Enabling FPGAs for Future Deep Space Exploration Missions: Improving Fault-Tolerance and Computation Density with R3TOS

Xabier Iturbe1,2, Khaled Benkrid1, Tughrul Arslan1, Chuan Hong1 and Imanol Martinez2

1The University of Edinburgh, Edinburgh EH9 3JL, Scotland, UK
{x.iturbe, k.benkrid, t.arslan, c.hong-2}@ed.ac.uk
2IKERLAN-IK4 Research Alliance, Mondragón 20500, Basque Country (Spain)
{xiturbe, imartinez}@ikerlan.es

Abstract

Future deep space exploration missions will require small, lightweight, low-power, fault-tolerant, autonomous and durable onboard electronic systems. In this paper we present R3TOS, a novel approach for online scheduling and allocating different pieces of circuitry onto partially reconfigurable FPGAs to match the specific computation needs at all times. R3TOS targets two main objectives: (i) obtain the best performance per unit of device’s area and per unit of consumed energy and, (ii) minimize the impact of device’s degradation on performance.

1 Introduction

Ordinary electronic circuits that work fine on Earth are useless in space. The high-energy electromagnetic waves and charged particles, collectively known as “ionizing radiation”, pass through them, damaging whatever gets in their way. The degrading effects that radiation provokes can be divided into two categories: Single Event Effects (SEE) and Total Ionizing Dose (TID) [1]. SEEs refer to non-destructive electron-hole pairs caused by an energetic particle that strikes the chip and lead to changes in the state of one or various transistors. TID refers to the permanent damage to a device and results in oxide breakdown and leakage current. Moreover, as process technology shrinks and electronic circuits become more complex, radiation hazards are gaining more prominence in other fields beyond space. For instance, in the last decade radiation-provoked upsets were reported at aircraft altitudes [2] and more recently even at ground level [3].

As space missions range farther and farther from Earth, SEEs and TID become major problems for electronics survival considering the long time spacecraft’s circuitry must be exposed to them. For instance, Voyager 1 needed 30 years to complete its journey to the edge of our solar system. Hence, for future far-deep space exploration missions not only mass (weight), power and reliability but also electronics’ lifetime will be a mandatory constraint when building spacecraft’s circuitry. The electronics to be used in these missions will have to be designed assuming limited capabilities for diagnosis, no replacement possibilities and little onboard capacity for spares.

At the same time, space missions are becoming more and more data intensive, with ever greater transmission times. E.g. The camera aboard NASA’s Mars Odyssey spacecraft can generate images in the gigabyte range. With the current, relatively low-bandwidth, spacecraft-to-ground-station data links, it is not possible to get this much data down to Earth for processing by ground based supercomputers. This applies specially in cases where the results of the data processing are to be used to make mission decisions, such as ‘what to do next’. And the problem gets even worse as the spacecraft moves away to the far-deep space. In this scenario the data must be processed onboard the spacecraft, the required decisions made onboard as well, and only the final results can be sent down.

From all this we conclude that, in order to ensure the success of future deep space exploration missions, we need small, lightweight, high performance, low power, autonomous and durable onboard computers able to operate successfully in harsh deep space environments.

We envision that Field-Programmable Gate Arrays (FPGAs) could meet these requirements. In fact, they have already been used in NASA’s Mars exploration missions with success. The advantages of FPGAs in space applications are numerous and well known, including the ability to reconfigure the architecture on-the-fly in response to emerging needs at all times. These needs could come from reliability issues (e.g. circumvent the use of a damaged resource in the chip [4]), but also from performance demands (e.g. deal with a workload peak when any onboard sensor releases a
new data burst [5]). It is time to turn the potential of modern reconfigurable FPGAs into real exploitable capabilities.

