamically decides which backup copy should be configured and executed. Since the scheduling is static-dynamic, the exact number of backups for each job is unknown a priori. For this reason, the notation $\tau'_{i,j}$ distinguishes backup jobs from their primaries. It should be noted that primary jobs are configured at the end of the hyperperiod. Thus, the job residency times ($r_{i,j}$) for $\tau_{2,1}$ and $\tau_{1,1}$ are 1 and 2 s, respectively, which must be considered when calculating job reliability [Eq. (7)].

One example is the scheduling table shown in Fig. 2a with the following scenario. All three jobs simultaneously start their execution at time 0. If $\tau_{1,1}$ fails, it will be detected at time 3 and then the scheduler configures its backup ($\tau'_{1,1}$). Otherwise, the scheduler will configure $\tau'_{2,1}$, which is the backup of job $\tau_{2,1}$. If $\tau_{1,1}$ successfully finishes its execution at time 3, there is no need to re-execute it and so a backup of $\tau_{2,1}$ is configured at this time. $\tau'_{2,1}$ starts its execution at time 4. Exactly at this time, a failure in $\tau_{2,1}$ is detected and the scheduler immediately configures another backup of this job ($\tau'_{2,1}$). This new backup starts its execution at time 5, at which point there are two backups for $\tau_{2,1}$ simultaneously executed. Following the assumption of the *consecutive-one-out-of-three* system, it is logical that reliability increases in comparison with the strong static scheduling in Fig. 1. MTTF_{Sys} for the hybrid scheduling shown in Fig. 2a is 7104 $\mathcal{T}$, which represents a 40.8% increase as compared to the scheduling table in Fig. 1.

Although the MTTF of the system can remarkably increase by the hybrid scheduling technique, improvement is still possible in the static phase by correct selection of candidate backups for overloading preserved areas. For example, Fig. 2b provides another possible hybrid scheduling of task set Γ_1. Clearly, the only difference between Fig. 2a, b is the overlapped backup jobs. In this scheduling, if both $\tau_{1,1}$ and $\tau_{2,1}$ successfully complete their execution, then $\tau_{3,1}$ will have three backups (a *consecutive-one-out-of-four* system). With this scheduling, $\mathrm{MTTF}_{\mathrm{Sys}}$ is 7202$\mathcal{T}$, which indicates a 1.4% increase compared to the scheduling table in Fig. 2a.

Surprisingly, the hybrid static-dynamic scheduling in Fig. 2b enhances the MTTF of the system by a factor of 1.43 in contrast to the strongly static scheduling table in Fig. 1. Furthermore, Fig. 2b presents a dedicated preserved slack area for $\tau'_{3,1}$ which is not overloaded with another backup. That is, if $\tau_{3,1}$ does not fail, this slack area will be useless, which occurs 99% of the time ($\tau_{3,1}$'s reliability is 0.9900498.). Considering this, if $\tau'_{3,1}$ overlaps with $\tau'_{1,1}$ in the time interval of 5 s to 9 s, then $\mathrm{MTTF}_{\mathrm{Sys}}$ will rise up to 12552$\mathcal{T}$. In contrast to the static scheduling in Fig. 1, this dramatic increase of about 2.5 times is because, at time 5, a third chance is given to $\tau_{1,1}$ to meet its deadline whenever $\tau_{3,1}$ does not fail. The next section shall describe the proposed scheme in detail.

6 Proposed scheme

The first stage of the proposed algorithm schedules tasks according to a non-preemptive earliest deadline first (EDF-NP) policy. Next, the algorithm addresses the issue of selecting candidate backup jobs for overloading in the possible slack times and areas. Finally, it determines which candidate backup job should be configured and executed at run time.

6.1 Principles for backup overloading

This subsection highlights some fundamental principles of the proposed technique by presenting several lemmas and theorems that provide the criteria by which backup jobs can provide higher reliability. Then, the proposed algorithm is summarized in two pseudo-codes. The main principles are first discussed.

Principle A1: When there is a free area for configuring a backup task and there are two candidate tasks, scheduling a backup of the task with lower reliability leads to greater improvement of total system reliability.

This simple yet important principle can be easily proven by the following Lemma.

Lemma 1 *Consider two tasks τ_1, τ_2 where $R_{\tau_1} < R_{\tau_2}$. Slack time t is available while $t = max\{C_1, C_2\}$, and there exist at least free CLBs where $= max\{W_1, W_2\}$; then, scheduling a backup of τ_1 better improves the total reliability of the system.*

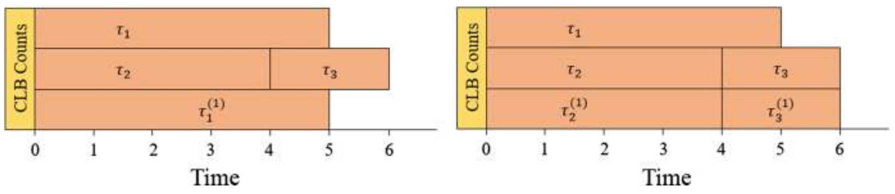

Fig. 3 Principle A2: scheduling two backups of tasks τ_2 and τ_3 yields higher reliability than scheduling a backup of task τ_1

Proof If a backup of τ_1 is considered, according to Eqs. (10) and (11), the reliability of the system would be:

$$R_{\mathrm{Sys}} = \left(1 - \left(1 - \mathrm{e}^{-\lambda_1 C_1}\right)^2\right)\mathrm{e}^{-\lambda_2 C_2} \tag{13}$$

and if a backup of task τ_2 is created, then

$$R_{\mathrm{Sys}} = \left(1 - \left(1 - \mathrm{e}^{-\lambda_2 C_2}\right)^2\right)\mathrm{e}^{-\lambda_1 C_1} \tag{14}$$

Since $\lambda_1 C_1 > \lambda_2 C_2$ ($R_{\tau_1} < R_{\tau_2}$), it is concluded that (13) > (14), thus indicating that, when there are two tasks, making a backup of the task with lower reliability will further increase the whole reliability. Considering this, the following principle extends Lemma 1 by about three tasks. □

Principle A2: When there are three candidate tasks in which $R_{\tau_1} \le R_{\tau_2}$, $R_{\tau_1} \le R_{\tau_3}$, $R_{\tau_1} \ge R_{\tau_2} R_{\tau_3}$ and there is enough free area for configuring and executing either a backup task of τ_1 or two backups of τ_2 and τ_3, then the scheduling of the two backups, τ_2 and τ_3, better improves the total reliability of the system (Fig. 3).

