WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Review of RASA: Reliability-Aware Scheduling for FPGA-Based Resilient Embedded Systems in Extreme Environments

Aditya Priyadarshi, Sumana S.

Dept. of EEE, Dayananda Sagar College of Engg.,
Email: priyadarshiaditya36@gmail.com, sumana-eee@dayanandasagar.edu.in

ABSTRACT-

Field-Programmable Gate Arrays (FPGAs) are a key element of real-time embedded systems for extreme environments such as space and nuclear power plants, where radiation-induced faults like Single Event Upsets (SEU) undermine reliability. Reliability-Aware Scheduling Approach (RASA) suggests a hybrid offline-online scheduling approach to guarantee task execution in the presence of such faults. This review examines the methodology of Reliability-Aware Scheduling Approach, using partial reconfiguration and preemptive scheduling to achieve a 72% Task Success Ratio with high workloads and fault rates. Compared to conventional methods such as scheduling Triple Modular Redundancy (TMR) and Earliest-Deadline First (EDF), Reliability-Aware Scheduling Approach provides lower overhead and improved efficiency. Strengths are flexibility and new metrics, and weaknesses are simulation-based verification and hard ware specificity. Future work suggests real-world experimentation and wider applicability. This paper sets the context for the applicability of Reliability-Aware Scheduling Approach to rugged systems in extreme environments.

Keyword- *Reliability-Aware Scheduling Approach, Field-Programmable Gate Array, reliability-aware scheduling, extreme environments, real-time systems, fault tolerance, Single-Event Upsets*

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) integrate software flexibility with hard- ware performance, enabling real-time task execution in domains like avionics, robotics, and space exploration [1]. In extreme environments—characterized by high radiation ($\lambda \in [10{-}2, 102]$ faults/hour), temperature variations (−55◦C to 125◦C), and electromagnetic interference—FPGAs face reliability challenges from Single- Event Upsets (SEUs). SEUs induce bit flips in configuration memory (CM), altering logic, routing, or state, with a probability modeled as a Poisson process ($P(\text{fault}) = 1 - e^{-\lambda t}$) [7].

Traditional fault-tolerant methods, such as TMR (200% area overhead) and bitstream scrubbing (18 ms full reconfiguration latency), are resource-intensive [5, 8]. RASA, introduced by Saha et al. [1], addresses these limitations through a hybrid offline-online scheduling approach for partially reconfigurable FPGAs. Achieving a 72% TSR at 70% utilization under high fault rates, RASA outperforms conventional methods. This review provides a technical analysis of RASA's methodology, performance, strengths, limitations, and future directions.

2. BACKGROUND AND RELATED WORK

2.1 FPGA-based systems in extreme environments

FPGAs, such as the Xilinx Virtex-5, support dynamic partial reconfiguration, partitioning the FPGA into Partially Reconfigurable Regions (PRRs) for independent task execution [4]. In extreme environments, SEUs disrupt CM, with fault rates scaling inversely with feature size ($\lambda \propto 1/\text{nm2}$) [1]. For example, in nuclear inspection robots, SEUs corrupt architectural registers, violating real-time constraints. Radiation-hardened FPGAs mitigate this but incur 50% performance penalties and high costs [6].

2.2 Evolution of fault tolerant techniques

2.2.1 Hardware redundancy:

Triple Modular Redundancy (TMR) and Duplication with Compare (DWC) are traditional hardware-based methods [8]. TMR replicates a module three times, using voting to mask faults, while DWC replicates it to compare. Jacobs et al. [8] demonstrated their effectiveness in low-fault environments, but both have over 200% area overhead and are power-expensive [1]. In high-fault rate extreme environments, these non-preemptive methods do not work

because faulty tasks need to be replayed from the beginning, wasting resources [1]. TMR's static allocation limits adaptability, making it less suitable for dynamic real-time systems [1].

2.2.2 Bitstream scrubbing:

Scrubbing extends FPGA reconfigurability to SEU recovery by reloading the configuration memory [9]. Sari et al. [9] combined scrubbing with checkpoint/rollback, keeping hardware costs but introducing timing overheads from configuration memory reads at periodic intervals [1]. Psarakis and Sari [10] extended this to EDF scheduling, enhancing utilization at low fault rates. But misestimating scrubbing timing violates real-time deadlines, a challenge overcome by RASA via targeted partial reconfiguration [1].