Toward this goal we have recently launched the R3TOS project [6]. It is aimed at developing a fault-tolerant real-time operating system for FPGAs. R3TOS will put the versatile resources embedded in the reconfigurable chip at the service of the computation in a reliable way. The data is to be processed by specialized circuits (‘hardware tasks’) that are swapped in and out of FPGA’s reconfigurable area according to the computation necessities at each time [7]. By sharing FPGA’s resources along time, R3TOS will be able to execute very large applications in small devices, achieving the best performance from each mm² put into space and reducing the static power consumption. Central to R3TOS is real-time performance and fault-tolerance. The hardware tasks are scheduled in order to meet their computation deadlines and mapped to non-damaged resources, keeping the system fault-free at all times. Besides this, the continuous reconfiguration of the hardware tasks automatically ‘scrub’ SEEs in the configuration memory of the FPGA [8]. We target a mixed-criticality system where vital computations can use FPGA’s resources in detriment of peripheral functionality; E.g. an image processing task can be stopped, giving their resources to spacecraft steering tasks. Likewise, in this scenario system performance can be gracefully degraded as device gets damaged.

Up to date we have presented a real-time algorithm for scheduling hardware tasks, Area-Time response Balancing algorithm (ATB) [9], as well as a fault-tolerant implementation for it [10]. Addressing the area-time duality of FPGA-based computation, ATB makes the scheduling decisions according to either time or area criteria depending on the timing requirements, the area usage and the fragmentation on the device at each time. Based on the fact that several small tasks can be mapped to the adjacent resources that are released when large area tasks finish their execution, ATB prioritizes the execution of great area tasks at the expense of delaying the system response time. In order to ensure no deadline is missed when early scheduling large area tasks, a feasibility test is evaluated. However, some simulation results suggest that ATB performance might decrease when scheduling task-sets with a high amount of tasks \((N >> 10)\), especially when deadlines are tight. This is related to the interaction between the scheduler and allocator which will be the subject of future study.

In this paper, however, we will use the classic Earliest Deadline First (EDF) scheduler and explore the complementary strategy; i.e. Instead of analyzing area aspects when scheduling the hardware tasks, we provide the allocator with time analysis capability. After all, this represents what really occurs in the FPGA in a better way: While FPGA’s reconfiguration port is scheduled based on exclusive time criteria, the FPGA’s logic resources are assigned based on both time (resources are shared along time) and area criteria (several hardware tasks perform concurrent computation in space).

The rest of the paper is organized as follows. After introducing some basic concepts on FPGAs in Section 2, we present the big-picture of R3TOS in Section 3. In Section 4 and Section 5 we respectively focus on the algorithms used for scheduling and allocating the hardware tasks. Then in Section 6, an experimental evaluation of the algorithms performance is included. Finally, concluding remarks are pointed in Section 7.

## 2 Context Definition

A modern Xilinx FPGA fabric consists of several logic resources (e.g. CLBs, DPS48s, BRAMs, etc.) as shown in Fig. 1. Some authors use the term sandbox to refer to the regular array of reconfigurable CLBs occupying the central region of the chip where the tasks are to be executed.

![Figure 1: Xilinx Virtex-4 XCVLX200 device layout (A modern Xilinx FPGA fabric consists of several logic resources (e.g. CLBs, DPS48s, BRAMs, etc.) as shown in Fig. 1. Some authors use the term sandbox to refer to the regular array of reconfigurable CLBs occupying the central region of the chip where the tasks are to be executed.)](image)

The functionality assigned to the logic resources included in the FPGA and their interconnection is defined by a configuration bitstream, which is stored into device’s internal configuration memory. For Xilinx Virtex-4 FPGAs the bitstream is divided into configuration frames, which can be seen as vertical stacks of 1312 bits each that span the whole height of a fabric clock region [11]. Every configuration frame is univocally addressed within the bitstream and includes the so-called Frame_ECC codes, which permit to detect the corruption provoked by faults (both SEEs and TID) that is manifested as configuration upsets.

The use of SRAM technology in the configuration memory of Xilinx FPGAs permits to download new configuration data at runtime. This process is called Dynamic Partial runtime Reconfiguration (DPR) and makes it possible to change the functionality assigned to a specific region of a device while the rest of it is still operating. The smallest amount of configuration information that can be accessed in the configuration memory is the configuration frame. Hence, the minimum chip’s logic area modifiable when using DPR is a whole fabric clock region spanning column, which defines thus the granularity of the reconfiguration...
(e.g. 16 CLBs, 4 BRAMs or 4 DSPs). Consequently, by using DPR any given piece of circuitry allocated in a dedicated clock region can be individually replaced without interrupting the operation of the rest of the circuits in a similar way to software tasks are switched in a Von Neumann processor. Following this analogy we call such circuits hardware tasks. The fact of directly dealing with hardware components when programming the hardware tasks permits to outperform their software counterparts.

