Real-Time Application Processing for FPGA-Based Resilient Embedded Systems in Harsh Environments

Sangeet Saha[1], Shoaib Ehsan[1], Adrian Stoica[2], Rustam Stolkin[3], Klaus McDonald-Maier[1]
[1]*Embedded and Intelligent Systems Lab, University of Essex*
[2]*Jet Propulsion Laboratory, California Institute of Technology*
[3]*Extreme Robotics Lab (ERL), University of Birmingham*
{ [1]sangeet.saha, [1]sehsan, [1]kdm }@essex.ac.uk, [2] adrian.stoica@jpl.nasa.gov, [3] r.stolkin@cs.bham.ac.uk

***Abstract*—Real-time embedded systems nowadays get employed in harsh environments such as space, nuclear sites to carry out critical operations. Along with the traditional software based (CPU) execution, FPGAs are now also emerging as a bright prospect to accomplish such routines. However, these platforms are often get plagued by faults generated due to the high radiations in such environments. As a result, the real-time applications running on the platform could also get jeopardized. Thus, efficient execution of a set of hard real-time applications on reconfigurable systems with anomaly detection and recovery mechanism is inevitable. This work aims at tackling such problem with a "healing" approach for extreme environments. Initially, the applications are intelligently partitioned for hardware and software execution, then attempts have been made to schedule hardware applications with intermittent preemption point. Upon detecting any abnormality on such distinct points, our approach orchestrates a healing mechanism to remediate the scenario without hampering the pre-determined schedule. Experimental validation of our proposed method reveals its effectiveness.**

***Index Terms*—FPGA, Real-time Scheduling, Harsh Environments, SEUs, Healing mechanisms, Resilient Systems**

I. Introduction

In the present application specific computing era, embedded system plays a crucial role by executing a set of dedicated functions as per user requirements. Many such embedded systems also exhibit *"real-time"* characteristics where, the correctness of the system not only depends upon the logical results being produced but also on the "time" at which the output is generated. In such systems, output delivered after the stipulated time-span (commonly known as "deadline") may cause a catastrophe and thus, requires to be handled through sophisticated applications management mechanisms.

Such embedded systems may also be part of complex robotic systems. A complex robotic system deals with the management of persistent periodic applications where such application repeatedly arrives at a certain interval [1]. These persistent applications may arise from events such as constant environmental data capture from thermal sensors, navigation and collision avoidance through camera and even the speed control through motion sensors. Moreover, such applications also impose real-time constraints and thus, a well defined scheduling methodology is indeed required. Execution of such applications can be carried out through a software based execution (CPU) or through a complete hardware based execution (FPGA). To a large extent, the answer for choosing a correct execution platform would depend upon the type of processing that needs to be performed. However, the hardware based execution offers an ideal resource for implementing applications inherently parallel in nature

Thus, FPGAs are becoming an inescapable paradigm of real-time embedded systems. Therefore, applications ranging from automotive systems to nuclear reactors etc. [2], have initiated to employ FPGA. Reconfigurable architecture has also secured its position in synthetic vision, object tracking [3], cryptography [4] etc.

In recent past, research community has focused on developing seamless periodic application scheduling strategy for reconfigurable systems [5], [6]. Most of the proposed algorithms dealt with the problem of efficient placement of tasks on the FPGA to minimize the fragmentation and the intertask communication delay [6], [7]. Due to the involved engineering challenges to implement hardware context switching in FPGAs, most of the existing scheduling strategies [6], [8] confined themselves into non- preemptive scheduling strategies. However, in order to achieve higher resource utilization, preemptive scheduling schemes are essential. Authors in [5], proposed a preemptive scheduling strategy based on "Deadlined Partitioned Fair (DP-Fair)" [9] technique and achieved high resource utilization.

Concern looms over the performance of such system in harsh conditions. In some extreme environments, these systems may have to perform in the presence of high radiations [10]. It is worth mentioning that FPGAs are susceptible to faults in such conditions [11]. As a result, the applications tend to malfunction and thus, it can jeopardize the scheduling and hinder real-time applications to complete their execution within the deadline. Such a scenario may be catastrophic .

