Availability Analysis for Satellite Data Processing Systems Based on SRAM FPGAs

Felix Siegle, Tanya Vladimirova, Jørgen Ilstad, and Omar Emam

***Abstract*—This paper presents a novel methodology, which allows a systematic availability analysis of satellite payload data processing systems implemented on SRAM-based FPGAs. The methodology allows (i) comparison of different Fault Detection, Isolation and Recovery (FDIR) schemes and (ii) prediction of the expected system availability in a particular radiation environment. Furthermore, it advances the state of the art by analysing embedded Block RAMs and employing a novel fault injection algorithm that enables more complex stochastic models. The applicability of the method is demonstrated by a case study representing a high availability payload data processing application. Since Block RAMs are also taken into account, the availability prediction precision is greatly increased. It is shown that the reliability prediction for two border cases, in which the Block RAMs are either ignored or assumed to be fully susceptible, can differ as much as 75%. With the proposed Block RAM profiling tool it is possible to determine a realistic reliability figure that is eventually required for the accurate estimation of the system availability.**

***Index Terms*—Availability Analysis; Fault Injection; Radiation Environment; Reconfigurable Hardware; Space Technology; SRAM-based FPGAs; Stochastic Petri Nets**

I. Introduction

INCREASED processing capabilities are required for payload data processing on board spacecraft. Modern approaches necessitate that the data is processed in real time while being streamed from a data source to a data sink, e.g. from a payload instrument to a mass memory device. On its way the data is possibly processed by several processor units in series, performing computationally intensive digital signal processing (DSP) tasks including reduction of the data volume, which must be stored on board and later transmitted to Earth. In a technology assessment NASA scientists found that Static Random-Access Memory (SRAM) based Field Programmable Gate Arrays (FPGAs) are best suited for hardware acceleration of such high-performance tasks due to their flexibility, parallel architecture and embedded DSP blocks compared to single board computers and DSP processors [1]. Apart from the increase in performance, SRAM-based FPGAs offer the capability of being reconfigured, which is a valuable feature allowing remote upgrade and repair in space missions.

However, except of the radiation hardened Xilinx Virtex-5QV device, all modern SRAM-based FPGAs are prone to non-destructive radiation effects, regardless of their grading (space, defence, or commercial). When dealing with such FPGAs in space applications the common types of faults that must be mitigated are caused by Single Event Upsets (SEUs) that can change the state of a bi-stable element. SEUs are triggered by heavy ions and protons and result from ionisation by a single energetic particle or the nuclear reaction products of an energetic proton. The ionisation induces a current pulse in a p-n junction whose charge may exceed the critical charge which is required to change the logic state of the element. As a result, the value of a memory bit can be flipped [2]. SEUs occur in both the configuration memory and the user memory, i.e. in embedded Block RAM cells and flip-flops. If and when a fault ultimately manifests itself as a failure depends on the function of the affected memory cell and the dynamic behaviour of the application, e.g. only a fault in a memory cell that is actually used in the design can potentially lead to a failure.

Due to the susceptibility of SRAM-based FPGAs to radiation effects a number of mitigation techniques have been designed to protect them in the harsh space radiation environment. A Fault Detection, Isolation and Recovery (FDIR) methodology was proposed in [3], [4], which is an attempt to provide a generalised and unified mitigation framework. With such a methodology, failures propagating through the circuit are detected by appropriate monitors. Then, the circuit is automatically repaired by executing some kind of a reconfiguration procedure. Depending on the redundancy scheme applied the circuit might not function correctly in the periods, in which the circuit is under repair and therefore the system might suffer from down time. The probability that a system functions correctly is called *steady state availability*, often defined as:

$$A = \frac{T_m}{T_m + T_d} \tag{1}$$

where T_m is the mission duration and T_d the observed down time, which is the sum of the time required to detect and recover failures.

Availability analysis is an integral part of space product assurance procedures and a dedicated European Cooperation for Space Standardization (ECSS) standard is available for European space projects [5]. This standard gives several definitions for availability. In the rest of this paper the term availability stands for *inherent* steady state availability, i.e. downtime during idle periods does not contribute to the overall system availability. According to the ECSS standard, important objectives of availability analysis include the verification of whether or not a system conforms to the availability requirements. In addition, it also identifies unavailability contributing

F. Siegle and T. Vladimirova are with the Department of Engineering of University of Leicester, UK.

J. Ilstad works at ESTEC, the European Space Research and Technology Centre of the European Space Agency (ESA), Noordwijk, Netherlands.

O. Emam works at Airbus Defence and Space, Stevenage, UK.

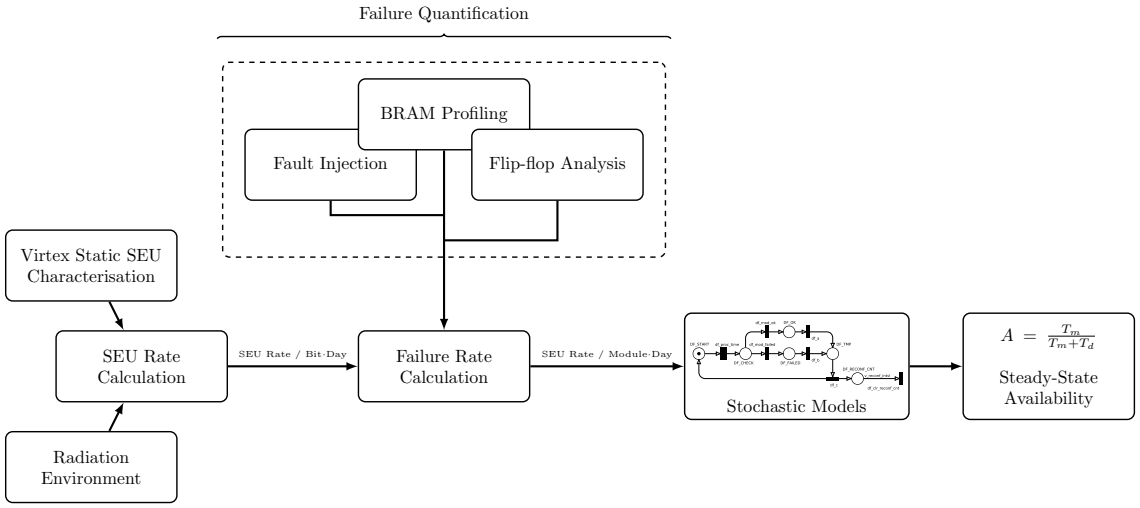

Fig. 1. Block diagram of the proposed availability analysis methodology.

factors in order to quantify their impact on (i) the decision-making process, and (ii) the risk evaluation, reduction and control. The implementation of such an availability analysis method with regards to SRAM-based FPGAs has two significant advantages:

- It gives an estimation of the number of failure occurrences that are to be expected either during a specific mission time frame or the overall mission lifetime.
- It allows the comparison of different redundancy and recovery schemes, which is especially important for applications where an optimal mitigation approach must be found trading-off power, area and reliability overheads.