Traditionally, an FPGA device is partitioned into independent reconfigurable slots whose boundaries match with device’s clock regions. At design time each hardware task is assigned to a specific slot, being possible to switch at runtime among them by reconfiguring the slot content. However, as the boundaries of the reconfigurable slots cannot be changed at runtime, resources are wasted when fitting small hardware tasks into larger area slots, leading to low efficiency. In addition, from the fault-tolerance viewpoint, a single damaged CLB could make useless a whole reconfigurable slot. In order to achieve a higher flexibility relocatable bitstreams have been proposed [12]. The idea is to exploit the regular structure of the Xilinx FPGAs sandbox for allocating the same partial bitstream in different positions. In order to do so, the configuration frame addresses can be modified and hence, the same pieces of configurations can be used to reconfigure different physical resources within the sandbox. Relocation permits to overcome reconfigurable slots boundaries, being possible to merge and later separate again the clock regions based on the shape and size of the hardware tasks to allocate.

Finally, modern Xilinx FPGAs ease the access to the configuration memory by including an Internal Configuration Access Port (ICAP), which removes the need for external components that control the reconfiguration process. Unfortunately, there is a single ICAP in an FPGA and its access is relatively slow (400 MB/s). Consequently, only one hardware task can be switched at any time and doing so takes a non negligible amount of time [13]. This time can however be exploited to make hardware tasks’ scheduling and allocation decisions and thus, sophisticated algorithms are enabled in FPGA-based reconfigurable computers.

### 2.1 Hardware Task Model and FPGA Area Model

The hardware task is the basic unit of computation that R3TOS manages. When a hardware task $\theta_i$ is synthesized, a partial bitstream is obtained which configures the resources included within a rectangular region of $h_{x,i} \times h_{y,i}$ CLBs in the sandbox. Based on the fact that the reconfiguration granularity of Xilinx FPGAs is 16 CLBs, we set $h_{y,i}$ always to be a multiple of 16. This avoids interference with other tasks, which would be produced when, due to task switching, a frame which includes user changeable information (e.g. LUT RAM or LUT shift register) of a different running task is written through the ICAP. On its part, the sandbox is considered to be a regular arrangement of $H_x \times H_y$ CLBs with their associated routing resources. Targeting a slotless system with relocatable hardware tasks, we assume a 2-D area model where the tasks can be placed everywhere within the sandbox in such a way that they span a fixed number of clock regions. In this context, we define the Maximal Empty Rectangle (MER) as the greatest adjacent amount of available CLBs on the sandbox with a size $L_x \times L_y$.

In the time domain, a hardware task is characterized by the amount of time needed for configuring it onto the FPGA (allocation time $t_{A,i}$) and the period of time it needs to finish the computation (execution time $t_{E,i}$). Thanks to the temporal isolation among different hardware tasks execution, it is feasible to assume that every allocated task will complete its computation within $t_{E,i}$, releasing the results at $f_i$. Hence, the computation time refers to the total amount of time a hardware task uses the assigned resources $C_i = t_{A,i} + t_{E,i}$. The real-time feature of R3TOS involves the existence of a relative execution deadline for each task $D_i$, that represents the maximum acceptable delay for that task to finish its execution. The absolute deadline for task execution $d_i$ is computed by adding the task arrival time $r_i$ to the relative execution deadline. Even more important are the relative deadline for task allocation $D_i^r = D_i - t_{E,i}$, that represents the maximum acceptable delay for a task to be placed in the device in order to meet its execution deadline, and the associated absolute allocation deadline $d_i^r$. Finally, a priority $P_i$ is assigned to each task according to the criticality of the computation it performs. Higher priority tasks are always executed before lower priority ones.