Merits are immense when healing approach is incorporated within the system design to make it resilient. Bitstream scrubbing or Triple Modular Redundancy (TMR) [12] are classical approaches to deal with such faults. However, designing strategy of a resilient system with real-time applications on reconfigurable platforms is merely a handful. This work deals with a newly proposed periodic real-time scheduling strategy

This work is supported by the UK Engineering and Physical Sciences Research Council through grants EP/R02572X/1 and EP/P017487/1.

for fully reconfigurable systems, entitled "DPSFR" [5]. The strategy is briefly described and then we illustrate how such an effective real time scheduling strategy can be jeopardized at runtime by radiation induced fault in extreme environments. It is quite obvious that effects of such disruption can be fatal. Hence, incorporation of healing approach is essential. At first we propose our offline software-hardware partitioning strategy. Once, we select the applications for hardware execution, our scheduling approach generates the schedule for those applications. At runtime, our strategy is capable of detecting and mitigating any kind of faulty activity without hampering the schedule. Simulation based experiments on benchmark task sets reveal the efficacy of our approach.

The main contributions of the present work can be summarized as follows:

- Analyzing the extreme environment scenario where predetermined scheduling strategy could be jeopardized at runtime due to the effect of radiation at the platform.
- Proposing an on-board healing approach which possesses the ability to take remedial action in order to bring the system back to normalcy.

The rest of the paper is organized as follows. Section II, gives a brief background of the DPSFR strategy. The offline partition and scheduling strategy is discussed in Section III with an illustrative example. Section IV, illustrates performance of the scheduling approach under extreme conditions. The proposed healing mechanism is presented in Section V. Section VI, discusses on experimental outcomes followed by a discussion on the same. Finally, the paper concludes with Section VII.

II. Background

A. Deadline Partitioning Schedular for Fully Reconfigurable Systems (DPSFR)

DPSFR schedules existing real-time periodic applications by employing deadline partitioning approach on a Fully reconfigurable FPGA. At first, time slice is represented as ts_r (r^{th} time slice) to denote the interval between the $(r-1)^{th}$ and r^{th} application's deadlines. Let us assume that at the beginning of a particular time slice ts_r, there are q periodic ready applications $T_1, T_2, \ldots T_q$ to be scheduled on m partitions (predefined) (V_1, V_2, ..., V_m) on an FPGA. Application T_i has the execution requirements Fp_i time units and period / deadline, d_i time units. The *"weight" (wt_i^h)* of each application can be defined as : $\frac{Fp_i}{d_i}$. The time-slice length can be calculated by finding the difference between two consecutive deadlines (say, d_r, d_{r-1}) among applications. Hence, the length of a r^{th} time-slice will be:

$$ts_r = d_r - d_{r-1} \tag{1}$$

Deadline partitioning approach maintains proportional fairness among the rates of execution of the applications in time slices demarcated by allocating a *"work-share"* denoted as $WShr_i (= Fp_i/d_i \times ts_r)$ to be completed by each application within the time slice ts_r. However, when the processing elements are m equal sized partitions[1] on a fully reconfigurable FPGA, context switches have to be handled more carefully as they may only be realized through full reconfigurations of the FPGA. Thus, context switches on all the m partitions must occur synchronously and each such system wide context demands a full reconfiguration. For every context switch, each of the m partitions will incur a full reconfiguration overhead. Thus, the total scheduling overhead for a context switch becomes, $OF_{rg} \times m$, where OF_{rg} denotes the time required for the full reconfiguration. This overhead is usually compensated by the available system slack (if any) at time slice ts_r, and such a slack is given by:

$$ts_r \times m - sum_{WShr} \tag{2}$$

sum_{WShr} is sum of work-shares of all arrived applications.