This principle is proven in the following Theorem 1:

Theorem 1 *Assume that τ_1, τ_2, and τ_3 are three RT tasks in which $R_{\tau_1} \le R_{\tau_2}$, $R_{\tau_1} \le R_{\tau_3}$, the reliability of task τ_1 is equal to or more than the multiplication of the other tasks' reliabilities ($R_{\tau_1} \ge R_{\tau_2} R_{\tau_3} \Rightarrow \lambda_1 C_1 \le \lambda_2 C_2 + \lambda_3 C_3$), and there exists enough free area for configuring and executing either a backup of τ_1 or two backups of τ_2 and τ_3. Then, scheduling a backup of τ_1 yields a reliability equal to or less than that of scheduling backups of the other two tasks.*

Proof Consider a backup of τ_1 is scheduled. Therefore, the reliability of the system is calculated as follows:

$$\begin{aligned} R_{\mathrm{Sys}}(option\ a) &= \left(1 - \left(1 - R_{\tau_1}\right)^2\right) R_{\tau_2} R_{\tau_3} \\ &= R_{\tau_1} R_{\tau_2} R_{\tau_3}\left(2 - R_{\tau_1}\right) \end{aligned} \tag{15}$$

On the other hand, if a backup of each task τ_2, τ_3 is scheduled, the total reliability would be

$$R_{\mathrm{Sys}}(option\ b) = \left(1 - \left(1 - R_{\tau_2}\right)^2\right)\left(1 - \left(1 - R_{\tau_3}\right)^2\right)R_{\tau_1} = R_{\tau_1}R_{\tau_2}R_{\tau_3}\left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \tag{16}$$

and we want to prove that

$$\begin{gathered} R_{\mathrm{Sys}}(option\ a) \le R_{\mathrm{Sys}}(option\ b) \\ \Rightarrow R_{\tau_1}R_{\tau_2}R_{\tau_3}\left(2 - R_{\tau_1}\right) \le R_{\tau_1}R_{\tau_2}R_{\tau_3}\left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \\ \Rightarrow 2 - R_{\tau_1} \le \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \end{gathered} \tag{17}$$

$R_{\tau_1} \ge R_{\tau_2}R_{\tau_3}$ is known and hence $\lambda_1 C_1 \le \lambda_2 C_2 + \lambda_3 C_3$. Suppose $\lambda_1 C_1 = \lambda_2 C_2 + \lambda_3 C_3 - \alpha$, where α is a nonnegative real number. As a consequence,

$$R_{\tau_1} = \mathrm{e}^{-\lambda_1 C_1} = \mathrm{e}^{-\lambda_2 C_2 - \lambda_3 C_3 + \alpha} = \frac{R_{\tau_2}R_{\tau_3}}{\mathrm{e}^{-\alpha}},\ \alpha \in \mathbb{R}_{\ge 0} \tag{18}$$

If this is substituted for R_{τ_1} in Inequality (17), then

$$2 - \frac{R_{\tau_2}R_{\tau_3}}{\mathrm{e}^{-\alpha}} \le \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \tag{19}$$

Now, considering inequality (19), the following two bounds for α may be assumed as:

(A) α is very high

$$\begin{gathered} \lim_{\alpha \to \infty} \frac{R_{\tau_2}R_{\tau_3}}{\mathrm{e}^{-\alpha}} = +\infty \\ \Rightarrow 2 - \infty \le \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \end{gathered} \tag{20}$$

As $R_{\tau_i} \in [0, 1]$, clearly, Inequality (19) always holds.

(B) α is very small

$$\begin{gathered} \lim_{\alpha \to 0} \frac{R_{\tau_2}R_{\tau_3}}{\mathrm{e}^{-\alpha}} = R_{\tau_2}R_{\tau_3} \\ \Rightarrow 2 - R_{\tau_2}R_{\tau_3} \le \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \end{gathered} \tag{21}$$

The following two conditions can be assumed for the environment:

1. The environment is not too harsh and the amount of λ_i is very low. As a result, the reliability of the tasks is very high. That is,

$$\lim_{R_{\tau_2},R_{\tau_3}\to 1} 2 - R_{\tau_2}R_{\tau_3} = \lim_{R_{\tau_2},R_{\tau_3}\to 1} \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) = 1 \tag{22}$$

2. The environment is very harsh and the amount of λ_i is also very high. As a result, the reliability of the tasks is very low. That is,

$$\left(\lim_{R_{\tau_2},R_{\tau_3}\to 0} 2 - R_{\tau_2}R_{\tau_3} = 2\right) < \left(\lim_{R_{\tau_2},R_{\tau_3}\to 0} \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) = 4\right) \tag{23}$$

From Eq. (22) and Inequality (23), it is easily concluded that Inequality (21) also holds.Since upper and lower bounds were considered to prove Inequality (19), it is essential to prove that $f(x) = \mathrm{e}^{-x}$ is a monotonic (non-increasing or non-decreasing) function on the interval $[0, +\infty)$. In this case, e^{-x} is non-increasing. By definition, the continuous function $f(x)$ is non-increasing on the closed interval $[a, b]$ if and only if

$$\forall x \in (a, b); f'(x) \le 0 \tag{24}$$

Moreover, $f(x) = \mathrm{e}^{-x}$ is a continuous function on $\mathbb{R}(f : \mathbb{R} \to \mathbb{R}_{>0})$ and

$$\forall x \in \mathbb{R}; f^{'}(x) = -\mathrm{e}^{-x} < 0\left(f^{'} : \mathbb{R} \to \mathbb{R}_{<0}\right). \tag{25}$$

Evidently, not only is the reliability function $f(x) = \mathrm{e}^{-x}$ non-increasing, it is also decreasing and so this completes the proof of Theorem 1. In other words, in such conditions, the best option is to schedule backups of the two more reliable tasks, especially when the reliability of the tasks is not high. The induction is complete. □

Principle A3: When there are three candidate tasks in which $R_{\tau_1} \le R_{\tau_2}$, $R_{\tau_1} \le R_{\tau_3}$, $R_{\tau_1} \le R_{\tau_2}R_{\tau_3}$ and there is a slack time and free area for configuring and executing either a backup of τ_1 or two backups of τ_2 and τ_3, then scheduling two backups of τ_2 and τ_3 better improves the total reliability of the system if $R_{\tau_2} + R_{\tau_3} \le 1$.

This principle states that, when $R_{\tau_1} \le R_{\tau_2}R_{\tau_3}$, then selecting candidate tasks for the backup is not simple and requires exact calculation. However, if $R_{\tau_2} + R_{\tau_3} \le 1$, then two backups of τ_2 and τ_3 should be scheduled (Fig. 4). This claim is proven in Theorem 2.

Theorem 2 *Assume that* τ_1, τ_2, *and* τ_3 *are three independent RT tasks in which* $R_{\tau_1} \le R_{\tau_2}$, $R_{\tau_1} \le R_{\tau_3}$ *and the reliability of task* τ_1 *is equal to or less than the*

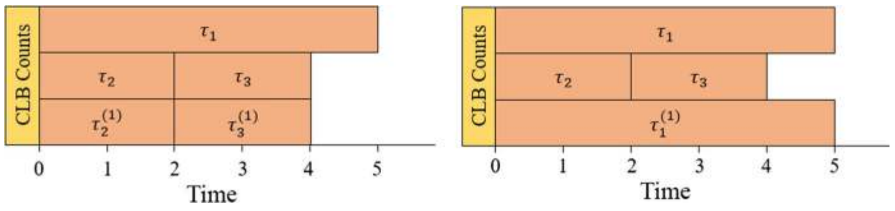

Fig. 4 Principle A3: scheduling two backups of tasks τ_2 and τ_3 yields higher reliability than scheduling a backup of task τ_1 if $R_{\tau_2} + R_{\tau_3} \leq 1$

multiplication of the other task reliabilities ($R_{\tau_1} \leq R_{\tau_2} R_{\tau_3} \Rightarrow \lambda_1 C_1 \geq \lambda_2 C_2 + \lambda_3 C_3$). There are also enough area and time for scheduling either a backup task of τ_1 or two backups of τ_2 and τ_3. Scheduling backups of τ_2 and τ_3 then further increases the total reliability of the system if $R_{\tau_2} + R_{\tau_3} \leq 1$.