![Figure 2: Hardware task definition and FPGA area model](image)
also involves saving the state of all the registers by reading-
back their values through the ICAP. Consequently, this pro-
cess triples the ICAP occupation. On the other hand, the
preemption of an allocating tasks does not significatively
increase the ICAP utilization. In any case, allocation task
preemption must be carefully used as the preempted tasks
occupy reconfigurable area in the sandbox without carry-
ning out computation. Hence, in order to meet the deadline
for critical computations R3TOS preempts the allocation of
low priority tasks. Besides this, if strictly necessary, a low
priority task running on the FPGA can be deallocated (the
computation performed until then will be lost), reassigning
its resources to a higher priority task. These mechanisms
enable hard real-time performance for critical tasks at the
expense of increasing the amount of missed deadlines for
soft real-time non-critical tasks.

In order to improve the performance, R3TOS keeps the
tasks allocated in the FPGA after they finish the execution,
being removed only when the resources they use are needed
for allocating other ready tasks. Hence, when an allocated
task is again triggered it can immediately start performing
computation, circumventing the reconfiguration phase. A
hardware task goes thus through a series of states until it
finally finishes its execution: Waiting, Ready, Allocating,
(Preempted), Executing an Allocated (See Fig. 3).

Figure 3: States of a hardware task

2.2 Xilinx FPGAs defining Computational Model

FPGAs open the door for more sophisticated and pow-
erful computation systems in which the available resources
can be dynamically utilized in order to efficiently execute
several concurrent hardware tasks in a reliable way.

However, the fact there is a single ICAP in an FPGA in-
troduces a bottleneck that restricts the intensive exploita-
tion of hardware parallelism. \( U_{ICAP} \) refers to the utilization
of the ICAP, which characterizes the real-time laxity of a task-
set \( \phi \) (Eq. 1).

\[
U_{ICAP} = \sum_{\forall \phi_i \in \phi} \frac{t_{A,i}}{D_i} < 1
\]  

(1)

Furthermore, the limited amount of resources avail-
able in the device constrains the number of hardware
tasks that can be executed in parallel. When extend-
ing along time domain, the CLBs included in the sand-
box define a 3-D cube, the so-called Computation Cube
\( CC(t) = t \cdot H_x \cdot H_y \). Analogously, every hardware task \( \theta_i \)
defines a computation volume in the area-time space equal
to \( C_i \cdot h_{x,i} \cdot h_{y,i} \) to be inserted in the CC. In this scenario,
the utilization of the computational resources of the FPGA
\( U_{COMP}(t) \) is defined as the proportion of the computation
volume occupied by the hardware tasks in the CC. This term
depends on the triggering of the hardware tasks and thus
varies along time. A time-independent expression which
characterizes the computation requirements of a task-set \( \phi \)
can be deduced assuming periodic hardware tasks with the
period equal to their deadline [9] (See Eq. 2). We note that
the CC can never be fully occupied due to the serial recon-
figration mechanism.

\[
U_{COMP} = \frac{1}{H_x \cdot H_y} \cdot \sum_{\forall \theta_i \in \phi} \frac{C_i \cdot h_{x,i} \cdot h_{y,i}}{D_i} < 1
\]  

(2)

A feasible schedule, where all the computation dead-
lines are met, can only be produced when both \( U_{ICAP} \) and
\( U_{COMP} \) remain less than the unit. In fact, a computation
deadline is missed if (i) the occupation of the ICAP delays
too much the allocation of a task, or (ii) in case the logic
resources required by the task are not available before its
deadline expires. This can occur because either there are
not sufficient free resources on the FPGA or because the re-
sources are not available in a sufficiently large contiguous
region due to fragmentation. Fragmentation is provoked not
only by occurring faults in the silicon substrate, but also
by the continuous allocation of hardware tasks with differ-
ent sizes that split the sandbox into multiple fragments over
time. The feasibility conditions 1 and 2 are mandatory for
critical hard real-time tasks while they do not apply to pe-
ipheral soft real-time tasks.

In contrast with related work in the field, in this paper
we use a realistic computation model which includes all the
technological particularities and limitations of current FP-
GAs. This ensures a successful implementation of R3TOS
when using real hardware.