In this paper, a methodology is presented that allows availability analysis of so-called *stream processors*, implemented on SRAM-based FPGAs. Even though Xilinx Virtex-4 devices are used as an example, the methodology is applicable to all SRAM-based FPGAs, which are not fully radiation hardened and thus are prone to radiation-induced soft errors.

The rest of this paper is structured as follows: Section II outlines the proposed availability analysis methodology. Section III describes an architecture of a stream processor as well as a particular image compression implementation, which is used as a case study for a high availability payload data processing application. In Section IV, the different failure modes for the configuration memory, the Block RAMs and the flip-flops are quantified using the case study stream processor implementation. In Section V, the quantified failure modes are used as input parameters for availability prediction with stochastic Petri nets models, which estimate the processors availability for a specific orbit in space. For illustration purposes, the orbits of two exemplary European space missions, Sentinel-3 and Galileo, are employed in the case study. Finally, the increase of the prediction precision, which is made possible by our novel Block RAM profiling tool, is illustrated in Section VI.

II. Availability Analysis Methodology

McMurtrey et al. use Markov models to estimate the reliability of Triple Modular Redundancy (TMR) systems [6]. Among other things, the authors investigate how multiple TMR partitions increase the reliability. Ostler et al. present a reliability analysis of SRAM-based FPGAs in [7]. The methodology takes particular radiation environments into account and is based on fault injection experiments. Kastil et al. present a dependability analysis of their fault tolerant systems in [8]. Applications hosted on SRAM-based FPGAs are partitioned into functional units to which different redundancy configurations can be applied. An automatic tool creates a Markov model of the overall system. Martin et al. also uses Markov chains in [9] to model the availability that can be achieved with different scrubber implementations. In two recent publications, Hoque et al. deal with probabilistic model checking techniques for aerospace applications [10], [11]. The authors use a continuous-time Markov reward model to determine the availability of the configuration memory when scrubbing is applied as recovery technique. Stochastic Petri nets are used to model the availability of the configuration memory for the first time in [12].

The aforementioned works take only the configuration memory into account. However, other FPGA building blocks, especially Block RAMs, cannot be ignored due to the large amounts of buffers that are typically used by streaming applications. Thus, we present a more complete, yet easy to follow availability analysis methodology in this paper, which also takes Block RAM and flip-flop upsets into account. Furthermore, the proposed methodology advances previous work since it also includes:

- A novel fault injection methodology that distinguishes between several failure modes.
- The usage of extended deterministic and stochastic petri nets (eDSPN). We believe that this class of petri nets arrange stochastic models more clearly than Markov

chains. As a result, complex models are easier to build, maintain and understand. Furthermore, eDSPNs allow one expolynomially timed transition in each marking. Thus, time transitions can be based on complex distributions, e.g. deterministic delay, uniform distribution, or triangular distribution.

- The usage of a novel Block RAM profiling tool that gives a more accurate estimate of the used Block RAM cells.

A block diagram of the proposed methodology is depicted in Figure 1. First, the SEU rates per bit-day are determined for the configuration memory, the Block RAMs and the flip-flops of a particular Virtex-4 device in a specific orbit. The calculations are based on static SEU characterisation data that was gathered from accelerated radiation testing and published by Xilinx and NASA [13]. The radiation environment, i.e. the heavy ion and proton fluxes of the orbit, is calculated using radiation models standardised in the European standard ECSS-E-ST-10-04C [14]. The calculated SEU rates give the probability of a bit flip in one single memory element.

However, to compute the availability of a stream processor, the SEU rate per day and stream processor must be known and not just the rates for the memory elements. Thus, the bit upset rates must be scaled by the number of sensitive memory elements that first must be determined. The sensitivity of the configuration memory is quantified by randomly injecting faults into the memory using a fault injection system whereas the sensitivity of the Block RAMs is estimated using a custom-built memory profiling tool. For the usually small number of flip-flops it is simply assumed that all of them are sensitive. Once the sensitivity of all memory types is quantified, the SEU rate for the whole stream processor is known.

To analyse the availability, the chosen failure recovery approach must also be considered. The typical recovery approach for the configuration memory is often referred to as *scrubbing* [15], a mitigation technique that refreshes the memory content from time to time with a correct bitstream. In contrast to partial reconfiguration, scrubbing does not affect the user memory and can thus be executed during circuit operation. For some failures, however, such a memory refresh is not sufficient because the failure could have already manifested itself in the user logic (e.g. state machines, counters and similar state-dependent logic). Then, the user logic must be additionally reset. For user memory (RAMB16s and flip-flops), two basic failure modes exist: (i) a failure can either propagate to an output of the FPGA or (ii) it can get trapped in a feedback loop. In the first case, the failure is of a transient nature only and no further recovery actions are required. In the latter case, the user logic must be reset to a safe initial state.

If one wants to model the availability of a processor to which a specific recovery approach is applied, e.g. frame-based scrubbing [15] or partial reconfiguration, the SEU rates for the aforementioned failure modes must be calculated separately. For instance, some configuration memory bit upsets only require a memory scrub, while others need an additional circuit reset. Therefore, we need to determine the SEU rates for all these cases. This is accomplished for the configuration memory by our fault injection system that automatically tries to recover from a failure in several ways and is thus

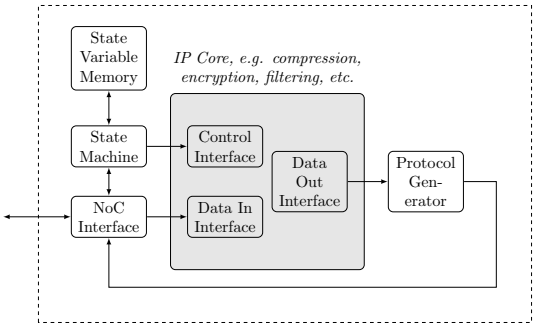

Fig. 2. Example of a stream processor.

able to classify and quantify the sensitive bits, described in Section IV-A. For the RAMB16 cells and flip-flops, a custom-built netlist parser tool analyses how many of these memory elements propagate into feedback loops and how many of them do not. Then, the SEU rates for both cases can be calculated.

Once all SEU rates of interest are available, the steady-state availability is calculated using *Stochastic Petri Nets*. Several models are proposed, which take different redundancy and failure recovery approaches into account but in principle, this modelling technique is flexible enough to support all kind of FDIR approaches.