Proof Similar to Inequality (17), a situation should be found in which

$$R_{\text{Sys}}(option\ a) \leq R_{\text{Sys}}(option\ b)$$
$$\Rightarrow 2 - R_{\tau_1} \leq \left(2 - R_{\tau_2}\right)\left(2 - R_{\tau_3}\right) \Rightarrow 1 - R_{\tau_2} - R_{\tau_3} \geq \frac{-R_{\tau_1} - R_{\tau_2} R_{\tau_3}}{2} \tag{26}$$

If condition $R_{\tau_2} + R_{\tau_3} \leq 1$ is satisfied, then the statement is true. □

Principle A4: Tasks with lower reliabilities should be scheduled as soon as possible (ASAP). This will raise the chance of scheduling additional backups (a result of Lemma 1). With the ASAP strategy, tasks with lower reliability are moved toward the head of the ready queue and are scheduled before other tasks.

Principle A5: Pre-fetch tasks when possible.

Task pre-fetching is the configuring of tasks before they arrive. Consequently, this can release more free areas for scheduling more backups. However, the residency time overhead ($r_{i,j}$) is added to the task computation time [Eq. (7)].

$$r_{i,j} = \text{EST}_{\tau_{i,j}} - \text{CFT}_{\tau_{i,j}} \tag{27}$$

where $\text{EST}_{\tau_{i,j}}$ is the execution start time of $\tau_{i,j}$ and $\text{CFT}_{\tau_{i,j}}$ denotes the configuration finish time of $\tau_{i,j}$.

Because configurations of tasks are performed in a serial manner, $\text{CFT}_{\tau_{i,j}}$ is calculated as:

$$\text{CFT}_{\tau_{i,j}} = \text{CST}_{\tau_{i,j}} + \text{CD}_i \tag{28}$$

i in which $\text{CST}_{\tau_{i,j}}$ is the configuration start time of $\tau_{i,j}$ and CD_i is the configuration delay time of task *i*.

Since all tasks are independent, each job $\tau_{i,j}$ can be configured when RD is available:

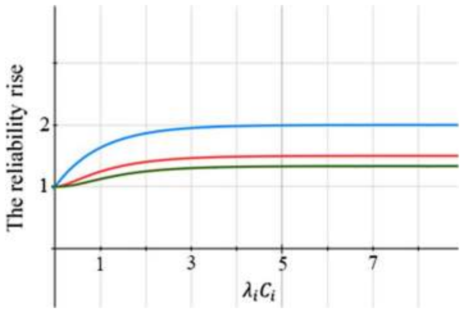

Fig. 5 The reliability rise rate (the slope of reliability rise) declines when the number of backups increases. The blue line indicates the reliability rise when the first backup is added. The red line and the green line show the reliability rise when the second and third backups are added, respectively (color figure online)

$$\mathrm{CST}_{\tau_{i,j}} = \mathrm{avl}\left(\mathrm{RD}, \tau_{i,j}\right) \tag{29}$$

where $\mathrm{avl}\left(\mathrm{RD}, \tau_{i,j}\right)$ determines the immediate time after the last configuration when the RD is available and has enough free area for configuring $\tau_{i,j}$.

Principle A6: The difference in the number of task backups must not exceed 1. In other words, as long as there are some tasks with $m-1$ backups or less and there is enough time–space slack to schedule mth backup, $m+1$th backups of the other tasks should not be scheduled.

In fact, Principle A6 explains that the slope (acceleration) of the reliability rise declines when the number of task backups increases. Figure 5 shows that the slope of reliability rises when the first, second, and third backups are added. The blue line depicts a rise in the slope of reliability when the first backup is added. That is, the result of R(1 out of 2 system)/R(simple system) is at most 2 for a large value of $\lambda_i C_i$. With the addition of a second backup, the reliability of the task increases at most by a factor of 1.5 in comparison with the previous step, that is, the result of R(1 out of 3 system)/R(1 out of 2 system) $\in [1, 1.5]$ (Fig. 5's red line). Similarly, the green line represents the reliability rise factor when the third backup is added, which is between 1 and 4/3.

The proposed heuristic algorithm mainly uses proven principles (A1, A2, A3, and A6) as well as Principles A4 and A5, which are good heuristics for scheduling primary jobs and overloading possible free areas with candidate backups. The algorithm description is further detailed in Algorithm 1. This algorithm schedules all tasks and their jobs with the EDF-NP policy so that Principles A1–A6 hold. For example, after jobs are scheduled according to the EDF-NP policy on RD in Line 4, jobs in Line 5 are reordered so that jobs with a lower reliability start their execution as soon as possible (Principle A4). Next, in Line 6, the algorithm attempts to pre-fetch jobs when possible (Principle A5). Finally, according to Principles A1, A2, A3, and A6, backup overloading is performed in Lines 7–9 for possible slack areas. In other words, in the *for each loop*, Principles A1–A3 are applied according to Principle A6.

Algorithm 1: EDF-NP scheduling with overlapping backup jobs (*static phase*)

```
01  input $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$  //A task set which is a collection of independent periodic real-time tasks
02  input RD  //Reconfigurable Device
03  output scheduled jobs on RD
04  $\mathcal{S}$ ← Schedule tasks (and their jobs) in $\Gamma$ with EDF-NP policy  //Scheduling table $\mathcal{S}$
05  Reorder jobs in $\mathcal{S}$ so that jobs with lower reliability begin their execution sooner  //Principle A4
06  Pre-fetch jobs if it is possible  //Calculate $r_{i,j}$ according to Equations 27, 28, and 29; Principle A5
07  foreach slack area "s" in $\mathcal{S}$ do
08      $overloadedBackups_s$ ← Determine candidate jobs according to Principles A1-A3 and A6
09  end foreach
10  Place scheduled jobs table $\mathcal{S}$ on RD
```

In the dynamic phase, on the other hand, the scheduler behaves event-triggered at run time. That is to say, events, such as job arrive, job finish, and start backup jobs in slack areas, initiate activities in the system. Such events are detailed in Algorithm 2.

Algorithm 2: On scheduler event (*dynamic phase*)

```
01  On job arrive( ) {
02      input job $\tau_{i,j}$
03      start $\tau_{i,j}$ according to the scheduling table $\mathcal{S}$
    }
04  On job finish( ) {
05      input job $\tau_{i,j}$, $executionResult$  //$executionResult \in \{success, failure\}$
06      status$_{\tau_{i,j}}$ ← $executionResult$
    }
07  On slack start( ) {
08      input set of overloaded backups in the slack area "s"  //Call this set $overloadedBackups_s$
09      foreach job $\tau_{i,j}$ in $overloadedBackups_s$ do
10          if status$_{\tau_{i,j}}$ = "$success$" then remove job $\tau_{i,j}$ from $overloadedBackups_s$
11      end foreach
12      Start the backup job with the lowest reliability from $overloadedBackups_s$
    }
```

As seen, Algorithm 2 illustrates the three main run-time events which activate the dispatcher. The first event is upon the arrival of a new job (instance). The second event is when a job finishes its execution. The primary task's execution result (success or failure in the proposed algorithm) is the deciding factor for the dispatcher when determining which candidate backup copy should be selected to configure on the preserved overloaded area. This determination is made at the start of the third event (on slack start), as described in Lines 7 to 12. When the slack area starts, the scheduler dynamically checks the status of all overloaded backup jobs at run

Table 2 Characteristics of task set Γ_2 depicted in Fig. 6 (all times are in milliseconds)

Task (τ_i)	Computation time (C_i)	Deadline ($D_i = T_i$)	CLB count (W_i)	Configuration delay (CD_i)	Failure rate (λ_i)
τ_1	350	1000	200	50	8.54701E−10
τ_2	800	2000	250	150	1.68380E−9
τ_3	200	1000	400	50	1.49573E−9
τ_4	450	1000	200	150	8.54701E−10

time and removes those jobs whose primaries have successfully finished execution. Finally, among the remaining overloaded backups, the backup job with the lowest reliability is selected by the dispatcher to configure and execute on RD. In the following subsection, the working of the proposed hybrid technique is explained.