3 The R3TOS Approach

Fig. 3 shows the simplified model of R3TOS, which in-
cludes two main modules: the Scheduler and the Allocator.
The real-time Scheduler coordinates the access to the
ICAP, assigning the appropriate allocation starting times to
the hardware tasks in order to meet computation deadlines.
The scheduler is also responsible for the management of
the state of the tasks as well, inserting them into the suit-
able task state queues (Ready, Executing and Allocated).
Together with the scheduler, R3TOS includes a fault-aware
Allocator, which is aimed at deciding the best placement for
every scheduled hardware task avoiding damaged resource
usage and reducing fragmentation.
That is, the real-time scheduler dictates the chronological order of execution of the triggered tasks, ensuring the temporal correctness, and the fault-aware allocator maps them to the available logic resources at each time, ensuring the logical correct computation. The cooperation between both parts ensures the best results. In few words, R3TOS can be described as a discrete system that selects at every kernel-tick the most suitable hardware task to be executed according to both time and area criteria. The fact scheduling decisions are made online permits to deal with the unpredictable degradation provoked by spontaneous faults and enables data-dependant computation.

In order to diagnose the occurring faults in the silicon substrate of the FPGA, the Frame_ECC codes included in the system configuration bitstream are periodically checked. When the rewriting of the correct value of a corrupted configuration bit does not fix the problem, we consider that the logic resource which is configured by that bit is damaged. This information is passed to the allocator for consideration when allocating the hardware tasks in the future. Furthermore, with the objective of achieving the highest-reliability level, critical tasks are executed in parallel at distinct positions within the sandbox (spatial redundancy) or at different moments (temporal redundancy). This functioning scheme makes R3TOS virtually immune to SEEs and capable of coping with TID.

R3TOS is our bid for building the first fully operational Autonomous Fault-Tolerant System (AFTS), able to configure its own resources in presence of faults to maintain its functionality without any external human intervention [14].

5 Allocating Hardware Tasks

While several research efforts can be found in the technical literature for improving the computation density when allocating the hardware tasks, as far as the authors know there are no specific solutions for fault-tolerance. To increase the computation density, the hardware tasks must be packed as compact as possible in both time and area domains. With this objective Tabero et al. proposed the 3-D Adjacency heuristic (3DA) [16], which is aimed at placing the tasks at the position with the highest contact in x and y directions with previously running tasks or with the sandbox boundaries; E.g. In Fig. 5, \( \theta_i \) shares 4 CLBs with previously running \( \theta_1 \) and \( \theta_2 \) and 6 CLBs with the sandbox boundary at position A. The third dimension refers to time and thus, 3DA prolongs in time the contact perimeter leading to vertical surfaces. The 3DA heuristic is then computed as the sum of the length of the task to be allocated in contact with each adjacent edge, multiplied by the temporal adjacency between the new task and the edge. The temporal adjacency is the minimum between the edge’s lifetime and the executing time of the task to be placed. E.g. In the aforementioned example, the adjacency of \( \theta_i \) with \( \theta_1 \) and \( \theta_2 \) (at position A) and with \( \theta_3 \) and \( \theta_4 \) (at position B) prolongs for 5 units of time and hence, \( 3DA(A) = 3DA(B) = 4 \cdot 5 + 4 \cdot 5 + 6 \cdot 5 = 70 \).

Moreover, Tabero et al. proposed to keep track of the state of the sandbox occupation by means of a Vertex List Set (VLS). The VLS included the four vertices which define the shape of a hardware task: Top-Right (TR), Top-Left (TL), Bottom-Right (BR) and Bottom-Left (BL). They assumed that placing a hardware task in a vertex of an already running task always provokes less fragmentation than placing it in any other position within the sandbox as it ensures the highest contact among tasks. Consequently, each time a task is scheduled, the allocator evaluates the vertex included in the VLS in the search of the optimum allocation for it.