III. Stream Processor Architecture

The architecture of a typical FPGA based stream processor is shown in Figure 2. A soft Intellectual Property (IP) core, the purpose of which is to accelerate a desired DSP functionality, is embedded into the stream processor. The stream processor further comprises a Network-on-Chip (NoC) interface for the data exchange, some state machine logic and a memory for state variables. Input control words are interpreted by the state machine whereas input data words are directly fed into the accelerator IP core. An additional memory holds all variables necessary to configure the IP core. If the processing pipeline uses a specific protocol, a protocol parser and/or protocol generator may be added to the inputs and outputs of the core.

The employed IP cores are of a passive nature, e.g. they represent hardware accelerators that are designed to be connected as slaves to a Central Processing Unit (CPU) bus. With the additional logic in the stream processor, however, the cores become intelligent enough to process incoming data without the interaction of a CPU, solely by interpreting data and command words in the network input stream. Furthermore, suitable IP cores process input data block-wise. For instance, such a block could be a line of pixels, a full image or a series of images (think of hyperspectral image compression and similar applications). As a rule of thumb, it should be possible to reset the core after each data block without affecting the processing of subsequent blocks.

The stream processor itself is protected against soft errors by means of modular redundancy and therefore all FPGA building blocks within the processor are multiplied. This is also beneficial for the embedded Block RAMs. Rollins et al. [16] showed that redundancy is the only effective mitigation approach because other mitigation techniques (like error detection and correction codes) rely on additional logic that must be

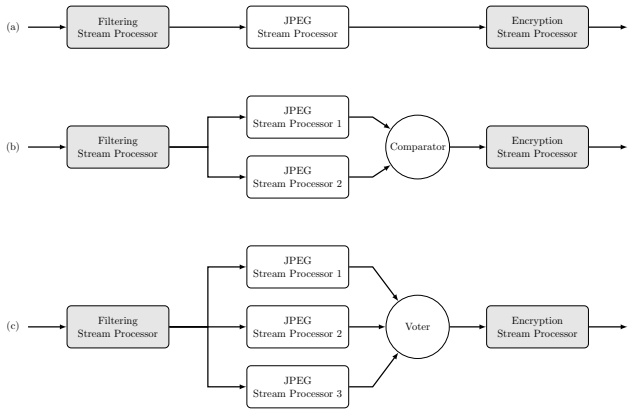

Fig. 3. Example of a typical satellite payload streaming architecture with a JPEG stream processor in different redundancy configurations: (a) No redundancy, (b) Duplication with Comparison, (c) Triple Modular Redundancy.

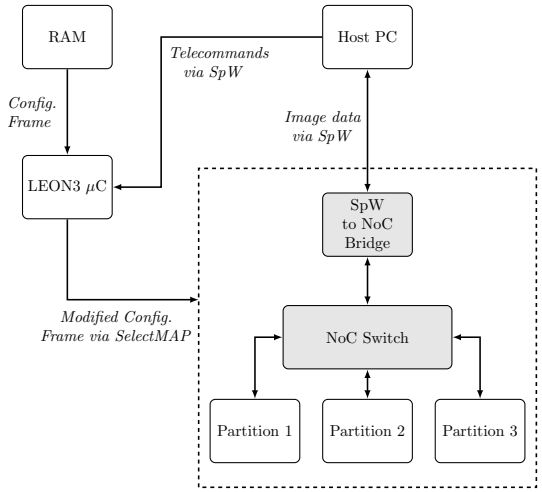

Fig. 4. Fault injection system with a Virtex-4 device under test (dashed box).

implemented as part of the potentially unreliable FPGA fabric. In theory, internal Error-Correcting Code (ECC) logic found in some types of SRAM-based FPGAs could be used instead. However, since this logic is usually not hardened by design, it is not taken into account by the example models presented in this paper.

The work presented in the following sections is based on a case study that represents a typical satellite payload streaming architecture, illustrated in Figure 3, in which a Joint Photographic Experts Group (JPEG) image compression IP core [17] is embedded into a stream processor. The input data is a raw 24 bit video stream with an image geometry of 640x480 pixels. The network stream is delivered through a NoC implementation called SoCWire [18], which is based on SpaceWire, a popular flow-controlled point-to-point network protocol [19]. One data block corresponds to one image frame. Since the subsequent processing pipeline uses a low-level Consultative Committee for Space Data Systems (CCSDS) protocol [20], a protocol packet generator has been added to the output of the IP core which segments the compressed image into 1 kB sized data chunks before transmitting it to the next processor. The stream processor is implemented on a Virtex-4 SX55 FPGA by Xilinx, a device that can be purchased in a space-qualified version. The methodology could also be applied to other SRAM-based FPGAs though.

IV. Quantification of Sensitive Memory Elements

The following three main types of memories, available in radiation-tolerant SRAM-based FPGAs, must be considered in a detailed availability analysis:

- Configuration memory: The configuration of the FPGA is stored in volatile SRAM memory cells. A part of the memory stores control bits, which define the routing resources and the content of the look-up tables (LUTs). Thus, a radiation-induced upset in one of these sensitive memory elements can manipulate the circuit in such a way that a failure becomes measurable at its output.
- Block RAM: In many streaming applications, a large amount of embedded RAM blocks is utilised by the user logic. If a value, which was falsified by an upset, is read from such a memory, it can manifest itself as a failure and either propagate to the output of the circuit or get trapped in a feedback loop within the circuit.
- User flip-flops: In most cases, flip-flops do not contribute as much as the aforementioned memory types to the overall cross-section due to their typically low number. However, since flip-flop upsets can get trapped in feedback loops too (which requires some failure recovery action), they are not neglected for the proposed availability analysis method.

A. Configuration Memory (via Fault Injection)

Although analytic tools for the quantification of sensitive configuration memory bits are available, for instance the tools proposed by Sterpone et al. [21], fault injection is probably still the most reliable technique for this task due to its capability to take into account the dynamic behaviour of the application. With this capability, the following steps can be accomplished:

- The quantification of three types of sensitive configuration bits. Those which lead to failures that:
  - can be scrubbed ($F_{C/S}$).
  - need an additional circuit reset after scrubbing ($F_{C/SR}$).
  - can only be repaired by a repeated partial reconfiguration of the stream processor ($F_{C/Re}$).
- The creation of a database that stores information about configuration bits, for which the failure mode is exactly known. The database can later be used to validate FDIR techniques.