6.2 Example

This section offers a general example of the proposed algorithm's operation by using task set $\Gamma_2 = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. It is assumed that the RD is a simple FPGA with a capacity of 650 CLBs. All the assumptions of the earlier example hold but each task has its own size, configuration delay time, and deadline. Thus, the task failure rate, which depends on task size, computation (and residency) time, and the SER of the environment, is not the same for all tasks. Table 2 provides the task failure rates and other characteristics of Γ_2.

Figure 6a shows a static FT scheduling without backup overloading. In this schedule, $\tau_{1,2}$ is pre-fetched at time 800 ms, which frees up additional time for configuring and executing another backup at time 1350 ms. As seen, $\tau_{1,2}$ has three backups, but, according to Principle A6, the reliability rise declines when the number of backups of a specific task increases. The total reliability and MTTF of the system are about 0.999997777 and 449999 $\mathcal{T}$, respectively.

According to Fig. 6b, all the mentioned principles, A1 to A6, are taken into account to schedule Γ_2. The step-by-step explanation of Algorithm 1 is as follows: In Line 4, scheduling table $\mathcal{S}$ is created according to the EDF-NP policy.

$$\mathcal{S} = \{\tau_{4,1}, \tau_{3,1}, \tau_{1,1}, \tau_{2,1}, \tau_{1,2}, \tau_{4,2}, \tau_{3,2}\}$$

It should be noted that, in the EDF-NP policy, whenever a scheduling event occurs, the task closest to its deadline is selected for execution. In this subsection's example, τ_1, τ_2, τ_3 have the same deadline and, therefore, are selected in a random order. Next, Principle A4 is applied (Line 5) so that jobs with lower reliability start their execution sooner. According to Table 2:

$$R_{\tau_2} < R_{\tau_3} < R_{\tau_4} = R_{\tau_1}$$

Hence, $\tau_{2,1}$ and $\tau_{3,1}$ move to the front of the scheduling table:

$$\mathcal{S} = \{\tau_{2,1}, \tau_{3,1}, \tau_{4,1}, \tau_{1,1}, \tau_{1,2}, \tau_{4,2}, \tau_{3,2}\}$$

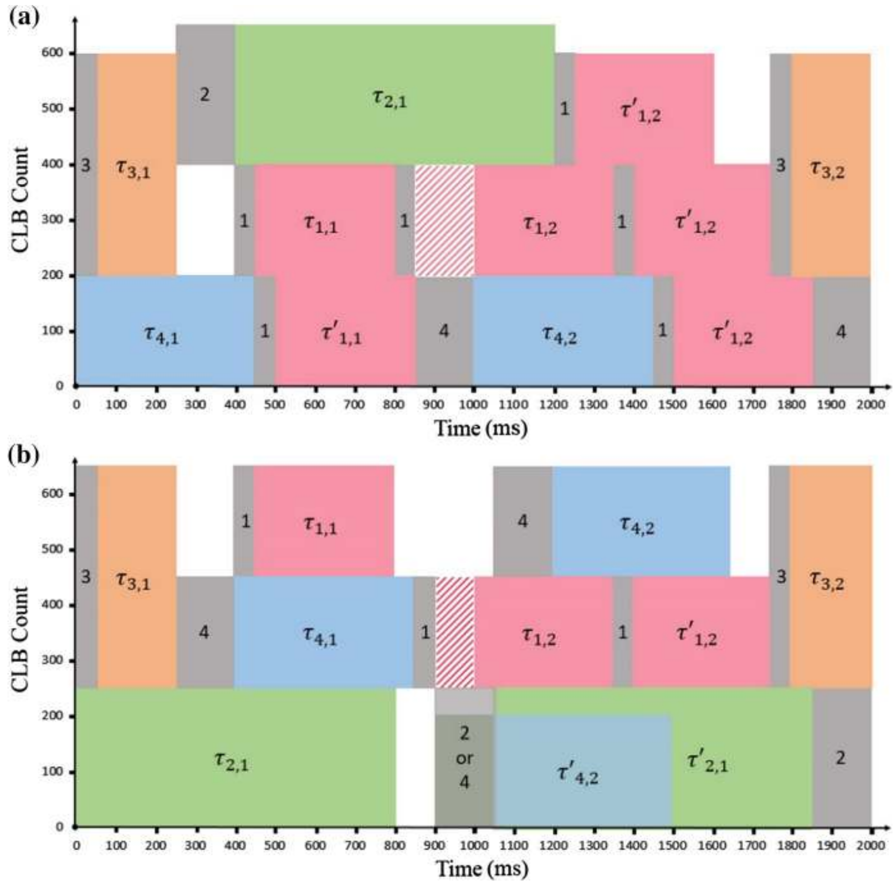

Fig. 6 Two possible scheduling methods of task set Γ_2 composed of four tasks. Gray boxes represent configuration delays

At this point, in Line 6, the algorithm looks for those tasks which are possible to pre-fetch before their next request (Principle A5). In this subsection's example, there is enough area and time to pre-fetch $\tau_{1,2}$ before time 1000 ms. Finally, in Lines 7–9, the algorithm determines the candidate jobs for overloading in the time–space slacks. In the *for each loop*, it should be noted that Principle A6 always holds. That is to say, the algorithm first tries to avoid a situation in which some tasks have multiple backups, while some others do not. According to Principle A1, the best candidate task is the task with the lowest reliability. Therefore, in the present example, the first selected candidate for a backup is τ_2. Furthermore, according to Principles A2 and A3, when there are three tasks, τ_1, τ_2, τ_3, and there is enough free area for configuring and executing either a backup task of τ_1 or two backups of τ_2 and τ_3, then scheduling two backups of τ_2 and τ_3 usually better improves the total reliability of the system than scheduling a backup of the task with the lowest reliability. Thus, instead of scheduling a backup of the task with the lowest reliability, all two-element subsets of $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ are checked to find a pair of tasks for which to schedule two backups. In this example, all two-element subsets eligible for replacement by a backup of τ_2 are:

$$\{\tau_1, \tau_3\}, \{\tau_1, \tau_4\}, \{\tau_3, \tau_4\}$$

By definition, a set of n elements has $\binom{n}{2}$ subsets of 2 elements and, therefore, calculating all 2-element subsets is performed in $O(n^2)$. In the present example, the algorithm looks to replace a possible pair of tasks with a backup of candidate task τ_2 between time 900 ms and 1850 ms. Since none of the above pair of tasks are schedulable in this area, the first backup of τ_2 is scheduled ($\tau'_{2,1}$). In the next step, the hybrid scheduler looks for the best option with which to overload the slack time–space area. Due to its high failure rate, the first candidate is $\tau_{3,2}$, but it does not fit into the slack area. The second candidate can be either $\tau_{4,2}$ or $\tau_{1,2}$. Since it is possible for $\tau_{1,2}$ to have a backup between 1350 and 1750 ms in the slack time–space area, a backup of $\tau_{4,2}$ is used to overload the slack area according to Principle A6. Since, at the most, n jobs are checked, this step is performed in $O(n)$.