2.2.3 Time redundancy:

Time-based methods like task re-execution and recovery with checkpoints offer alternatives in resource-constrained systems [11]. On multicore processors, these are effective but challenging for FPGAs due to context-switching complexities [1]. Happe et al. [12] showed that hardware context switching via bitstream readback through Internal Configuration Access Ports (ICAP) is feasible, yet its overhead remains a bottleneck for non-preemptive schemes [1].

2.3 Scheduling approaches for real-time FPGA systems

Real-time scheduling of tasks on FPGAs has been challenging since the task is NP-hard, just like multicore scheduling issues [13]. The development started with non-preemptive solutions like SJF and FCFS, as in [14]. They are easy but cause inefficient use of resources with variable deadlines or use of tasks, especially with more than one PRR [1]. Saha et al. say that non-preemptive schedules are not used in Extreme environments, as fault-driven interrupts demand flexibility [1].

Preemptive scheduling is robust in utilization but immature on FPGA platforms due to hardware context-switching complexities [12]. Saha et al. [15] presented an effective scheduling technique for partially reconfigurable systems, yet its synchronous reconfiguration (18 ms overhead) is impractical for continuous operation, in contrast to the flexibility of partial reconfiguration [1]. Psarakis and Sari [10] merged EDF with scrubbing, having greater utilization than static methods, yet its fixed-priority model leaves PRRs idle post-task completion, reducing efficiency on faults [1].

2.4 Fault models and detection

Fault models for extreme environments primarily model SEUs as transient with a Poisson distribution with rate λ [7]. Haque et al. [16] suggested burst fault models, but they are uncommon in extreme environments [1]. Mechanisms for detection are distinct: sanity checks, memory violation monitoring, control flow checks, and hardware duplication all catch some fault types, but not random faults in general [1]. RASA employs sanity checks in checkpoints, at the cost of simplicity and efficiency [1].

2.5 Positioning rasa

RASA builds on these foundations, addressing gaps in overhead, adaptability, and real-time performance. Unlike TMR's resource intensity or EDF's idle PRRs, RASA uses preemptive scheduling and partial reconfiguration to optimize utilization and recovery [1]. Compared to fully reconfigurable scheduling [15], it reduces overhead and against scrubbing-based EDF [10], it avoids ICAP conflicts, enhancing reliability in high-fault scenarios [1]. This positions RASA as a significant advancement for FPGA-based resilient systems in extreme environments.

3. RASA METHODOLOGY

RASA employs a hybrid offline-online scheduling framework to manage peri- odic real-time tasks on partially reconfigurable FPGAs, ensuring reliability un- der high fault rates ($\lambda \leq 0.03$). It leverages the Xilinx Virtex-5's dynamic par- tial reconfiguration, dividing the FPGA into NPRR Partially Reconfigurable Regions (PRRs), each capable of independent task execution. The methodology comprises two phases: RASA-Offline, which pre-computes a preemptive sched- ule, and RASA-Online, which executes the schedule with fault detection and recovery.

3.1 RASA-Offline

RASA-Offline constructs a preemptive schedule for a task set {T1, T2, . . . , Tn}, where each task Ti is defined by its execution time Ci, deadline Di, and period Pi = Di. The schedule spans a hyper period H = LCM(Di), ensuring all task instances are accounted for. The algorithm proceeds as follows:

3.1.1 Task Weight Calculation

Each task's weight reflects its resource demand:

$$wi = \frac{Ci}{Di}$$

representing the fraction of execution time required per unit time.

3.1.2 Workload Quota Assignment

For a time window W, typically a scheduling slot within H, the workload quota for task Ti is:

$$Qi = [wi \,.W]$$

Ensuring sufficient time allocation. The total work load across all tasks must satisfy:

$$\sum_{i=1}^{n} Qi \leq W \,.N_{PRR}$$

where N_{PRR} is the number of PRRs

3.1.3 Slack Computation

Available slack accounts for reconfiguration overhead:

$$S = W - T_{reconfig}$$

where $T_{reconfig}$ = 18ms for full reconfiguration. Partial reconfiguration, used by RASA, reduces this to 5 ms per PRR.

3.1.4 PRR Allocation

Tasks are assigned to PRRs using a first-fit strategy, starting with PRR-1. For each PRR, the scheduler ensures:

$$\sum_{Ti \,\epsilon\, PRR_j} Qi \leq W$$

preventing over-allocation. Preemption is enabled by saving task contexts via bitstream readback through the Internal Configuration Access Port (ICAP)

3.1.5 Checkpoint Placement

Detection and Recovery (DAR) checkpoints are inserted to monitor task execution. A checkpoint is placed midway through a task's execution if:

$$Qi \geq K \,.T_{DAR}$$

where TDAR = 5ms and K ∈ [3,10] is empirically tuned based on fault rate and task granularity. Otherwise, a checkpoint is placed at task completion. The DAR interval is:

$$I_{DAR} = \frac{Qi}{[{}^{Qi}/_{T_{DAR}}]}$$

ensuring even distribution. To avoid ICAP conflicts, reconfiguration events are staggered using a priority queue, with a maximum ICAP bandwidth of 400 MB/s.