In the course of this work, a fault injection system has been implemented as outlined in Figure 4. The FPGA floorplan is divided into three partition, which are interconnected using a NoC routing switch. A network bridge allows the communication with each partition via SpaceWire (SpW). The fault injection campaign is controlled by two instances:

- An embedded software running on a LEON3 microprocessor IP core [22] is responsible for the fault injection

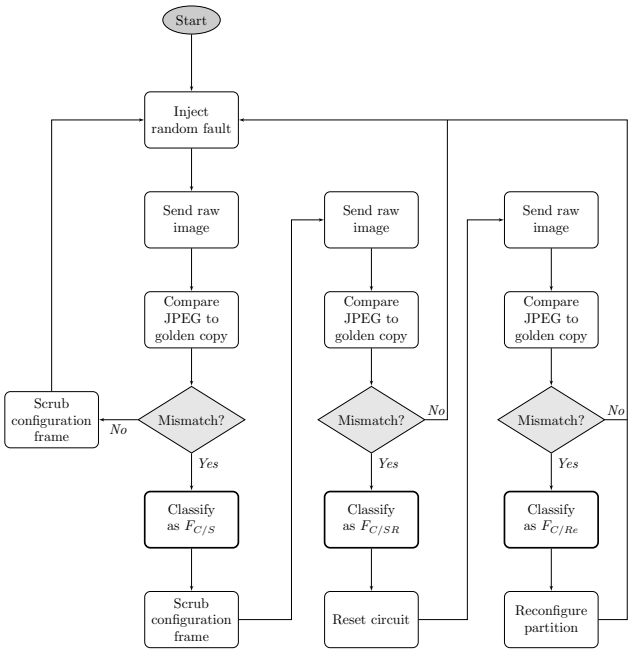

Fig. 5. Flow chart of the novel fault injection and failure classification algorithm.

itself. This is done by first retrieving a desired single configuration frame from RAM, to which the partial bitstream under investigation has been copied before. Then, a bit in this configuration frame is flipped before it is finally downloaded to the FPGA via the SelectMAP interface.

- A C++ software running on the host PC controls the fault injection campaign by sending telecommands to the embedded software system.

A flow chart of our novel fault injection and failure classification algorithm can be seen in Figure 5. After fault injection, the host PC software sends a full data block to the stream processor (here, a full image to the JPEG processor) and is then waiting for a response from the processor. The returning data is reassembled to a JPEG file and compared to a golden copy of that image. If the JPEG files are identical, the configuration frame is scrubbed to avoid an accumulation of faults in the configuration memory and the next random fault is injected. If there was a mismatch, however, the sensitive bit is classified as $F_{C/S}$. The configuration frame is scrubbed and the host PC software sends another data block to the stream processor to check if the scrubbing was successful. If so, no additional actions are required and the next random fault can be injected. If not, the classification of the sensitive bit is updated to $F_{C/SR}$ and the circuit is reset by the embedded system (again by sending an appropriate telecommand). Then, the procedure is repeated. If the reset was successful, the next random fault is injected. Otherwise, if the stream processor is still not returning a correct answer, the classification is updated to $F_{C/Re}$. The classification for each injected bit is stored in a Structured Query Language (SQL) database for later analysis. Furthermore, in case of a detected failure, the returned data is written to hard disk (here, the corrupted JPEG image).

Faults are injected just before a full raw image is sent to the JPEG processor. In reality, however, a fault could also occur during processing. Preceding fault injection was chosen intentionally as it represents a worst-case scenario that maximises the detection coverage of sensitive bits.

For our proof of concept implementation random images were used during the test campaign. It was later found that the sensitivity of a very small number of sensitive bits also depends on the data processed (e.g. very dark or light images). Therefore, it must be mentioned that data-dependency was not taken into account during the fault injection campaign.

The automatic fault injection technique is rather slow since full data blocks are sent through the stream processor. However, this is important to make sure that all of the states of the circuit are traversed. It might take some time until the fault manifests itself as a failure by propagating to the circuit's output and as a consequence, the failure could stay undetected if the state space is not fully covered. In case of the JPEG processor, only four faults can be injected per second. But, the technique allows a very detailed classification of the different failure modes, using a real application. It is also non-intrusive, i.e. no circuit modifications are required for the fault injection campaign. Therefore, a fully placed and routed design could be analysed in a very late design phase, an important aspect for the typical qualification process in space engineering projects. And even if only a limited number of faults can be injected, useful results can be still obtained by using statistical theory. Since the outcome of the fault injection campaign has a binomial distribution (recall the classic urn problem), the number of sensitive configuration bits can be estimated by a rather low number of statistical samples. To get an idea how many samples are necessary, the formula for Wald confidence intervals [23] can be used:

$$p = \hat{p} \pm z_{1-\frac{1}{2}\alpha} \cdot \sqrt{\frac{1}{n}\hat{p}(1-\hat{p})} \tag{2}$$

where $\hat{p}$ is the proportion of success (e.g. the ratio of sensitive bits to all bits), n the number of trials, and z the $1-\frac{1}{2}\alpha$ quantile of a standard normal distribution with α as error percentile. Suppose, a typical 95% confidence interval ($z = 1.96$) with an interval width of $\pm 0.2\%$ ($p = \hat{p} \pm 0.002$) is desired while 15% of all configuration bits are assumed to be sensitive ($\hat{p} = 0.15$). In this case, n becomes 122,451. For the JPEG processor, a value of $n = 150,000$ fault injections is chosen because then, a campaign can still be conducted over the course of one night (less than 11 hours).

Once the fault injection results are available, a more conservative approach for the final estimation of the proportion is chosen. The so-called Clopper-Pearson interval is known to have never less than the given coverage rate [23] (here 95%) which makes it a good candidate for worst-case estimations.

Table I lists the fault injection results for the processor placed on partition 1 (the partial bitstream of this processor comprises 3,735,264 configuration bits). The number of sensitive bits is subdivided into two categories. The first category includes the three failure modes $F_{C/S}$, $F_{C/SR}$ and $F_{C/Re}$. The second category distinguishes between data errors and

TABLE I
FAULT INJECTION RESULTS

	Absolute	95% Confidence Rel. Min.	95% Confidence Rel. Max.	95% Confidence Abs. Min.	95% Confidence Abs. Max.
Sensitive bits	21,051	13.86%	14.21%	517,655	530,811
$F_{C/S}$	20,080	13.21%	13.56%	493,605	506,503
$F_{C/SR}$	961	0.60%	0.68%	22,446	25,487
$F_{C/Re}$	10	0.00%	0.01%	119	458
JPEG errors	14,834	9.74%	10.04%	363,766	375,077
Network errors	6,217	4.04%	4.25%	151,067	158,628

network errors. Data errors affect only the application data itself, i.e. the JPEG image, whereas network errors affect the network protocol (falsified protocol headers, missing packets etc.). The first column lists the absolute outcomes of the 150,000 fault injection experiments. In column 2 and 3, the lower and upper bound of the confidence interval is given. In the last two columns, these proportions are scaled to the real size of the partition using its total number of bits. In theory, the percentage figures of the first category ($F_{C/S}$ + $F_{C/SR}$ + $F_{C/Re}$) and the second category (data errors + network errors) should both add up to the overall percentage of sensitive bits. However, as can be seen in Table I, the values do not add up precisely. The reason is that the confidence intervals for each failure mode are calculated separately. How precise the prediction of a confidence interval is depends on the number of measured samples. Therefore, if 20,080 failures are measured for $F_{C/S}$, the predicted confidence interval is more precise than the confidence interval for $F_{C/Re}$, for which only 10 failures were measured.