The task with lower reliability, τ_2, is thus configured and executed as soon as possible. As a result, it has enough area and time for scheduling a backup version at time 900 ms. Moreover, the preserved area between times 900 and 1850 ms is overloaded by backup copies of jobs $\tau_{2,1}$ and $\tau_{4,2}$. If $\tau_{2,1}$ fails, its backup ($\tau_{2,1}^{(1)}$) will be configured at time 900 ms. Otherwise, a backup of $\tau_{4,2}$ ($\tau_{4,2}^{(1)}$) will be configured. It should be noted that $\tau_{2,1}$ is pre-fetched just before its period (at time 1850 ms). Furthermore, there is no scheduled job with a second or more backups (Principle A6 holds). Similar to Fig. 6a, $\tau_{1,2}$ is pre-fetched but its residency time decreases from 150 ms in the previous scheduling table to 100 ms, thus indicating an increase in reliability. The result shows a considerable improvement in the MTTF of the system:

$$R(\mathcal{T})_{\text{Sys}} \approx 0.999998718, \;\; \text{MTTF}_{\text{Sys}} \approx 780031\mathcal{T}(73.3\% \text{ increase})$$

7 Simulation results and comparative study

The efficiency of the proposed hybrid scheduling technique is evaluated by comparing it with the static FT EDF-NP scheduling policy without backup overloading. To achieve a fair comparison, the current study also adopted the Pareto-based technique in Ref. [13] for its platform. The Pareto-based technique conducts an exhaustive search for the optimal scheduling table (with the highest reliability) of all tasks coupled with their backups at design time. Some tasks may not have any backups, while others may have one (DWC) or two (TMR) active backups. While in the Pareto-based technique all backups are active, in the hybrid technique proposed in this paper, backup tasks can be either active or passive. Scheduling passive backups offers a considerable advantage: There is no need to execute a backup if the primary successfully finishes its execution. Such abortions avoid unnecessary runs. This also indicates that slack areas, for the most part, will be overloaded with different backup tasks. The present work's simulation experiments demonstrate the advantage of passive backup overloading as well.

Table 3 Estimated SERs for different orbits and solar conditions

Orbit	SEUs/bit/day (XUPV5LX110T)		
	Solar max	Worst week	Worst day
GEOsynchronous (GEO)	6.09E−08	6.47E−05	3.35E−04
Global positioning system (GPS)	6.09E−08	5.71E−05	2.89E−04
MOLniya (MOL)	3.01E−07	6.09E−05	3.12E−04
POLar (POL)	2.25E−07	1.33E−05	7.99E−05

In the current study, 3000 real-world inspired task sets were first generated, each of which contains 5 to 20 independent periodic tasks. These include H.264 and MP3 task sets consisting of 10 tasks [47] with a hyperperiod less than the integer-max-size; the task sets with a hyperperiod exceeding 2.14E09 ms have been dismissed to keep the simulation time within reasonable bounds. Task computation times are set in the range of 10–500 ms [48] and selected task width and height are between [7…42] CLBs [49]. As in the examples mentioned in Sects. 5 and 6.2, the hardware tasks of the present work's experiments consist of CLBs. Nevertheless, the proposed hybrid scheduling technique is scalable to other tasks that exploit other types of FPGA resources.

In our experiments, the architecture of a realistic partial FPGA is assumed to model RD and task sets are generated to be emulated on it. In this particular case, the Xilinx™ Virtex-5 XUPV5LX110T FPGA has been employed [50]. This FPGA features 160 rows and 54 columns of CLBs. In total, there are eight CLB groups per column, each of which has 47,520 bits in the bitstream (36 frames × 1320 SRAM bits). As previously mentioned, the percentage of sensitive bits in the bitstream (S_i) is considered to be no more than 35% to mimic the characteristics of real-world tasks [36]. Fairly extensive experiments were performed on this FPGA by our laboratory members in Ref. [13] to determine the average time for configuring each CLB group. According to their results, each CLB group takes, on the average, 3.53 ms to be completely configured. Hence, RD = (160, 54, 20, 3.53 ms) [Eq. (4)].

According to the present study's earlier discussion, the SER may be estimated as a number of SEUs per bit per time unit. Based on the values of SEs reported in Ref. [37], and with regard to different satellite orbits and solar conditions, fellow researchers [13] have employed CREME96 tools [51] to estimate SEs (Table 3). Even though Table 3 provides SERs [Λ in Eq. (8)] for the Virtex-5 XUPV5LX110T FPGA corresponding to different satellite orbits, the proposed hybrid scheduling technique can be used for other SER values corresponding to other real space environments. For the proposed target FPGA, the SER range was determined by the SERs lower and upper bounds from Table 3 (6.09E−08 and 3.35E−04). Finally, with the independent RT tasks and SER, Eqs. (8), (10), and (12) could provide the total MTTF of the system before and after applying the FT techniques.

With scheduling possible backup jobs but without backup overloading, the present study's first set of experiments scheduled the randomly generated task sets with a static FT EDF-NP algorithm. Next, the proposed technique was applied to

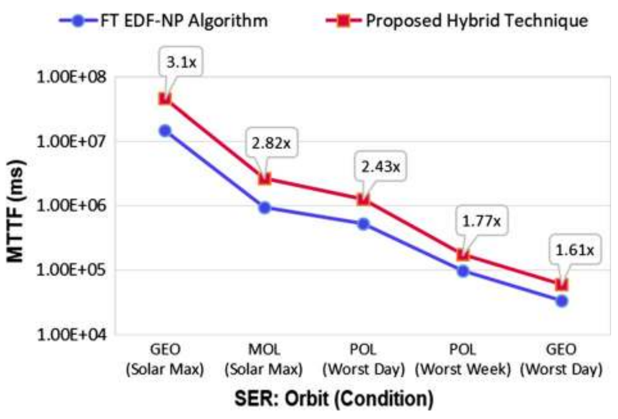

Fig. 7 Comparison of the proposed technique's MTTF with the static FT EDF-NP algorithm

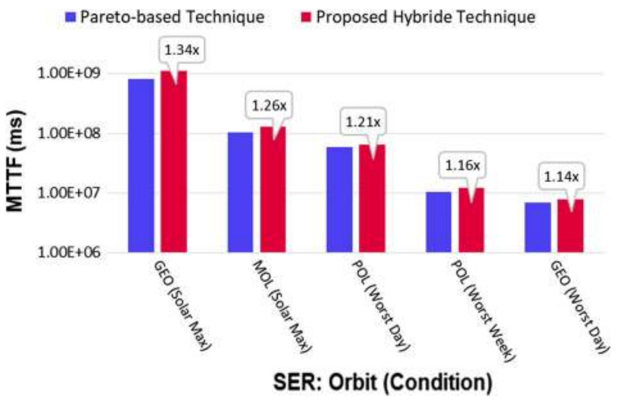

Fig. 8 Comparison of the proposed technique's MTTF with the Pareto-based FT scheduling technique [13]

schedule the same tasks as well as their probable backups and to overload slack time–space areas with candidate backup copies. The acquired results are presented as an average value of 100 runs for each task set.