3.1.6 Schedule Validation

The schedule is validated to ensure no deadline misses, using a feasibility test:

$$\sum_{i=1}^{n} \frac{C_i'}{D_i} \leq N_{PRR}$$

Where $C_i' = C_i + T_{DAR} \,.[{}^{C_i}/_{I_{DAR}}]$ accounts for DAR overhead.

The offline phase optimizes for high utilization while reserving slack for on line fault recovery, balancing computational complexity $(\vartheta(n \,.N_{PRR} \,.H))$ with schedulability.

3.2 RASA-Online

RASA-Online executes the offline schedule, dynamically handling faults through DAR checkpoints. It operates in a feedback loop involving detection, anomaly assessment, and recovery, leveraging partial reconfiguration to minimize latency. The process is detailed as follows:

3.2.1 Execution and Monitoring

Tasks are executed on assigned PRRs accord ing to the offline schedule. At each DAR checkpoint, the system performs bitstream readback via ICAP to retrieve the task's context, including program counter and register states.

3.2.2 Fault Detection

The Anomaly Factor (AF) quantifies execution discrepancies:

$$AF = I_{expected} - I_{actual}$$

where $I_{expected}$ is the expected number of instructions completed (based on clock cycles and instruction rate) and I_{actual} is the actual number, derived from context analysis. A positive AF indicates an SEU – induced fault.

3.2.3 Recovery Decision

Recovery is triggered based om AF and available slack:

$$S = W - \sum T_{exec} - T_{reconfig}$$

where T_{exec} is the time already consumed. Two scenarios are considered:

- Scenario 1 ($AF \leq S$): The task continues on the same PRR, using slack to complete execution. The remaining time is:

$$T_{remain} = AF \,.\, T_{instr}$$

Where T_{instr} is the average instruction execution time.

- Scenario 2 ($AF > S$): If the slack is insufficient, the task is either:
    - Migrated to a PRR with sufficient slack, identified via a slack table:

$$S_j = W - \sum_{Ti \,\epsilon\, PRR_j} Qi$$

where S_j is the slack on PRR_j

    - Accelerated by increasing the clock frequency, quantified by the Rise – Up Factor (RUF):

$$RUF = \frac{AF \,.\, T_{instr}}{S + T_{reconfig}}$$

ensuring completion within the deadline. The new frequency is:

$$f_{new} = f_{base} \,.\, (1 + RUF)$$

capped at the FPGA's maximum frequency

3.2.4 Partial Reconfiguration

Recovery involves scrubbing the affected PRR using partial bitstreams, with a latency of 5 ms per PRR for a 4-tiled FPGA. The bitstream size per PRR is:

$$B_{PRR} = \frac{B_{total}}{N_{PRR}}$$

where $B_{total} \approx 10$MB for Virtex-5. The ICAP transfers data at 400 MB/s, ensuring:

$$T_{reconfig} = \frac{B_{PRR}}{400} \approx 5ms$$

3.2.5 Context Restoration

Post – reconfiguration, the task's context is restored from the last checkpoint, resuming execution. The context size is typically 1-2 KB, with a readback latency of 0.1 ms.

RASA - Online's fault tolerance is enhanced by its ability to prioritize recovery actions based on deadline proximity, using a heuristic that minimizes deadline misses:

$$Priority\ (T_i) = \frac{1}{D_i - t_{current}}$$

This ensures critical tasks are recovered first, maintaining real – time guarantees under high fault rates.

4. PERFORMANCE EVALUATION

RASA's performance was rigorously evaluated through simulations on a Xilinx Virtex-5 FPGA, configured with 4 or 8 Partially Reconfigurable Regions (PRRs), using a custom simulator modeling SEU fault with a Poisson distribution ($\lambda \in [0.005, 0.03]$ faults/unit time). The simulation environment emulated a real-time embedded system with periodic tasks ($n \in [10,50]$), each with execution times $C_i \in [5,20]$ ms and deadlines $D_i \in [20,100]$ ms. Key metrics included Task Success Ratio (TSR), Reliability Cost, and Reconfiguration Overhead, assessed across varying system utilizations (50%–80%) and fault rates. The setup incorporated realistic FPGA parameters, such as a 400 MB/s ICAP bandwidth and 5 ms partial reconfiguration latency per PRR, to ensure fidelity to hardware constraints.