The initial guess of 15% sensitive bits was quite accurate, with a real upper bound of 14.21%. More than 95% of the sensitive bits are of type $F_{C/S}$, i.e. the processor can simply be recovered by scrubbing. Approximately 4.5% of the sensitive bits are of type $F_{C/SR}$, i.e. the circuitry of the processor must be reset after scrubbing. These kind of failures occur after state-dependent logic gets affected by a fault. For instance, if a state machine moves to an illegal state due to a configuration memory upset, scrubbing will remove the upset but the state machine is still in the illegal state. Then, only a circuit reset can recover the processor. A very small number of bits is of type $F_{C/Re}$, i. e. failures that can only be repaired by a partial reconfiguration. It was found that this critical failure mode occurs due to Read-Only Memories (ROMs) that are implemented in Block RAM. If the Block RAM interconnect is affected by an upset, ROM content is likely to be overwritten.

Roughly 70% of the failures affect the data itself, here the JPEG image. The other 30% of failures are network failures. Since the outcome of each fault injection experiment is stored in a database, the failure modes can later be reproduced to validate possible FDIR techniques.

B. Embedded Block RAM (via Memory Profiling)

Streaming applications often utilise huge amounts of embedded Block RAM elements, e.g. as First In, First Out Buffers (FIFOs) or ROMs. As a consequence, the number of sensitive bits inside these elements can be rather large, sometimes comparable to the total number of sensitive configuration bits. Therefore, upsets in Block RAMs cannot be neglected in an availability analysis. However, the estimation of the Block RAM utilisation is complicated if not impossible with the standard Xilinx toolchain. If only the absolute number of Block RAMs is taken into account, the number of susceptible bits would be way too overestimated. For instance, the JPEG stream processor utilises 84 Block RAMs. With 18 KBits per RAM, 1,548,288 sensitive RAM bits could be assumed. In reality, however, the number of sensitive bits depends on the dynamic behaviour of the application. First, many rows within a Block RAM are simply not used. Secondly, faults affecting memory rows which are not read out after the fault occurred will also not lead to a failure.

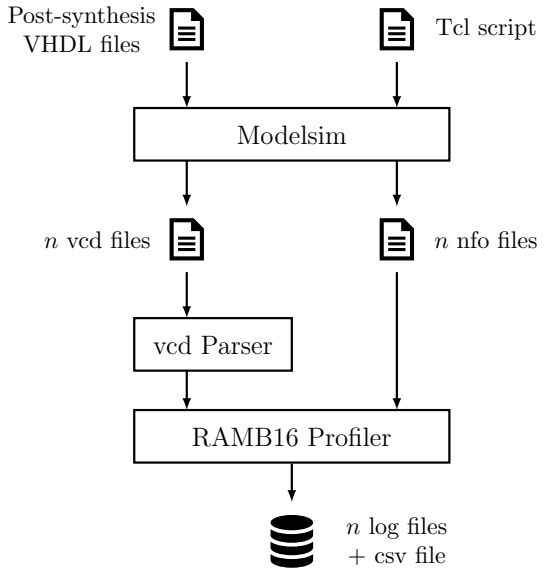

Fig. 6. Block RAM profiling tool.

In the course of this work, a novel tool for the dynamic profiling of Block RAMs has been developed. The basic idea, as outlined in Figure 6, is as follows: The processing of one full data block is simulated using a post-synthesis simulation model, e.g. in ModelSim. During simulation, a `vcd`-file is created for each Block RAM which records the changes on the address and write enable lines of both ports for the whole simulation run. Furthermore, some information about the Block RAM is stored in a `nfo`-file, including the read and write width configuration, the write mode (write-first, read-first, no-change) as well as a flag which signals if the data in and data out lines of both ports are actually connected. All these information is gathered and stored automatically by a Tcl script which is executed within the simulator.

Once all files are available, one of our tools is parsing the `vcd`-files into C++ objects where the time-value pairs of the signals are stored. Then, the memory profiling begins: The tool steps through the simulation run to identify read and write accesses. This is accomplished by searching for signal changes in the address and write enable lines of connected ports. While write accesses can easily be identified by the corresponding write enable signal, read accesses must be guessed. This is

TABLE II
SUMMARY OF THE BLOCK RAM PROFILING

	Absolute	Relative
Available Bits	1,548,288	100%
ROM bits	25,216	1.63%
RAM bits	521,264	33.67%
RAM bits (corrected by τ_S)	220,003	14.21%

done by interpreting an address change, which occurs while the write enable signal is low, as a read access. The approach is slightly conservative but without any further knowledge about the circuit, it must be assumed that any valid data at the output of a Block RAM is also used in the design.

The memory profiling tool has three main tasks. First, it must identify the real utilisation of the Block RAM. Since the tool keeps track of the memory addresses that were accessed during the simulation run, the number of used memory bits can easily be calculated. Secondly, it must identify which memory rows, respectively addresses, are used as ROMs and which are used as RAMs. This is achieved by interpreting rows, which are only accessed for reading but not for writing, as ROM rows. Thirdly, the tool must determine a correction factor for each RAM row that takes the time spans into account in which a fault cannot manifest itself as a failure.

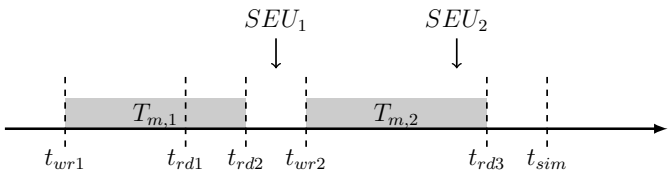

Fig. 7. Example for the calculation of the correction factor.