However, despite 3DA reduces the fragmentation in the sandbox by promoting the compaction of the hardware
tasks, it is not aware of future fragmentation situations. This can be seen in Fig. 5: Both positions A and B give the same 3DA value, but only allocation B reduces fragmentation at $t = f_I$. To address this issue we propose a 3DA enhanced heuristic which promotes the contiguous placement of the tasks with similar $f_s$. This leads to large contiguous pieces of free area when these tasks finish the computation. The 3DA$_h$ heuristic for a hardware task $\theta_i$ at any given position is computed as the sum of the lengths of $\theta_i$ in contact with the FPGA boundaries ($P_{\theta_i-FPGA}$), multiplied by $t_{E,i}$, and the lengths of $\theta_i$ in contact with other running tasks $\theta_j$ ($P_{\theta_i-\theta_j}$), multiplied by a temporal adjacency factor among the tasks ($T_{\theta_i-\theta_j}$). This $T_{\theta_i-\theta_j}$ factor is the absolute difference between the amount of time both tasks are to run in parallel and the remaining time when only one of them is to continue executing alone.

$$3DA_e = \sum P_{\theta_i-FPGA} \cdot t_{E,i} + \sum P_{\theta_i-\theta_j} \cdot T_{\theta_i-\theta_j} \quad (3)$$

Fig. 5 illustrates the functioning of 3DA$_h$. In this example: $t_{E,i} = 5$, $P_{\theta_i-FPGA} = 6$ (for both positions A and B), $P_{\theta_i-\theta_j} = P_{\theta_i-\theta_2} = P_{\theta_i-\theta_3} = P_{\theta_i-\theta_4} = 4$, $T_{\theta_i-\theta_2} = 5 - 5 = 0$ and $T_{\theta_i-\theta_3} = T_{\theta_i-\theta_4} = 5 - 0 = 5$. When applying the definition of 3DA$_h$ shown in equation 3, we obtain $3DA_e(A)=30$ and $3DA_e(B)=70$.

![Figure 5: 3DA$_h$ heuristic computation](image)

A more important problem is that 3DA is not very efficacious in dealing with the fragmentation introduced by the permanent damage, as it has a vision of only one CLB row/column beyond the boundaries of the task being placed. To tackle this problem, we propose the Empty Area/Volume Compaction heuristics (EAC and EVC). These heuristics address the state of the whole sandbox to assign a quality score to each candidate placement.

In order to compute EAC, the sandbox is firstly analyzed in both vertical and horizontal directions. A counter is assigned to each column ($cc_i$) and row ($cr_i$), being updated as follows. In the vertical analysis, every time the CLB in the next row is available to be used, the corresponding $cc_i$ is incremented and; in case the CLB is not available, $cc_i$ is reset. A CLB is not available when it belongs to the same configuration frame of any CLB already assigned to an executing task or when it is damaged (marked in black in Fig. 6). The CLBs assigned to the tasks in allocated state are considered to be available. On its part, in the horizontal analysis, every time the CLB in the next column is available to be used, the corresponding $cr_i$ is incremented and; in case the CLB is not available, $cr_i$ is reset. When an entire row or column has been analyzed and thus all the counters have been updated based on the availability of CLBs in each direction, their value is multiplied by the number of adjacent counters which store the same or greater value. This analysis is performed in the opposite direction, that is, $cc_i$s are horizontally analyzed while $cr_i$s are vertically analyzed. For each column-row, the greatest value of the aforementioned multiplications is saved (marked as HV-C/R in Fig. 6). This value represents the area of the maximal empty rectangle which is formed in vertical-horizontal directions including one of the CLBs in the analyzed column-row. The EAC is equal to the sum of all these areas, and represent thus a measure of the “compactness” of the available resources.

Likewise, the greatest value among the aforementioned areas is equal to the MER’s area. Therefore, all the ready tasks which require more area than available in the MER ($h_{x,i} \cdot h_{y,i} \geq L_x \cdot L_y$) can be discarded by the scheduler, saving vital time for the allocator.