As Fig. 7 indicates, the proposed hybrid technique outperforms the FT EDF-NP algorithm in terms of the system MTTF; in this set of experiments, the average MTTF rose 2.34 times. Task reliabilities increased because of applying the mentioned six principles (A1–A6) in Sect. 6.1. Principles A4 and A5 provided more slack areas on the FPGA. Consequently, there were more opportunities for scheduling backup jobs. Furthermore, by using Principles A1–A3 as well as A6, the scheduler selected proper candidate backup jobs (with lower reliabilities) to be overlapped in the slack areas.

A similar trend is observed in Fig. 8 where, in the second part of the experiments, the proposed technique improves the total MTTF of the system by an average factor of 1.22 in comparison with the Pareto-based FT technique. This increase factor is less than the 2.34 of the first part of the experiments. A plausible explanation is that the FT EDF-NP algorithm is much more basic than the FT scheduling algorithm proposed in Ref. [13], which conducts an exhaustive search for the optimal scheduling table at design time. Thus, the proposed technique better improves the MTTF of task sets scheduled by FT EDF-NP.

Although the Pareto-based technique statically finds the optimal scheduling table for all tasks coupled with their active backups (e.g., DWC and TMR) via an exhaustive search, the MTTF of the task sets scheduled by the proposed technique is higher. The reason for this result is very simple: Passive backup overloading in the slack areas outperforms scheduling tasks as active backups. In other words, in the Pareto-based technique, active backups are always executed even for no-failure (fault-free) cases.

In the proposed technique, by contrast, the slack areas are overloaded with passive backups and the event-triggered dispatcher dynamically decides which backup should be executed at run time (backups remain dormant until their primaries fail). If a primary task successfully finishes its execution, the execution of all its passive backup copies is immediately canceled (Lines 9–11 in Algorithm 2). Then, another candidate backup, with the lowest reliability and without a priori knowledge of whether its primary fails or not, is then re-configured (Line 12 in Algorithm 2). Such abortions, which avoid unnecessary runs, point out the superiority of passive backup tasks in the reliability rise. Nonetheless, when the scheduling table is so tight that the scheduler cannot find enough slack area and time to schedule tasks as passive backups or when tasks are very low-laxity, then tasks are inevitably scheduled as active backups.

By looking further at Figs. 7 and 8, it can be seen that the magnitude of MTTF falls as the value of SER rises. According to Table 3, the lowest and the highest SER occur in the GEO orbit in Solar Max and Worst Day conditions, respectively. This decline in MTTF is quite logical owing to the exponential relationship between the task reliability and task failure rate (Eqs. 7–12 in Sect. 3.3). It is important to note that the MTTF ($\mathcal{T}$) is a logarithmic function. However, MTTF values for the proposed technique are rather different in Figs. 7 and 8, because the FPGA has a higher utilization factor in the first set of experiments (Fig. 7). It was observed that the task set with a higher FPGA utilization factor virtually had a lower MTTF [Eq. (5)].

The experimental results depicted in Fig. 9 show how the task set MTTF decreases with an increasing FPGA utilization factor. Similar to the concept of multiprocessor systems, the higher the FPGA utilization, the less slack areas provided by the FPGA. Thus, probably a few numbers of tasks can have backups. Moreover, higher FPGA utilization means that the total task computation time over a hyperperiod rises, consequently yielding to lower reliability [Eqs. (10), (11)].

The present study conducted another set of experiments in which two principles (Principles A4 and A5) were ignored and for which Fig. 10 provides the results. As mentioned in Sect. 6.1, Principle A4 suggests that tasks with lower reliabilities should be scheduled as soon as possible. With the removal of Line 5 from Algorithm 1, this principle was ignored. As a result, the MTTF decreased between 20 and 40% in regard to the SER. Next, only Principle A5 (job pre-fetch) was ignored

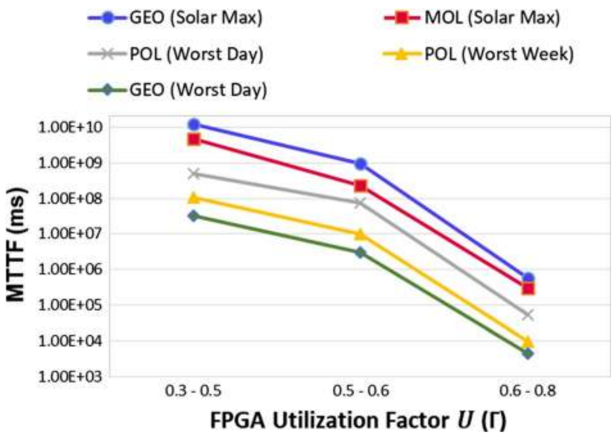

Fig. 9 The relationship between FPGA utilization factor and MTTF

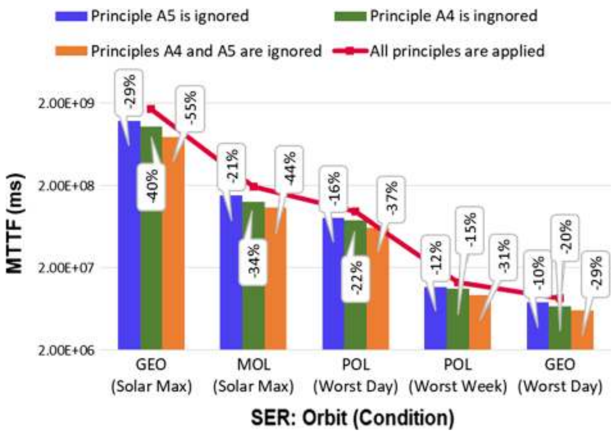

Fig. 10 The impact of applying Principles A4 and A5 on the MTTF of task sets

by removing Line 6 from Algorithm 1. Consequently, the MTTF decreased between 10 and 29%. These two experiments indicate that Principle A4 is more important than Principle A5. In the end, though, both Principles A4 and A5 were ignored. As shown in Fig. 10, the experiments show an MTTF decline between 29 and 55%.

8 Conclusion and potential future work

It is patently obvious that RT embedded systems are widely utilized in critical applications, such as safety and mission critical systems. As the most prominent RDs, SRAM-based FPGAs are employed in many such systems. Nevertheless, SRAM-based FPGAs are well-known for their susceptibility to transient and intermittent faults mainly caused by high-energy neutrons, protons, or heavy ions. As a result, SRAM-based FPGAs need protection. Generally, FT scheduling complements the enormous redundancy of achieving higher reliability as an effective solution. Consequently, the need for FT scheduling will exponentially grow in the years ahead.

The current paper outlines a hybrid technique for the RT FT scheduling of independent periodic tasks. The proposed technique schedules tasks in two main phases: a static phase and a dynamic phase. In the static phase, all jobs are scheduled according to the EDF-NP policy. Next, according to the mentioned principles (Principles A1–A6), jobs are reordered (re-scheduled). Furthermore, candidate backup jobs are selected to overload possible slack areas. Then, in the dynamic phase, an event-triggered dispatcher decides which candidate backup job should be configured and executed on FPGA at run time. Experimental results show that applying the proposed principles to the scheduling algorithm enhances the MTTF of the system by an average factor of 1.22 in comparison with the previous strongly static fault tolerance study.