Hence, the maximum number of possible context switches CT within the interval ts_r can be defined as:

$$CT = \lfloor \frac{ts_r \times m - sum_{WShr}}{OF_{rg} \times m} \rfloor \tag{3}$$

Initially, one reconfiguration is required to allocate applications into the m partitions at the beginning of ts_r. The remaining will be used to allow "CT-1" context switches at equal interval within ts_r. Interval between any two consecutive context switching is termed as a "time-frame". An application must be executed at a partition at least for the duration of a single time-frame of length TF. So, the number of time-frames required by an application T_i to complete its work-share is: $\lceil \frac{WShr_i}{TF} \rceil$.

At the beginning of each time-frame within ts_r, m applications having *highest remaining shares* are selected and get assigned on the m available partitions for TF duration. Thus, all applications will be able to complete their work-shares within the time slice provided equation 4 is satisfied.

$$\sum_{i=1}^{q} \lceil \frac{WShr_i}{TF} \rceil \leq CT \times m \tag{4}$$

III. Partition and Scheduling Strategy

Being an embedded system, it is possible to know the initial distribution of the applications which will periodically arrive for execution at runtime. Thus, the offline Partitioning and Scheduling strategy will attempt to partition and then schedule the applications from entire application set $\mathcal{N}$ (consists of n applications) where, $\mathcal{N} = \{T_1, T_2, ..., T_n\}$. Among these applications, our partition strategy will chose which application will be suitable for the execution on the fully reconfigurable FPGAs or CPUs, respectively.

A. Hardware-Software Partitioning

Let us assume that the application T_i exhibits execution time Cp_i and Fp_i while it executes on the CPU and FPGA, respectively. T_i also has a common deadline d_i. Hence, the resource demand of an application from the perspective of

[1]It is assumed that any tasks can be accommodated in any partition.

software execution can be denoted as: $< Cp_i, d_i >$. When T_i is executed on the FPGA, it has a spatial resource demand (i.e. width (w_i) and height (h_i); consumed in the FPGA floor) and temporal resource demand (Fp_i). Thus, resource demand of an application from the perspective of hardware (FPGA) execution can be denoted as: $< Fp_i, d_i, w_i, h_i >$.

However at runtime, T_i will execute either on the CPU or on the FPGA. Thus, we define a variable χ_i where $\chi_i = 1$ while T_i executes on the CPU and $\chi_i = 0$ while T_i executes on the FPGA. Hence, execution time (E_i) for T_i becomes

$$E_i = \chi_i \times Cp_i + (1 - \chi_i) \times Fp_i \tag{5}$$

The total execution time (say, E) for all applications in the application set i.e ($\forall\ T_i \in \mathcal{N}$) becomes:

$$E = \sum_{i=1}^{n} E_i = \sum_{i}^{n} \{\chi_i \times Cp_i + (1 - \chi_i) \times Fp_i\} \tag{6}$$

Eventually, the equation 6 can be transformed into:

$$E = \sum_{i=1}^{n} \{\chi_i \times (Cp_i - Fp_i) + Fp_i\} \tag{7}$$

In equation 7, the term $(Cp_i - Fp_i)$ can be termed as *"Decision Parameter (DP)"* . This DP shall be calculated for each application T_i as:

$$DP_i = (Cp_i - Fp_i) \tag{8}$$

This DP will be used to evaluate whether an application is suitable for the software (CPU) or hardware (FPGA) execution. The next question would arise that how many number of applications we will select for hardware execution. In order to address this issue, we have defined a new parameter α and termed it as *"**H**ardware **A**pplication **A**ccomodate **C**apacity" (HAAC)*. This α will deliver an upper limit on the number of applications which we can chose for hardware execution. To estimate α, we have initially assumed a number of partitions in FPGA and denoted as $\mathcal{M}$. However, $\mathcal{M}$ is not a big integer[2] and will not represent the actual number of partitions of our system. In the next section, we have defined the actual number of partitions. We shall select α numbers of application in a way so that the following condition get satisfied.

$$\sum_{i=1}^{\alpha} \frac{Fp_i}{d_i} = \sum_{i=1}^{\alpha} wt_i^h \leq \mathcal{M} \tag{9}$$

Applications are real-time and periodic thus, the above condition is a quick check to keep the load of the applications under the total system capacity. The partitioning strategy is described in algorithm 1.