Take the timeline in Figure 7 as an example, where some read and write accesses of one memory address are shown. The first fault SEU_1 occurs between a read access t_{rd2} and a subsequent write access t_{wr2}. Since the memory row is overwritten with a new value, the fault cannot manifest itself as a failure. The second fault SEU_2, however, manifests itself as a failure because the memory row is read out at t_{rd3}. All N time spans $T_{m,n}$ in which a memory row m is susceptible (grey boxes in Figure 7) are first accumulated by the memory profiling tool. Then, the results of all M memory rows are averaged. Finally, dividing the averaged value by the total simulation time leads to the correction factor τ_S:

$$\tau_S = \frac{\sum^M \sum^N T_{m,n}}{M \cdot t_{sim}} \tag{3}$$

τ_S can only be calculated for RAM rows because obviously, ROM rows are never overwritten by the user logic. RAM rows in Block RAMs that are configured in read-first mode must be handled in a similar way: In this mode, a memory row is read out just before the write access and thus the whole time span between the last and the current write access must be assumed to be susceptible to upsets, i.e. $\tau_S = 1$.

The memory profiling tool outputs one `log`-file per Block RAM with detailed profiling information and one `csv`-file with a summary of all results. For the case study, we simulated a typical run. The final results are listed in Table II: In relation to the total number of bits in all Block RAMs, the real number of susceptible bits is much lower. Less than 34% of all bits are actually accessed. Taking the correction factor τ_S into account, this number is further reduced to only 14.2%.

With the profiling tool, we are able to quantify the number of susceptible Block RAM bits. However, the quantification of the failure modes is not done yet. As mentioned in the introduction, a failure can either propagate to an output of the FPGA or it can get trapped in a feedback loop. While no special recovery action is necessary in the first case, the latter failure mode necessitates a circuit reset. To get an idea how many of the failures propagate into feedback loops, we developed a netlist parser in C++. The tool takes an `edif`-file as input and parses the netlist into a directed graph of C++ objects. Then, the tool is able to traverse the graph using a Depth-first search (DFS) algorithm. The netlist is flattened in such a way that the graph only comprises primitives (flip-flops, RAMB16s etc.) as vertices. Within each vertex, all inputs are connected to all outputs. In other words, it is assumed that a failure which arrives at an input of a primitive can propagate to any output of that primitive.

The implemented algorithm takes a set of primitives, here RAMB16 blocks, as starting points. From the starting points, all possible edges and vertices are explored until either i) an output pin of the FPGA is found or ii) an input pin of a primitive is found which was already visited in the current exploration. In this case it is clear that the failure got caught in a feedback loop. If at least one output pin of the source primitive leads to a feedback loop, the second failure mode (necessitating a reset) is assumed for that primitive.

Interestingly, nearly all (83 out of 84) Block RAMs inside the JPEG processor propagate into feedback loops. For the worst case, we simply assume that all upsets in susceptible Block RAM necessitate a circuit reset. It is worth to mention that several other circuits has been analysed with the same method with varying results. While some highly pipelined circuits do not comprise any feedback loops at all, this JPEG processor has an especially high ratio of state-dependent logic.

C. User Flip-Flops

The number of user flip-flops can easily be determined using the standard Xilinx toolchain or alternatively, by counting the flip-flop primitives in the netlist graph. For instance, the JPEG processor comprises 3,349 flip-flops. But again, it is unclear how many of the flip-flops propagate into feedback loops. Therefore, the netlist parser tool is used a second time, using flip-flops primitives as starting points rather than Block RAM primitives. Unsurprisingly, the results reveal a similar high ratio: 3,268 out of 3,349 flip-flops propagate into feedback loops. For the worst case, it is again assumed that all upsets in flip-flops necessitate a circuit reset.

V. AVAILABILITY ANALYSIS

A. Radiation Environment

First, the bit upset rates for the configuration memory, the Block RAMs, and the flip-flops in a particular radiation

TABLE III
BIT UPSET RATES [1/BIT·DAY] FOR THE CASE STUDY

Memory Type	Sentinel-3	(SPE)	Galileo	(SPE)
Configuration memory	7.3E-07	2.9E-04	2.6E-07	9.4E-04
Block RAM	2.3E-06	1.1E-03	5.8E-07	3.7E-03
Flip-flops (all 0)	7.0E-07	2.0E-04	4.2E-07	6.8E-04
Flip-flops (all 1)	4.3E-06	1.8E-03	1.0E-06	5.9E-03

TABLE IV
MTBF [DAYS] FOR ALL FAILURE MODES

Mode	Sentinel-3	(SPE)	Galileo	(SPE)	Recovery Action
$F_{C/S}$	2.7	0.01	7.6	0.002	Reconf. OR Scrubbing
$F_{C/SR}$	54.0	0.14	150.3	0.04	Reconf. OR (Scrubbing AND Reset)
$F_{C/Re}$	3003.7	7.6	8366.6	2.3	Reconfiguration
F_{ROM}	17.4	0.04	68.4	0.01	Reconfiguration
F_{RAM}	2.0	0.004	7.8	0.001	Reconf. OR Reset
F_{FF}	119.0	0.3	403.2	0.09	Reconf. OR Reset

environment must be determined. The required heavy-ion and proton fluxes are calculated according to ECSS-E-ST-10-04C [14]: For cosmic rays, the ISO-15390 GCR model is used while solar minimum conditions are assumed. For trapped electrons, the AE8MAX model is used for Low Earth Orbits (LEO) and the MEOv2 model for Medium Earth Orbits (MEO). For trapped protons, the AP8MAX model is used for both, LEO and MEO. The fluxes during solar particle events (SPEs) are calculated using the CREME96 Worst Case 1 Day solar flare model. All fluxes are averaged over 100 orbits and thus take any anomalies into account. We followed common practice and assumed 100 mil of solid spherical aluminium shielding, although it was shown in the past that this assumption is an underestimation for most spacecraft electronics boxes [24].

Two missions of the European Space Agency (ESA) are considered for the case study: Sentinel-3 (LEO, 814.5 km altitude, 98.65° inclination) and Galileo (MEO, 23,222 km altitude, 56° inclination).

The resulting upset rates for the configuration memory, the Block RAMs and the flip-flops are listed in Table III. Under normal conditions, the bit upset rates for Sentinel-3 are higher than the rates for Galileo due to the influence of the inner radiation belt. The opposite is true for SPEs as the lower orbit offers a natural protection due to the geomagnetic shielding of the Earth.

B. Stochastic Petri Net Models

For modelling, stochastic Petri nets are used that can analytically be solved with the TimeNET 4.1 tool [25].