While the focus of this paper has been on calculating task reliability after configuration, assessing the reliability of tasks during configuration is also an important yet complex novel issue for future research. To solve this problem, the configuration process of hardware tasks must be studied in detail to calculate task reliability at any specific time in the course of the configuration process. In addition, not only does task size affect the configuration delay time of any hardware task but so do the configured and currently running tasks on FPGA. Moreover, the partial reconfiguration view involves many constraints such as routing, location, and macro blocks. Considering a general hardware model for reliability estimation and scheduling purposes in such a way that its parameters can be easily adapted to the architecture of existing well-known FPGAs, researchers can provide more real conditions in ongoing works.

From the point of view of importance level, the same digital system might execute both soft and hard real-time tasks. In other words, the functionalities of different levels of criticalities coexist in many embedded systems. Moreover, such complex RT systems are usually subject to sudden changes in their environments. That is, tasks criticality levels can change with time. Recently, significant research attention has been directed to mixed-critical (MC) systems that can be found in applications such as avionics and automobiles. The FT scheduling of RT tasks in MC systems that use FPGAs is another interesting—indeed absolutely crucial—issue that has not yet attracted the attention it deserves.

As a growing area of research, multi-FPGA systems offer high-performance solutions to highly complex computing tasks. Furthermore, in multi-FPGA systems, it is possible to schedule big tasks that cannot be configured on a single FPGA. However, the scheduling issue on the multi-FPGA platform is more complex than that on a multiprocessor because of FPGA limitations on configuration management. The problem becomes even more complicated when the system consists of FPGAs with different characteristics. In such systems, tasks can have different computation/configuration times on different FPGAs. Moreover, the task failure rate differs from one FPGA to another. Hence, as another interesting issue for future work, RT FT scheduling on multi-FPGA systems will call for an in-depth study as it raises many unanswered questions.

Needless to say, current critical embedded systems that employ an FPGA as a PE inherently have severe limitations on available resources and power capability. Therefore, another major problem in FT scheduling on FPGAs is power and energy consumption. This is especially the case when the number of active backups increases. Reliability-aware power management and thermal-aware fault-tolerant scheduling on FPGAs are other interesting research topics that deserve great attention. That is to say, as the workload on FPGA increases, static and dynamic power consumption along with the temperature increase and increase in the FPGA temperature results in the SER increase. Consequently, studying the impact of scheduling itself on the reliability of system can be one of the most critical issues to enhance future studies.

As can be deduced from the above discussion, there already exists a noteworthy body of potential work in the area of FT scheduling of RT embedded systems utilizing SRAM-configured FPGAs.

Acknowledgements The authors wish to sincerely acknowledge and thank Dr. Reza Ramezani, assistant professor at the University of Isfahan. He, generously, provided many constructive comments that greatly assisted the research.

References

1. Cardoso J, Hübner M (2011) Reconfigurable computing: from FPGAs to hardware/software codesign. Springer, Berlin. https://doi.org/10.1007/978-1-4614-0061-5
2. Cetin E, Diessel O, Li T, Ambrose JA, Fisk T, Parameswaran S, Dempster AG (2016) Overview and investigation of SEU detection and recovery approaches for FPGA-based heterogeneous systems. In: Rech P (ed) FPGAs and parallel architectures for aerospace applications. Springer, Berlin, pp 33–46. https://doi.org/10.1007/978-3-319-14352-1_3
3. Vipin K, Fahmy SA (2018) FPGA dynamic and partial reconfiguration: a survey of architectures, methods, and applications. ACM Comput Surv (CSUR) 51:72. https://doi.org/10.1145/3193827
4. Kean T, Buchanan I (1992) The use of FPGAs in a novel computing subsystem. Paper Presented at the Proceeding of 1st International ACM/SIGDA Workshop on FPGAs
5. Hauck S (1998) The roles of FPGA's in reprogrammable systems. Proc IEEE 86:615–638. https://doi.org/10.1109/5.663540
6. Koch D, Ziener D, Hannig F (2016) FPGA versus software programming: why, when, and how? In: FPGAs for Software Programmers. Springer, Berlin, pp 1–21. https://doi.org/10.1007/978-3-319-26408-0_1
7. Parrilla L, Álvarez-Bermejo JA, Castillo E, López-Ramos JA, Morales-Santos DP, García A (2018) Elliptic Curve Cryptography hardware accelerator for high-performance secure servers. J Supercomput. https://doi.org/10.1007/s11227-018-2317-6
8. Kastensmidt FL, Carro L, da Luz Reis RA (2006) Fault-tolerance techniques for SRAM-based FPGAs. Springer, Berlin. https://doi.org/10.1007/978-0-387-31069-5
9. Bolchini C, Miele A, Sandionigi C (2013) Autonomous fault-tolerant systems onto SRAM-based FPGA platforms. J Electron Test 29:779–793. https://doi.org/10.1007/s10836-013-5418-4
10. Zhao Z, Nguyen NT, Agiakatsikas D, Lee G, Diessel O (2018) Fine-grained module-based error recovery in FPGA-based TMR systems. ACM Trans Reconfigurable Technol Syst 11:4. https://doi.org/10.1145/3173549
11. Kastensmidt F, Rech P (2016) Radiation effects and fault tolerance techniques for FPGAs and GPUs. In: Rech P (ed) FPGAs and parallel architectures for aerospace applications. Springer, Berlin, pp 3–17. https://doi.org/10.1007/978-3-319-14352-1_1
12. Krishna C (2014) Fault-tolerant scheduling in homogeneous real-time systems. ACM Comput Surv (CSUR). 46:48. https://doi.org/10.1145/2534028