Algorithm 1: S/W, H/W Application Partitioning Strategy

```
Input:
1. $\mathcal{N}$: set of $n$ applications
2. $Cp_i$: CPU execution time of the application $T_i$
3.$Fp_i$: FPGA execution time of the application $T_i$
4. $\mathcal{M}$ : Initial number of partition
Output:
1. $\tau_s$: Task set selected for s/w execution
2. $\tau_h$: Task set selected for h/w execution
Initialize: $\tau_s$=$\tau_h$=NULL; ▷ Both the task set is initialized as NULL
for each application $T_i \in \mathcal{N}$ do
    Calculate $DP_i$ using equation 8
    if $DP_i < 0$ then
        $\tau_s = \tau_s \cup T_i$
    else
        Sort the applications in decreasing order based on $DP_i$;
        Select the first $\alpha$ numbers of application such that condition 9 get satisified;
        Insert those applicatios in the set $\tau_h$;
```

B. Hardware Application Scheduling

Having these set of persistent periodic applications (τ_h) to be scheduled on hardware, our scheduling approach will take the aid of the DPSFR to generate the time-slice wise offline schedule for all applications till the hyper-period ($\mathcal{H}$) [3]. In order to do so, it will be necessary to define the number of partitions inside the FPGA. However in this paper, we have not adopted a pre-defined static partition (like DPSFR). The number of partitions inside the FPGA shall be determined based on spatial demand of the applications which have been selected for hardware execution and we have termed it as *"**S**patial **D**egree of **P**arallelism (SDP)"* and denoted as β. Each partition will be created in such a way so that any task can get accommodated inside a partition. In order to define the SDP (eventually means the number of partitions inside the FPGA) (β), the following calculation has to be conducted:

$$\beta = \lfloor \frac{W}{w_{min}} \rfloor \times \lfloor \frac{H}{h_{min}} \rfloor \tag{10}$$

In the above equation, $w_{min} = min(w_j)\ \forall T_j \in \tau_h, h_{min} = min(h_j)\ \forall T_j \in \tau_h$. Here, τ_h denotes the set of applications selected for hardware execution. Our scheduling strategy has been shown in algorithm 2. Now we will depict our proposed strategy through an example.

Example 1: Let us assume eight real-time periodic applications $\{T_1, T_2,..., T_8\}$ have to be executed in our system. Now each of these applications has its software execution requirement (Cp_i) and corresponding hardware execution requirement (Fp_i). Both the versions of application share a common deadline (d_i). We assumed that the initial number of partitions ($\mathcal{M}$) is 5. The software version of all applications are as follows: {28/60, 28/60, 28/90, 38/60, 48/60, 48/90, 52/60, 74/90} and similarly the hardware versions are as follows: {34/60, 15/60, 36/90, 45/60, 40/60, 36/90, 45/60, 66/90}. Based upon the "*Decision Parameters (DP)"*, the application which will suitable for hardware execution are: $\{T_2, T_5, T_6, T_7, T_8\}$. If we take sum of the weights of this application then it can be found that the condition stated in equation 9 also gets satisfied. Hence, these set of applications get selected for hardware execution according to our algorithm 1.

[2]We have empirically chosen $\mathcal{M}$ to lie between 4 to 6, in experiments.
[3]LCM of the deadlines of all applications

Algorithm 2: Hardware Applications Scheduling strategy

```
Input:
1.$\tau_h$: Task set selected for h/w execution
2. $\beta$: The SDP i.e no. of partitions
3. $Fp_i$: The hardware execution time
4. $d_i$: Deadline of an application
5. $OF_{rg}$: Full Reconfiguartion Overhead
Output:
1. Generate schedule for the application
begin
    Calculate possible times-slices considering all
      applications using equation 1;
    for each time-slice $ts_r \in \mathcal{H}$ do
        Calculate the number of context switches ($CT$)
          using equation 3 where $m = \beta$ ;  ▷ The number of
          $CT$ is equal to the number of time-frame
        for Each time-frame within the time-slice do
            Check the Scheduling Criteria in equation 4
              using $m = \beta$;
            if Equation 4 gets satisfied then
                Generate schedule by assigning applications
                  into partitions for the time-frame;
            else
                Reduce the number of applications till
                  equation 4 get satisfied;
                Migrate those non-selected applications to
                  anothe FPGA resource;
```