The benefit of EAC when coping with permanent damage is illustrated in Fig. 6. According to 3DA, $\theta_i$ would have been allocated at the bottom-left vertex of the sandbox (candidate B) with an adjacency value equal to 14 (for this position the EAC measure would be 480). On the contrary, the highest EAC score (equal to 487) is obtained when placing $\theta_i$ at candidate position A, despite the adjacency value at this position is only 7. It can be seen that 3DA leads to a reduction of the size of the MER from 48 (when placing the task at position A using EAC) to 40. This makes it more difficult to allocate greater area tasks coming in the future. On the other hand, by using EAC the free area between the damaged CLBs is exploited for allocating the tasks, hence preserving as long as possible the MER.

We also propose to keep the state of the FPGA by means of $cc_i$s and $cr_i$s, instead of using a VLS. In fact, in presence of faults it is not true that the best position to place a task is always a vertex of a previously allocated task. We call this structure the Adjacency Matrix (AM). Hence, every position $(x,y)$ in the AM for which $cc_i \geq h_{y,i}$ and $cr_i \geq h_{x,i}$ is selected as an allocation candidate to be evaluated. This permits to immediately discard the non-feasible allocations, speeding-up the execution of the algorithm. When computing the EAC for a candidate, $H_y - y$ rows and $H_x - x$ columns need to be updated in the AM and thus, on average $H_y \cdot H_x / 4$ CLBs must be processed per candidate.

However, EAC does explore only the area domain. In order to include the time domain we propose the EVC heuristic. A consequence of this is the ability of EVC to deal with future fragmentation, as the afore-presented 3DA$_h$ does. The EVC for a task $\theta_j$ is computed as the sum of the EACs at each time an executing task $\theta_j$ finishes its computation,
multiplied by the amount of time the sandbox remains in the state left when \( f_j \)'s resources are released; that is, until the next executing task finishes. This applies to all the tasks for which \( f_j \leq f_i \).

Finally, a less time consuming, and also less accurate, approach (2Point-EVC) consists in summing the EAC only when the task \( \theta_i \) is allocated and when it finishes its computation: EVC\(_{2P} = EAC(t = r_i) + EAC(t = f_i)\).

6 Experimental Results

We built a discrete-time simulation framework in C to evaluate the performance of EAC, EVC and EVC\(_{2P}\) heuristics. The framework runs under Windows XP operating system on an Intel Core Duo CPU @ 3 GHz. We have targeted a Virtex-4 XC4VLX200 device with up to 12 clock regions and a sandbox of \( H_x = 42 \times H_y = 192 \) CLBs. We randomly generated different task-sets \( \phi \) of 50 synthetic periodic hardware tasks each aiming at covering a wide range of parameters for them. For simplicity, we considered that the allocation time of the tasks was equal to their size \( t_{A,i} = h_{x,i} \cdot h_{y,i} \). Furthermore, the execution period of the tasks was equal to their computation deadline. The task-sets have been classified in six classes, depending on their parameters (See Table 1). We performed 1000 experiments for each task-set class and averaged the obtained results. We set that each experiment was finished when all the tasks in the task-set had either met or missed their computation deadline at least once. To complete the characterization, we studied the behavior of the heuristics in a partially damaged device as well. We randomly selected up to 10 CLBs within the sandbox as damaged (0.12% of the CLBs). In the experiments we only considered the hardware tasks which could still be mapped to non-damaged CLBs. We used four metrics to evaluate the performance:

1. Missed Deadlines (MD): Is the percentage of missed computation deadlines.
2. Scheduling Feasibility (SF): Refers to the percentage of feasible schedules produced.
3. Exploited Computation Volume (ECV): Refers to the use of the CC by the hardware tasks which meet their computation deadlines. This term implicitly includes the Application Completion Time (ACT) parameter: A faster execution of the application involves greater computation density, leading to higher ECV.
4. Algorithm’s Execution Time (AET): Refers to the amount of time needed for making the scheduling and allocation decisions.