4.1 Task Success Ratio (TSR)

The Task Success Ratio (TSR) measures the percentage of tasks meeting their deadlines:

$$TSR = \frac{Task\ Accepted}{Task\ Scheduled} .100\%$$

At 70% system utilization and a fault rate of $\lambda = 0.01$, RASA achieved a TSR of 72%, significantly outperforming EDF-based scrubbing (50% TSR at 60% utilization) and fully reconfigurable systems (55% TSR) [10,15]. Scaling to 8 PRRs reduced DAR overhead from 5 ms to 3 ms, improving TSR by 10% due to lower contention for ICAP resources. At higher fault rates ($\lambda = 0.03$), TSR declined to 60%, reflecting increased task rejections as recovery actions saturated available slack. However, RASA's preemptive scheduling and partial reconfiguration maintained a 20% higher TSR than EDF, which suffered from idle PRRs post-task completion. Sensitivity analysis showed that TSR degraded gracefully with task count, dropping by 5% for $n = 50$ compared to $n = 10$, highlighting RASA's scalability.

4.2 Reliability Cost

The normalized reliability cost ($C_R \epsilon [0, 1]$) quantifies fault- induced overhead:

$$C_R = \frac{\sum Fault - Induced\ Reconfigurations}{Total\ Tasks}$$

At 50% utilization, RASA maintained $CR < 0.4$, compared to TMR's $CR > 0.7$ and DWC's $CR \approx 0.6$. This efficiency stems from RASA's targeted partial reconfiguration, which minimizes resource usage (5 ms/PRR vs. 18 ms for full reconfiguration). As task count increased to 50, CR rose to 0.5 due to higher fault susceptibility, but remained 30% lower than EDF-based scrubbing, which incurred frequent ICAP conflicts. At $\lambda = 0.03$, CR approached 0.6, reflecting the cost of frequent recoveries, yet RASA's dynamic slack allocation kept it below competitors. The trade-off between TSR and CR was evident at 80% utilization, where TSR dropped to 55% but CR remained stable, underscoring RASA's robustness.

4.3 Comparison with Existing Methods

RASA was benchmarked against TMR, DWC, EDF-based scrubbing, and fully reconfigurable scheduling across fault rates and utilizations. TMR and DWC, while effective at low fault rates ($\lambda < 0.005$), exhausted resources at $\lambda = 0.01$, achieving TSRs below 40% due to their non-preemptive nature and 200% area overhead [8]. EDF-based scrubbing, as in Psarakis and Sari [10], yielded 50% TSR at 60% utilization, limited by fixed-priority scheduling and periodic scrubbing overheads (10 ms/scan). RASA's 70% TSR at the same utilization resulted from its proportional fairness and ICAP conflict avoidance, reducing reconfiguration latency by 72% (5ms vs. 18ms) compared to fully reconfigurable systems.

5. STRENGHTS OF RASA

RASA's primary strength lies in its flexible and efficient use of partial reconfiguration, which enables asynchronous updates to PRRs with minimal downtime, achieving reconfiguration latencies as low as 5 ms compared to 18 ms for full reconfiguration. This capability, coupled with preemptive scheduling, allows RASA to dynamically adapt to fault-induced disruptions, maintaining a 72% TSR at high fault rates ($\lambda = 0.01$) and 70% system utilization. The approach optimizes resource utilization by leveraging available slack and adjusting clock frequencies through the Rise-Up Factor (RUF), ensuring tasks meet real-time deadlines even under constrained conditions. Unlike traditional methods like TMR, which incur a 200%area overhead, RASA's DAR mechanism introduces a low overhead of 5 ms per PRR, making it suitable for resource-constrained embedded systems in extreme environments.

The introduction of novel metrics, such as the Anomaly Factor (AF) and Reliability Cost (CR), further enhances RASA's technical merit by providing precise, quantifiable measures of fault impact and system reliability. These metrics enable engineers to assess performance under varying fault rates and workloads, offering insights into trade-offs between TSR and recovery overhead. RASA's adaptability is particularly notable in its ability to handle high-fault scenarios ($\lambda \leq 0.03$) through dynamic recovery strategies, such as task migration or frequency scaling, which ensure robustness without sacrificing

efficiency. This combination of low-latency reconfiguration, intelligent scheduling, and innovative metrics positions RASA as a significant advancement over conventional fault-tolerant techniques, particularly for FPGA-based systems operating in harsh conditions like space or nuclear facilities.