Let us assume a small hypothetical FPGA of size 52×72 [4] and minimum spatial resource demand among the selected hardware applications is 22×36. Hence according to equation 10, the SDP (β) becomes 4 and four will be the actual number of partitions in the FPGA. The full reconfiguration overhead (OF_{rg}) is 5 time units. The minimum deadline among all the applications is 60. Hence, the length of first time-slice ts_1 will become 60 and second time slice ts_2 becomes 90-60=30. However, we have shown the scheduling strategy for the first time slice throughout all the examples. According to our hardware application scheduling strategy as stated in algorithm 2, we will generate the schedule and it is pictorially shown in figure 1.

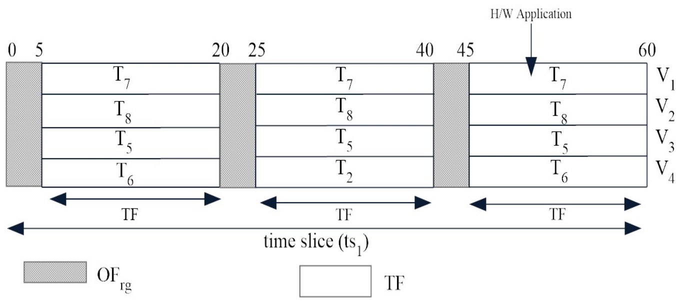

Fig. 1: H/W Application Scheduling

[4]Experiments have been conducted with practical FPGA size and hardware circuit (application) sizes, see Section VI

IV. Performance of Hardware Scheduling Under Extreme Environment

Based on the schedule generated offline, we attempt to process our real-time applications in our embedded system in extreme environments. Recent literatures (such as [12], [13]) shows that such radiation causes Single Event Upset (SEU) faults and even Multiple Event Upset (MEU) in the FPGA configuration memory. This SEU can cause an alteration in the register contents as well as it may affect the CLBs (Configurable Logic Blocks) by changing the value of LUT (Look up Tables). As a result it will change the *"context"* of any running hardware application. Subsequently, it will jeopardize the scheduling strategy which is primarily preemptive in nature and depends upon *"Context Switching"*. The context switch can be viewed as an amalgamation of two distinct components, saving the state of a partly accomplished application and restoring saved states to resume the execution. State-holding element like Flip-flops and LUT-RAMs of CLB are responsible for storing the "contexts" of a hardware application and in an extreme environment such state-holding elements get affected.

In our scheduling strategy, the radiation can upset the context and with such affected contexts (CLB's contents) our hardware scheduling scheme will malfunction. This is because, the execution of the applications may halt in between and such situation will also affect the resumption of hardware application after the preemption. Context switch in a fully reconfigurable FPGA can only be realized through a full reconfigurations. At the end of a time-frame, through the *"bit-stream readback"* procedure the contexts of the running applications can be retrieved. These contexts will reveal the present state of execution of the applications. By retrieving the execution status, we will attempt to derive a metric which will quantitatively estimate the effect on our scheduling scheme.

A. Measurement of Performance Degradation

At runtime, the retrieved context from the extracted bit-stream will reveal whether each application is able to complete their previous allotted set of program-steps (instruction) in hazardous situation. We have considered that our scheduling strategy will employ application T_η to execute P_j number of program-steps at a j^{th} time-frame (refer, section II-A). However, it may so happen that due to the upset of the internal CLB, the T_η will only be able to complete P'_j number of steps. Having this information, our strategy will attempt to derive a "Performance Degradation" parameter ("Degradation Metric (DM)") for an application as:

$$DM_\eta = P_j - P'_j \quad \forall T_\eta \in \tau_h \tag{11}$$

At the end of each time-frame, this DM will be calculated for each application to measure its performance. However, it may happen that all the running applications may not be hampered by the upset of CLB's contents. Thus, it can be inferred that applications which will exhibit "Degradation Metric" $DM_\eta > 0$ are affected and require remedial action.