13. Ramezani R, Sedaghat Y, Naghibzadeh M, Clemente JA (2017) Reliability and makespan optimization of hardware task graphs in partially reconfigurable platforms. IEEE Trans Aerosp Electron Syst. https://doi.org/10.1109/TAES.2017.2667338
14. Liang H, Sinha S, Zhang W (2018) Parallelizing hardware tasks on multicontext FPGA with efficient placement and scheduling algorithms. IEEE Trans Comput Aided Design Integr Circuits Syst 37:350–363. https://doi.org/10.1109/TCAD.2017.2697952
15. Stoddard A, Gruwell A, Zabriskie P, Wirthlin MJ (2017) A hybrid approach to FPGA configuration scrubbing. IEEE Trans Nucl Sci 64:497–503. https://doi.org/10.1109/TNS.2016.2636666
16. Zhang H, Kochte MA, Imhof ME, Bauer L, Wunderlich H-J, Henkel J (2014) GUARD: Guaranteed reliability in dynamically reconfigurable systems. Paper presented at the Proceedings of the 51st Annual Design Automation Conference. https://doi.org/10.1145/2593069.2593146
17. Santos R, Venkataraman S, Kumar A (2017) Scrubbing mechanism for heterogeneous applications in reconfigurable devices. ACM Trans Design Autom Electron Syst 22:33. https://doi.org/10.1145/2997646
18. Giordano R, Perrella S, Izzo V, Milluzzo G, Aloisio A (2017) Redundant-configuration scrubbing of SRAM-based FPGAs. IEEE Trans Nucl Sci 64:2497–2504. https://doi.org/10.1109/TNS.2017.2730960
19. Sterpone L, Violante M (2006) A new reliability-oriented place and route algorithm for SRAM-based FPGAs. IEEE Trans Comput. https://doi.org/10.1109/TC.2006.82
20. Huang K, Hu Y, Li X (2014) Reliability-oriented placement and routing algorithm for SRAM-based FPGAs. IEEE Trans Very Large Scale Integr (VLSI) Syst 22:256–269. https://doi.org/10.1109/TVLSI.2013.2239318
21. Bolchini C, Miele A, Sandionigi C (2011) A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. IEEE Trans Comput 60:1744–1758. https://doi.org/10.1109/TC.2010.281
22. Tambara LA, Almeida F, Rech P, Kastensmidt FL, Bruni G, Frost C (2015) Measuring failure probability of coarse and fine grain TMR schemes in SRAM-based FPGAs under neutron-induced effects. Paper presented at the International Symposium on Applied Reconfigurable Computing. https://doi.org/10.1007/978-3-319-16214-0_28
23. Yang M, Hua G, Feng Y, Gong J (2017) Fault-tolerance techniques for spacecraft control computers. Wiley, London. https://doi.org/10.1002/9781119107392
24. Xie G, Zeng G, Chen Y, Bai Y, Zhou Z, Li R, Li K (2017) Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2017.2665552
25. Pathan RM (2017) Real-time scheduling algorithm for safety-critical systems on faulty multicore environments. Real-Time Syst 53:45–81. https://doi.org/10.1007/s11241-016-9258-z
26. Kopetz H (2011) Real-time systems: design principles for distributed embedded applications. Springer, Berlin. https://doi.org/10.1007/978-1-4419-8237-7
27. Pathan RMJR-TS (2014) Fault-tolerant and real-time scheduling for mixed-criticality systems. Real-Time Syst 50:509–547. https://doi.org/10.1007/s11241-014-9202-z
28. Kim J, Lakshmanan K, Rajkumar R (2010) R-BATCH: task partitioning for fault-tolerant multiprocessor real-time systems. Paper presented at the 2010 IEEE 10th International Conference on Computer and Information Technology (CIT). https://doi.org/10.1109/CIT.2010.321
29. Zhu X, Wang J, Guo H, Zhu D, Yang LT, Liu L (2016) Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. IEEE Trans Parallel Distrib Syst 27:3501–3517. https://doi.org/10.1109/TPDS.2016.2543731
30. Löfwenmark A, Nadjm-Tehrani S (2018) Fault and timing analysis in critical multi-core systems—a survey with an avionics perspective. J Syst Archit. https://doi.org/10.1016/j.sysarc.2018.04.001
31. Yin J-Y, Guo G-C, Wu Y-X (2009) A hybrid fault-tolerant scheduling algorithm of periodic and aperiodic real-time tasks to partially reconfigurable FPGAs. Paper presented at the 2009 ISA 2009 International Workshop on Intelligent Systems and Applications. https://doi.org/10.1109/IWISA.2009.5072624
32. Yin J, Zheng B, Sun Z (2012) A hybrid real-time fault-tolerant scheduling algorithm for partial reconfigurable system. JCP 7:2773–2780. https://doi.org/10.4304/jcp.7.11.2773-2780
33. Ramezani R, Sedaghat Y, Clemente JA (2017) Reliability improvement of hardware task graphs via configuration early fetch. IEEE Trans Very Large Scale Integr (VLSI) Syst 25:1408–1420. https://doi.org/10.1109/TVLSI.2016.2631724

34. Say F, Bazlamaççı CF (2012) A reconfigurable computing platform for real time embedded applications. Microprocess Microsyst 36:13–32. https://doi.org/10.1016/j.micpro.2011.08.013
35. Herrera-Alzu I, Lopez-Vallejo M (2014) System design framework and methodology for Xilinx Virtex FPGA configuration scrubbers. IEEE Trans Nucl Sci 61:619–629. https://doi.org/10.1016/j.micpro.2011.08.013
36. Monson JS, Wirthlin M, Hutchings B (2012) A fault injection analysis of Linux operating on an FPGA-embedded platform. Int J Reconfigurable Comput 2012:7. https://doi.org/10.1155/2012/850487
37. Ramezani R, Sedaghat Y, Naghibzadeh M, Clemente JA (2018) A decomposition-based reliability and makespan optimization technique for hardware task graphs. Reliab Eng Syst Saf 180:13–24. https://doi.org/10.1016/j.ress.2018.07.007
38. Clemente JA, Resano J, González C, Mozos D (2011) A hardware implementation of a run-time scheduler for reconfigurable systems. IEEE Trans Very Large Scale Integr (VLSI) Syst 19:1263–1276. https://doi.org/10.1109/TVLSI.2010.2050158
39. Bushnell M, Agrawal V (2004) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits. Springer, Berlin. https://doi.org/10.1007/b117406
40. Ramezani R, Clement JA, Sedaghat Y, Mecha H (2016) Estimation of hardware task reliability on partially reconfigurable FPGAs. Paper presented at the 2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS). https://doi.org/10.1109/RADECS.2016.8093184
41. Mottaghi MH, Zarandi HR (2014) DFTS: a dynamic fault-tolerant scheduling for real-time tasks in multicore processors. Microprocess Microsyst 38:88–97. https://doi.org/10.1016/j.micpro.2013.11.013
42. Hazucha P, Svensson C (2000) Impact of CMOS technology scaling on the atmospheric neutron soft error rate. IEEE Trans Nucl Sci 47:2586–2594. https://doi.org/10.1109/23.903813
43. Koren I, Krishna CM (2010) Fault-tolerant systems. Morgan Kaufmann, Burlington. https://doi.org/10.1016/b978-0-12-088525-1.x5000-7
44. Namazi A, Safari S, Mohammadi S (2018) CMV: clustered majority voting reliability-aware task scheduling for multicore real-time systems. IEEE Trans Reliab. https://doi.org/10.1109/TR.2018.2869786
45. Kuo W, Prasad VR (2000) An annotated overview of system-reliability optimization. IEEE Trans Reliab 49:176–187. https://doi.org/10.1109/24.877336
46. Haahr M (2019) RANDOM.ORG: True Random Number Service. https://www.random.org. Accessed Sept 2018
47. Clemente JA, Beretta I, Rana V, Atienza D, Sciuto D (2014) A mapping-scheduling algorithm for hardware acceleration on reconfigurable platforms. ACM Trans Reconfigurable Technol Syst (TRETS) 7:9. https://doi.org/10.1145/2611562
48. Danne K, Platzner M (2006) An EDF schedulability test for periodic tasks on reconfigurable hardware devices. Paper presented at the ACM SIGPLAN Notices. https://doi.org/10.1145/1159974.1134665
49. Steiger C, Walder H, Platzner M, Thiele L (2003) Online scheduling and placement of real-time tasks to partially reconfigurable devices. Paper presented at the RTSS 2003. 24th IEEE Real-Time Systems Symposium. https://doi.org/10.1109/REAL.2003.1253269
50. XilinxCorporation Virtex-5 FPGA Configuration User Guide, UG191 (v 3.11). http://www.xilinx.com/support/documentation/user_guides/ug191.pdf. Accessed Sept 2018
51. Tylka AJ, Adams JH, Boberg PR, Brownstein B, Dietrich WF, Flueckiger EO, Petersen EL, Shea MA, Smart DF, Smith EC (1997) CREME96: a revision of the cosmic ray effects on micro-electronics code. IEEE Trans Nucl Sci 44:2150–2160. https://doi.org/10.1109/23.659030