First, the total number of estimated sensitive bits of each failure mode (see Table I) is multiplied by the corresponding bit upset rate (see Table III, for F_{FF}, it is assumed that half of the flip-flops is storing a logical 1 and the other half a logical 0). Then, since the stochastic Petri net method requires Mean Time Between Failure (MTBF) values rather than failure rates, the inverses of the resulting failure mode upset rates are used as input parameters for the models. For instance, the MTBF value for $F_{C/S}$ on Sentinel-3 is calculated as follows:

$$\mathrm{MTBF}(F_{C/S}) = \left(\frac{7.3 \cdot 10^{-7}}{\text{bit} \cdot \text{day}} \cdot 506,503 \text{ bits}\right)^{-1} = 2.7 \text{ days} \tag{4}$$

The results for the three configuration memory failure modes $F_{C/S}$, $F_{C/SR}$ and $F_{C/Re}$, for the read-only Block RAM cells (F_{ROM}), the normal Block RAM cells (F_{RAM}), and the flip-flops (F_{FF}) are listed in Table IV. The last column lists possible recovery actions for each failure mode.

1) No Redundancy: First, the availability of a single stream processor is investigated. Without redundancy, no reliable failure detector mechanism exists and as a consequence, failure recovery must be done in a preventative manner, i.e. without any knowledge about the health status of the processor. A strategy solely based on scrubbing cannot be recommended due to the failures which can get trapped in the user logic. Consulting Table IV, two approaches can be applied, depending on how the input data is streamed into the processor:

a) Approach 1 - Periodic Partial Reconfiguration: If the input data is delivered in bursts, say 500 image frames, and there is some time between the bursts to do a partial reconfiguration, it is recommended to do such a periodic reconfiguration just before the processing of each burst. The benefit of this solution is its simplicity and the fact that the processor can be recovered from all possible failure modes in one go.

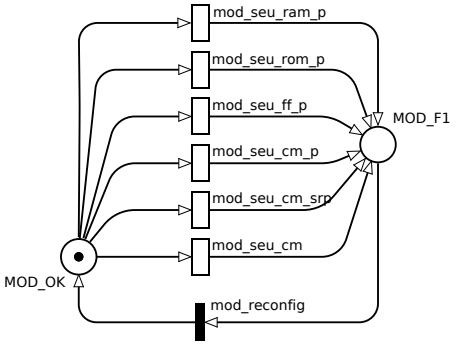

Fig. 8. No redundancy. Petri net no. 1.

All stochastic Petri net models have a similar structure with three basic nets that are linked to each other via logical conditions. For the aforementioned case, the first net is shown in Figure 8. The net models the health status of the processor: If the token is on place `MOD_OK`, the processor is working correctly. If one of the failure modes occurs (modelled by the exponential transitions `mod_seu_*`), the token moves into state `MOD_F1`. The failure recovery process is modelled by the immediate transition `mod_reconfig` which is only enabled when a trigger condition in the second net becomes true.

The second net, as depicted in Figure 9, models the data flow through the processor. A token represents the data block (here, a raw image frame). Transition `df_proc_time` is deterministic and models the time the data block resides in the processor. In case of the JPEG core, the frame rate is 20 Frames Per Second (FPS) and hence, a processing time of 50

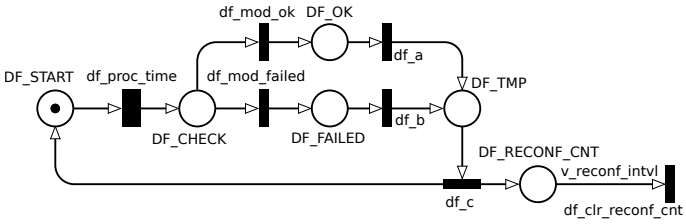

Fig. 9. No redundancy. Petri net no. 2.

TABLE V
PROCESSOR AVAILABILITY (APPROACH 1)

Reconf. Interval [1]	Sentinel-3	(SPE)	Galileo	(SPE)
5,000	0.9986	0.60	0.9996	0.24
500	0.99986	0.94	0.99996	0.82
50	0.999986	0.994	0.999996	0.98
5	0.999998	0.9993	0.9999995	0.998

[1] (images) with the processing time of 1 image being equal to 50 ms

ms is used. After processing, the first net is checked for the health status and the token moves either to place DF_OK or DF_FAILED. Then, the token moves back to the start place DF_START and another stochastic experiment begins. On the way back to the start place, the token is duplicated and one copy of the token moves to DF_RECONF_CNT. This place is a counter which counts the number of processed data blocks. If the reconfiguration interval threshold value is reached (here, the number of image frames per burst), the counter place is cleared and the failure recovery trigger condition becomes true.

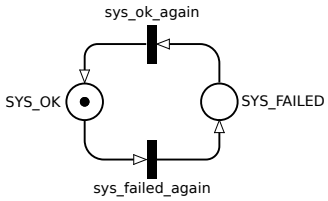

Fig. 10. No redundancy. Petri net no. 3.

A third net is used for the modelling of the overall processor availability. The currently processed data block can either be counted as OK or failed, see Figure 10. The transitions in this net are only enabled when the token is on place DF_OK or DF_FAILED in the second net. Ultimately, the steady state availability of the processor is determined by calculating the probability that the token is on place SYS_OK.

For approach 1, Table V lists the availability figures for different burst lengths. It becomes clear that a decent availability can be achieved for both orbits if the burst length, and therefore the duration between each partial reconfiguration, is not too long. However, it must be noted that the figures given here describe the so-called inherent steady-state availability, which does not include the planned downtime due to the periodic reconfiguration.

b) Approach 2 - Periodic Reconfiguration and Periodic Scrubbing: If the burst length is long or the probability of $F_{C/S}$ is rather high, it might be worth to combine the periodic partial reconfiguration with a more frequent periodic scrubbing. The Petri nets must be slightly modified to model this case: First, tokens moving via mod_seu_cm arrive at a new place called MOD_F2. Secondly, the token can leave that place back to MOD_OK via a new deterministic transition for scrubbing, analogous to mod_reconfig. To trigger that transition, a second counter is added to the second net, parallel to DF_RECONF_CNT.

TABLE VI
PROCESSOR AVAILABILITY (APPROACH 2)

Reconf. Interval [1]	Scrubbing Interval [1]	Sentinel-3	(SPE)	Galileo	(SPE)
5,000	500	0.999	0.671	0.99976	0.3394
	50	0.9991	0.6797	0.99978	0.3442
	5	0.99915	0.6805	0.999781	0.3446
500	50	0.99991	0.958	0.999976	0.870
	5	0.999914	0.960	0.999978	0.875
50	5	0.999991	0.996	0.999998	0.986

[1] (images) with the processing time of 1 image being equal to 50 ms

For approach 2, the availability figures for different reconfiguration and scrubbing intervals are listed in Table VI. If the frequency of the scrubbing is ten times higher than the frequency of the partial reconfiguration, a worthwhile increase of the availability can be achieved. But, if the scrubbing interval is further decreased the resulting increase of the overall availability is only minimal. This result suggests that a high scrubbing rate is not beneficial for all applications.