V. Healing Approach for Degraded Applications

To tackle the degraded application processing, the remedial action will be carried out through the following mechanisms:

- *Modified Bitsream Loading:* Once an application exhibits degraded performance, the next course of action will be to load the updated (new) bistream file (.bit file) of the application for the next consequent time-frames. This updated bistream is created with different architecture (containing different number of CLB and LUTs) and has to be loaded in another partition of FPGA. It will be less probable that the particular application will get again affected in the other partition with new bitstream. It may also happen that any other application may not get affected (with different internal CLB, memory architecture) in the previously "affected" partition.
- *Check and Speed up*: This mechanism attempts to compensate the degraded performance. Execution of any affected application (T_η) is degraded by DM_η amount which has to be completed along with pre-allocated program steps (p_j) by the end of upcoming time-frame [5]. In order to do so, we will speed-up the execution of the application by increasing the clock frequency. The frequency will be increased by a factor defined as Speed-up Factor (SF) :

$$SF = \frac{TF_j}{DM_\eta + P_j} \tag{12}$$

NOTE: The clock frequency will be increased by enhancing the clock period from Dynamic Clock Management (DCM) system for that particular partition only where the affected application is supposed to execute. [6]

We will now demonstrate the whole healing approach through the same example application set as shown in example 1.

Example 2: Let us assume that at the end of first time-frame, through bitstream read-back it is found that T_8 was able to complete only 10 program steps instead of alloted 15 program steps as shown in figure 2. As we have assumed that if the length of time-frame is 15 then each application has to execute 15 program steps. Thus, the "Degradation Metric" for T_8 becomes: DM_8= 5 (refer, equation 11).

However, it is found that other applications are able to complete their executions. So, in order to bring the normal operation of the system such that T_8 completes the allocated *work share*, we have loaded an updated bitsream to new partition V_1 and also we have to speed up the execution T_8 such that it can able to finish its pre-alloted 15 program steps at second time-frame along with the 5 steps which are pending from first time-frame. The procedure is shown in figure 3.

VI. Experiments and Results

We have measured the performance of our strategy through software simulation. We have chosen all the parameters from practical Virtex 4 FPGA and ITC99 benchmarks [14] for generating the application sets.

[5]Inside a time-slice, length of all time-frames are same and so does the required program-steps

[6]In case, "Degradation Metric" (DM) found in last time-frame of a time-slice ts_r, the healing approach shall be taken at the first time-frame of next time slice ts_{r+1}.