6. WEAKNESSES OF RASA

A significant limitation of RASA is its reliance on simulation-based validation, which, while comprehensive, lacks real-world testing in extreme environments such as radiation chambers or space missions. This gap raises concerns about its performance under actual SEU patterns, which may deviate from the Pois son model ($\lambda \in [0.005,0.03]$) used in simulations. The absence of physical FPGA implementation data limits confidence in RASA's applicability to mission-critical systems, where unmodeled factors like thermal variations or electromagnetic interference could impact reliability. Additionally, RASA's design is tailored to the Xilinx Virtex-5 FPGA, leveraging its specific partial reconfiguration architecture and ICAP bandwidth (400 MB/s). Its performance on other FPGA families, such as Intel Stratix or Microchip PolarFire, remains untested, potentially requiring significant modifications to PRR mapping or reconfiguration protocols, which could introduce unforeseen overheads.

Another critical weakness is the ICAP bottleneck, as the single ICAP on Virtex 5 restricts simultaneous PRR reconfiguration, leading to increased recovery latency at high fault rates($\lambda > 0.03$). This constraint can cause task rejections when multiple faults occur concurrently, reducing TSR by upto12% in extreme scenarios. Furthermore, the empirical tuning of the checkpoint parameter $K \in [3,10]$ complicates RASA's deployment across diverse workloads and platforms. The need for platform-specific optimization of K based on fault rate and task granularity introduces complexity, potentially hindering scalability and generalizability. These limitations underscore the need for further development to enhance RASA's robustness and portability in practical applications.

7. FUTURE DIRECTIONS

To address RASA's limitations and expand its applicability, a primary focus should be on real-world testing in extreme environments, such as radiation chambers simulating fault rates of $\lambda \sim 102$ faults/hour, to validate simulation results and uncover unmodeled behaviors. Such experiments would provide critical data on RASA's performance under realistic SEU patterns, thermal stresses, and electromagnetic interference, enabling refinements to its fault detection and recovery mechanisms. Additionally, adapting RASA for cross-platform compatibility is essential to broaden its adoption. This involves re-engineering its scheduling and reconfiguration algorithms for FPGA architectures like Intel Stratix, which use different PRR structures, or Microchip PolarFire, which offer enhanced radiation tolerance. These adaptations would require optimizing bitstream formats and ICAP-equivalent interfaces, potentially reducing reconfiguration latency by 10–20% on modern platforms.

Further advancements could explore sophisticated fault models, such as burst faults with probability $P(\text{burst}) \propto \lambda 2$, to handle rare but catastrophic scenarios in extreme environments [7]. Integrating machine learning techniques, such as neural networks for fault prediction or reinforcement learning for dynamic optimization of the checkpoint parameter K, would enhance RASA's adaptability to varying workloads and fault patterns. These AI-driven approaches could reduce recovery overhead by up to 15% by preemptively adjusting schedules based on predicted fault probabilities. By pursuing these directions, RASA can evolve into a more robust and versatile framework, capable of supporting a wide range of FPGA-based resilient systems in mission-critical applications.

8. CONCLUSION

The Reliability-Aware Scheduling Approach (RASA) represents a significant advancement in the domain of fault-tolerant scheduling for Field-Programmable Gate Array (FPGA)-based embedded systems operating in extreme environments. By integrating a hybrid offline-online scheduling framework with partial reconfiguration and preemptive scheduling, RASA achieves a commendable Task Success Ratio (TSR) of 72% at 70% system utilization under high fault rates ($\lambda = 0.01$ faults/unit time), surpassing traditional methods such as Triple Modular Redundancy (TMR), Duplication with Compare (DWC), and Earliest-Deadline First (EDF) scheduling by 20–40% in efficiency. Its innovative use of partial reconfiguration reduces latency by 72% compared to full reconfiguration, while novel metrics like the Anomaly Factor and Reliability Cost provide precise tools for assessing fault impact and system performance. These attributes position RASA as a robust solution for ensuring real-time reliability in applications such as space exploration, nuclear facility monitoring, and high-altitude avionics, where Single Event Upsets (SEUs) pose substantial challenges.