Based on the measured results, shown in Fig. 7, we can make the following conclusions:

1. At the expense of increasing the execution time, on average, by 15%, 3DA\(_e\) slightly outperforms 3DA (See Fig. 7b, 7d, 7f and 7h).
2. EAC does not always improve the performance of 3DA (See Fig. 7b for instance).
3. EVC and (to a lesser extent) EVC\(_{2P}\), which consider the time domain, outperform 3DA. On average EVC reduces the percentage of missed deadlines by 9%, improves the scheduling feasibility 20% and exploits the computation capabilities delivered by the FPGA by a factor of 5. However, the fact that EVC\(_{2P}\) analyzes the state of the sandbox at only two instants of time reduces its effectiveness (See Fig. 7a, 7b, 7c, 7e, 7f and 7g).
Table 1: Simulated tasks-set parameters

<table>
<thead>
<tr>
<th>$\phi_1$</th>
<th>$h_{x,\text{min}}$</th>
<th>$h_{x,\text{max}}$</th>
<th>$h_{y,\text{min}}$</th>
<th>$h_{y,\text{max}}$</th>
<th>$t_{E,\text{min}}$</th>
<th>$t_{E,\text{max}}$</th>
<th>$D_{\text{min}}^*$</th>
<th>$D_{\text{max}}^*$</th>
<th>$U_{\text{ICAP}}$</th>
<th>$U_{\text{COMP}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>6</td>
<td>24</td>
<td>16</td>
<td>48</td>
<td>2 $t_{A,i}$</td>
<td>25 $t_{A,i}$</td>
<td>100 $t_{A,i}$</td>
<td>500 $t_{A,i}$</td>
<td>0.20</td>
<td>0.16</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>12</td>
<td>30</td>
<td>48</td>
<td>80</td>
<td>2 $t_{A,i}$</td>
<td>25 $t_{A,i}$</td>
<td>100 $t_{A,i}$</td>
<td>500 $t_{A,i}$</td>
<td>0.20</td>
<td>0.44</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>18</td>
<td>42</td>
<td>80</td>
<td>112</td>
<td>2 $t_{A,i}$</td>
<td>25 $t_{A,i}$</td>
<td>100 $t_{A,i}$</td>
<td>500 $t_{A,i}$</td>
<td>0.20</td>
<td>0.93</td>
</tr>
<tr>
<td>$\phi_4$</td>
<td>30</td>
<td>42</td>
<td>112</td>
<td>160</td>
<td>2 $t_{A,i}$</td>
<td>25 $t_{A,i}$</td>
<td>100 $t_{A,i}$</td>
<td>500 $t_{A,i}$</td>
<td>0.20</td>
<td>1.63</td>
</tr>
<tr>
<td>$\phi_5$</td>
<td>6</td>
<td>24</td>
<td>16</td>
<td>48</td>
<td>5 $t_{A,i}$</td>
<td>50 $t_{A,i}$</td>
<td>100 $t_{A,i}$</td>
<td>60 $t_{A,i}$</td>
<td>0.68</td>
<td>0.78</td>
</tr>
<tr>
<td>$\phi_6$</td>
<td>12</td>
<td>18</td>
<td>32</td>
<td>48</td>
<td>10 $t_{A,i}$</td>
<td>25 $t_{A,i}$</td>
<td>50 $t_{A,i}$</td>
<td>60 $t_{A,i}$</td>
<td>0.90</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Figure 7: R3TOS performance with simulated task-sets of Table 1
4. With high workloads \((U_{COMP} \geq 1)\) or when the chip is damaged, the allocation of too big tasks can be discarded by the scheduler based on the MER size, which is automatically given by the EAC heuristic. The saved time can be used for including the time dimension in the analysis, resulting in affordable computation times of EVC and EVC\(_{2P}\) heuristics. In fact, in these situations EVC and EVC\(_{2P}\) can be computed even faster than 3DA (For instance see Fig. 7d and 7h).

7 Conclusions

With the objective of building FPGA-based reconfigurable computers enabled to be used in future deep-space applications we have presented the R3TOS approach. R3TOS schedules and allocates real-time hardware tasks onto partially reconfigurable FPGAs in such a way that damaged resources by TID are circumvented. As a result of the continuous reconfiguration of the tasks, SEEs are scrubbed in the configuration memory as well. In this paper, we have introduced three fault-aware heuristics for deciding the best position to place the hardware tasks: EAC, EVC\(_{2P}\) and EVC. We have demonstrated that these heuristics reduce the amount of missed deadlines and better exploit the computation capabilities delivered by the FPGA with regard to related approaches in the presence of faults.

References