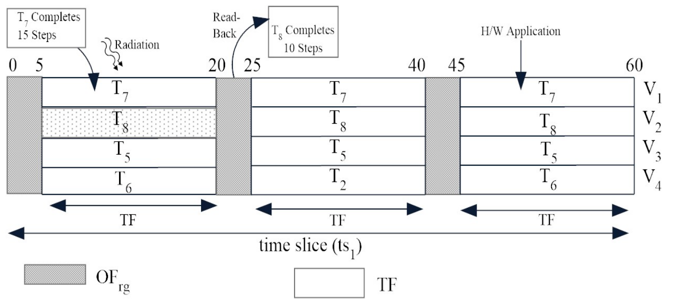

Fig. 2: H/W Application Scheduling affected by Extreme Environment

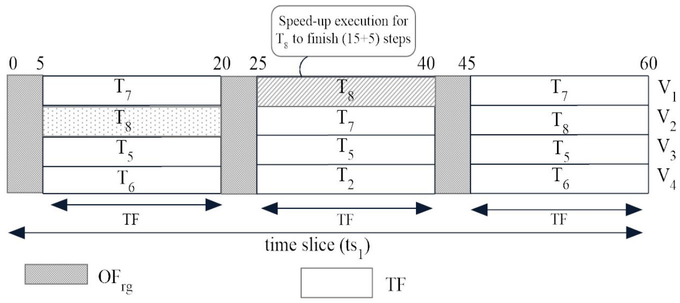

Fig. 3: H/W Application Scheduling with healing mechanism

A. Experimental Parameters:

We have chosen Virtex-4 (XC4VFX60) FPGA having a floor area 52 × 128 [15]. In [16], authors have shown that hardware application in virtex-4 FPGAs could have average area requirements of 32×32. Thus we have chosen number of partition (SDP, β=4). The data sets contain randomly generated synthetic periodic applications whose weights (execution time / deadline) have been taken from normal distribution with standard deviation σ=0.1 and mean μ =0.1. Similarly, deadlines (d_i) have also been generated from a normal distribution. Given System Pressure (SP), (obtained as the sum weights of the generated applications) and the SDP (β), the system utilization Sys_{uti} is defined as:

$$Sys_{uti} = \frac{SP}{\beta} \times 100 \tag{13}$$

In ITC99 Benchmark, an application usually contain 100 to 1000 instructions. Without loss of generality, we have also calculated the number of program steps from a normal distribution with mean: 500 and standard. deviation 200.

B. Online Overhead Analysis

Our proposed strategy will first observe the execution status of each application then it will measure the degraded performance and finally, will attempt to heal the degradation. In order to analyze the overall associated overhead, we will begin with measuring the reconfiguration overhead (RO). RO is characterized by three main factors i.e. i) Bitstream Size (*Bits*)

ii) Configuration clock frequency of the controller (C_{clk}) iii) Data Bus Width of the controller (D_{BW}). Thus:

$$RO = \frac{Bits}{C_{clk} \times D_{BW}} \tag{14}$$

In case of Xilinx Virtex-4 (XC4VFX60); $Bits$: 2.66 MB [17], C_{clk}: 100MHz and D_{BW} : 32 bit [15]. Putting these values in equation 14, RO is obtained as $\approx$ 7ms. Due to recent technical advancement [18], only $1/10^{th}$ time of RO is associated to extract any context of application. It is now possible to read that portion of bitstream which contains the register's context instead of reading the whole bitstream. Hence, the extraction time becomes ($\frac{1}{10} \times 7\ ms$) = $0.7\ ms$. Thus, we can judiciously consider the read-back time as 1 ms. Degradation measurement and healing operation is performed by the system's Embedded Processing Resource (EPR). Thus, measurement of "Degradation Metric (DM)" and finding the new bitstream can be performed in $\approx 2\ ms$ with standard EPR. Hence, the overall associated overhead can be quatified as follows : $1\ ms + 2\ ms + 7\ ms = 10\ ms$ *(Read-back + DM calculation and Bitstream locate + New bitstream download).*

C. Result and Analysis

We have defined *Application Acceptance Rate (AAR)* as our performance metric. *AAR* is defined as the percentage of the total number of applications accepted (ν) by the system over the entire schedule length out of total number of arrived applications (n). i.e.,

$$ARR = (\nu/n) \times 100 \tag{15}$$

We have taken 20 different instances of each dataset type and finally, generate the average over these 20 runs. The entire schedule length is 10000 time slots. The fault occurrence rate (λ) is considered as 0.02 which infers that there are two upsets occur within 100 time slots for any arbitrary partition.