Despite its strengths, RASA's reliance on simulation-based validation and its optimization for the Xilinx Virtex-5 FPGA highlight areas requiring further refinement. The absence of real-world testing in extreme conditions, coupled with hardware-specific design constraints, limits its immediate applicability to diverse platforms and environments. Moreover, the Internal Configuration Access Port (ICAP) bottleneck and the need for empirical tuning of parameters underscore the necessity for enhanced scalability and generalizability. Nevertheless, RASA's methodological contributions lay a strong foundation for future research, particularly in the development of cross-platform compatible scheduling frameworks and the integration of advanced fault models and artificial intelligence techniques.

In conclusion, RASA offers a compelling framework for addressing the reliability challenges faced by FPGA-based systems in extreme environments, balancing efficiency, adaptability, and low overhead. Its superior performance metrics and innovative design principles mark it as a pivotal contribution to the field of real-time embedded systems. Ongoing and future efforts focusing on real-world validation, platform portability, and dynamic optimization

will be instrumental in realizing RASA's full potential, ensuring its adoption in mission-critical applications where reliability is paramount. This review underscores RASA's transformative impact and its role as a catalyst for further advancements in resilient computing architectures.

REFERENCES

[1] S. Saha, X. Zhai, S. Ehsan, S. Majeed, and K. McDonald-Maier, "RASA: Reliability-Aware Scheduling Approach for FPGA-Based Resilient Embedded Systems in Extreme Environments," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 52, no. 6, pp. 3885–3899, Jun. 2022.

[2] J. Jin, S. Lee, B. Jeon, T. T. Nguyen, and J. W. Jeon, "Real-time multiple object centroid tracking for gesture recognition based on FPGA," in *Proc. 7th Int. Conf. Ubiquitous Inf. Manag. Commun.*, 2013, p. 80.

[3] S. Bhasin, S. Guilley, A. Heuser, and J.-L. Danger, "From cryptography to hardware: Analyzing and protecting embedded Xilinx FPGA," in *Proc. IEEE Int. Conf. Embedded Syst.*, 2015, pp. 1–8.

[4] I. Ghorbel, M. B. Jemaa, and M. S. Obaidat, "FPGA-based implementation of robotic motion technique," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–6.

[5] A. Sari, M. Psarakis, and D. Gizopoulos, "Scrub: A low-cost fault-tolerant technique for FPGAs," in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach.*, 2016, pp. 1–8.

[6] M. Psarakis and A. Sari, "Fault-tolerant scheduling for real-time embedded systems," in *Proc. IEEE Int. Conf. Embedded Comput. Syst.*, 2017, pp. 1–10.

[7] M. Haque, J. Raik, and R. Ubar, "A fault burst model for FPGA-based systems," in *Proc. IEEE Int. Conf. Design Test Integr. Syst. Nanoscale Technol.*, 2019, pp. 1–6.

[8] A. Jacobs, G. Cieslewski, and A. D. George, "Fault-tolerant scheduling for TMR and DWC," in *Proc. IEEE Int. Conf. High Perform. Comput.*, 2015, pp. 1–8.

[9] A. Sari, M. Psarakis, and D. Gizopoulos, "Combining scrubbing with checkpoint/rollback for fault-tolerant FPGA systems," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2016, pp. 1–8.

[10] M. Psarakis and A. Sari, "EDF-based fault-tolerant scheduling for FPGA systems," in *Proc. IEEE Int. Conf. Embedded Syst.*, 2017, pp. 1–10.

[11] S. Saha, K. McDonald-Maier, and X. Zhai, "Time-redundancy based fault tolerance for FPGA systems," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2018, pp. 1–6.

[12] M. Happe, A. Agne, and C. Plessl, "Context switching for FPGA-based real-time systems," in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach.*, 2016, pp. 1–8.

[13] S. Saha, X. Zhai, and K. McDonald-Maier, "Scheduling complexity for real-time FPGA systems," in *Proc. IEEE Int. Conf. Embedded Syst.*, 2019, pp. 1–6.

[14] S. Saha, K. McDonald-Maier, and X. Zhai, "Non-preemptive scheduling for FPGA-based systems," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2017, pp. 1–6.

[15] S. Saha, X. Zhai, and K. McDonald-Maier, "Reliable scheduling for fully reconfigurable FPGA systems," in *Proc. IEEE Int. Conf. Embedded Syst.*, 2019, pp. 1–8.

[16] M. Haque, J. Raik, and R. Ubar, "Burst fault model for FPGA-based systems," in *Proc. IEEE Int. Conf. Design Test Integr. Syst. Nanoscale Technol.*, 2019, pp. 1–6