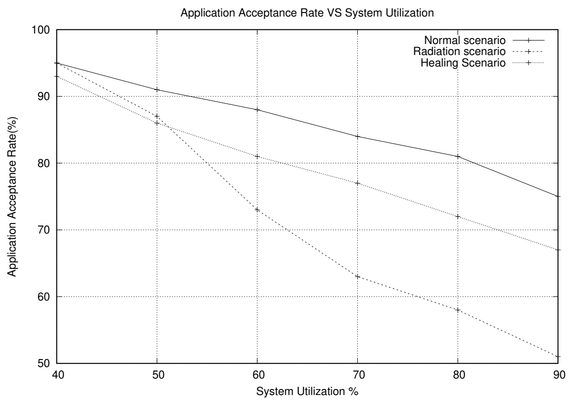

Fig. 4: AAR Vs. Sys_{uti}; β = 4

Figure 4 depicts the variation of ARR by our scheduling strategy (having $OF_{rg} = 10ms$) under varying system utilization. It can be observed that in normal scenario, our system can accept as high as 80% applications under high system pressure ($Sys_{uti} = 90\%$). ARR decreases further when the faults are injected. This is due to the fact that the presence of event upset prevents the applications from completing the required set of instructions. However, the healing mechanism can bring the AAR close to normal scenario and it is able to achieve 70% resource utilization with more than 75% AAR.

VII. Conclusion

Generating schedules for reconfigurable platforms is essential to ensure completion of a set of applications within a certain time interval. However, such schedules can be jeopardized by faults induced by radiation in extreme environments. We explore and illustrate such a scenario in the current context. In addition to this, we also present a healing methodology which attempts to mitigate vulnerable scenario for scheduling periodic hard real-time applications on fully reconfigurable platform. The proposed strategy detects the anomaly, deciphers the cause and heals the scenario by speeding up application processing and re-loading a fresh bitstream. We designed, analyzed and validated the approach thorough simulation framework and the outcomes are good.

References

[1] L. T. Yang, E. Syukur, and S. W. Loke, *Handbook on Mobile and Ubiquitous Computing: Status and Perspective*. CRC Press, 2012.

[2] T. Hayashi, A. Kojima, T. Miyazaki, and N. Oda, "Application of fpga to nuclear power plant i&c systems," in *Prog. of Nuclear Safety for Symbiosis and Sustainability*. Springer, 2014, pp. 41–47.

[3] 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. of the 7th Intl. Conf. on Ubiquitous Info. Mgmt. and Comm.*

[4] S. Bhasin, S. Guilley, A. Heuser, and J.-L. Danger, "From cryptography to hardware: analyzing and protecting embedded xilinx bram for cryptographic applications," *Journal of Crypto. Eng.13*, vol. 3, no. 4.

[5] S. Saha, A. Sarkar, and A. Chakrabarti, "Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms," *Embedded Syst. Letters, IEEE*, vol. 7, no. 1, pp. 23–26, 2015.

[6] Q.-H. Khuat and D. Chillet, "Communication cost reduction for hardware tasks placed on homogeneous reconfigurable resource," in *(DASIP), 2013*, Oct 2013, pp. 265–270.

[7] S. Baruah, M. Bertogna, and G. Buttazzo, "Real-time scheduling upon heterogeneous multiprocessors," in *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015, pp. 205–211.

[8] A. Eiche, D. Chillet, S. Pillement, and O. Sentieys, "Task placement for dynamic and partial reconfigurable architecture," in *DASIP, 2010.*

[9] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS 2010*, July 2010, pp. 3–13.

[10] M. Finckenor and K. de Groh, *Space Environmental Effects*, ser. A researcher's guide to. NASA ISS Program Science Office, 2015.

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

[12] J. Ranta, "The current state of fpga technology in the nuclear domain," VTT Technical Research Centre of Finland, Tech. Rep., 2012.

[13] L. Sterpone, S. Azimi, B. Du, D. M. Codinachs, and R. Grimoldi, "Effective mitigation of radiation-induced single event transient on flash-based fpgas," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 203–208.

[14] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers*, vol. 17, no. 3, pp. 44–53, 2000.

[15] I. Xilinx, "Virtex-4 family overview," *Tech. Doc. DS112*, 2010.

[16] Y. Lu, "Realistic online resource management for partially reconfigurable systems," Ph.D. dissertation, 2011.

[17] Xilinx, "Virtex-4 fpga configuration user guide, xilinx," *June*, 2009.

[18] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, and H. Takada, "Comparison of preemption schemes for partially reconfigurable fpgas," *IEEE ESL*, vol. 4, no. 2, pp. 45–48, 2012.