2) Redundancy with On-Demand Reconfiguration:

a) Approach 3 - Duplication with Comparison: If the stream processor is duplicated, a comparator can be used as failure detector. Then, the failure recovery process can be triggered on demand. Since the processor is immediately repaired and does not have to wait for the next recovery cycle, the availability increases. In the following, the failure detector itself is assumed to be fault-free.

While the failure detector is able to detect a failure mode effect, it cannot determine the underlying failure mode. Hence, the most reliable recovery approach is again a partial reconfiguration. And because it is unknown which instance of the redundant processor is faulty, both must be reconfigured, leading to an increased downtime.

b) Approach 4 - Triple Modular Redundancy: In contrast, if the processor is triplicated, a voter can be used for failure detection but also for failure masking, which further increases the availability.

To model this on-demand reconfiguration approach, the counters can be removed from the second Petri net. Now, the places DF_FAILED and DF_OK are used to trigger the reconfiguration process. Due to the deterministic transition df_proc_time, a worst-case failure detection time equal the processing time of one data block is assumed, which contributes to the downtime of a faulty processor instance. The time required to reconfigure the faulty processor also contributes to its downtime. This is modelled by transition mod_reconfig_time in the first Petri net shown in Figure 11, where the model for the triplicated processor is depicted. Now, three tokens are used, one for each redundant instance. The MTBF values of the different failure modes is divided by the number of the tokens in place MOD_OK. For example,

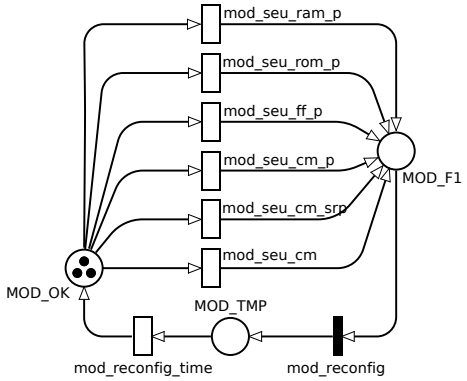

Fig. 11. Redundancy with On-Demand Reconfiguration. Petri net no. 1.

TABLE VII
PROCESSOR AVAILABILITY (APPROACHES 3 AND 4)

Redundancy Mode	Sentinel-3	(SPE)	Galileo	(SPE)
Duplicated	0.999998	0.9991	0.9999994	0.997
Triplicated	0.999999999998	0.9999997	0.9999999999999	0.999996

if three redundant instances are working correctly, the chance that one of them fails is three times higher compared to a single processor.

The results for the duplicated (approach 3) and triplicated processor (approach 4) are listed in Table VII. Compared to the periodic recovery approaches, the availability is in both cases very high. The good performance of the duplicated processor is due to the rather low ratio of the processing time of one data block (50 ms) to the reconfiguration downtime ($2 \cdot 30$ ms) because only a few data blocks are affected by the downtime. For larger data blocks, the situation might be significantly different. The availability for the triplicated processor is so high because a failed processor instance can be repaired in the background without affecting the availability of the overall system. It is only affected if a second instance fails before the first faulty instance is repaired.

VI. ADVANTAGES OF BLOCK RAM PROFILING

As discussed in Section II, the susceptibility of the Block RAM bits cannot be neglected in availability analysis of spaceborne SRAM-based FPGAs due to the large amount of Block RAMs used in some satellite payload data processing applications. In order to substantiate this claim, the reliability of a stream processor, to which no failure recovery method is applied, will be analysed and compared for the following three cases:

- Case (a): The susceptible Block RAM bits F_{RAM}, determined by the Block RAM profiling tool, are taken into account in the availability analysis.
- Case (b): The Block RAMs are fully neglected in the availability analysis.
- Case (c): All Block RAM bits within the stream processor ($F_{RAM/A}$) are assumed to be susceptible in the availability analysis.

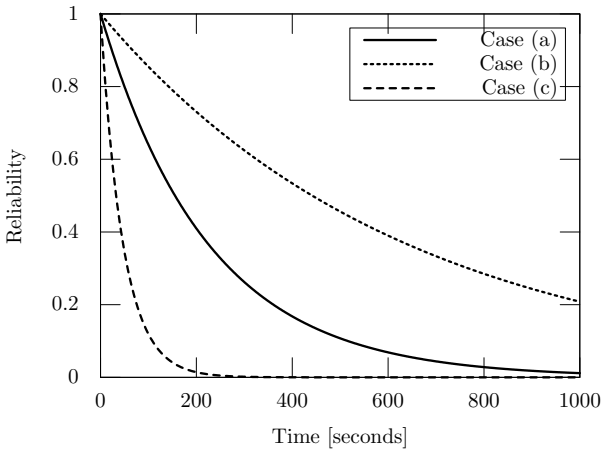

Fig. 12. Impact of Block RAM profiling on reliability results.

For illustration, the mission *Sentinel-3 (SPE)* is used in the ensuing analysis. For cases (a) and (b), the MTBF values are listed in Table IV. For case (c), where all Block RAM bits are assumed to be sensitive, the corresponding MTBF value can be calculated in the same way, but this time all 1,548,288 Block RAM bits must be taken into account, resulting in $\mathrm{MTBF}(F_{RAM/A}) = 50.73$ sec.

Since the system becomes unreliable each time when one of the failure modes occurs, its reliability diagram can be represented by a series connection of blocks corresponding to each failure mode. The reliability of a series system over time can be calculated as follows:

$$\mathrm{R}(t) = e^{-t\sum_{i=1}^{n}\lambda_i} \tag{5}$$

where t is the time and $\lambda_i = {}^1/_{\mathrm{MTBF}}$ the failure rate of mode i.

The reliability over time of the aforementioned three cases is plotted in Figure 12. It can be seen from Figure 12 that the reliability curve corresponding to case (c), where all Block RAM bits are assumed to be sensitive, drops to zero very quickly. The difference of the prediction between case (b), where no Block RAMs are considered and case (c) can be as high as 75%. However, since the Block RAM profiling tool identifies only a fraction of Block RAM bits as sensitive, the curve of the corresponding case (a) is the more realistic estimate. The maximum difference of the prediction between case (a) and the other cases is 37%. From this example, it becomes clear that the impact of the susceptible Block RAM bits cannot be ignored. Without any knowledge about the usage of the Block RAMs, the actual reliability curve could lie anywhere between case (b) and (c).

It has to be noted that the degree to which the total number of the estimated sensitive Block RAM bits eventually affects the estimated availability figure depends very much on the failure recovery approach. As an example, Table VIII shows the availability figures computed for the approach, in which a single stream processor is periodically reconfigured (approach

TABLE VIII
IMPACT OF BLOCK RAM PROFILING ON AVAILABILITY (APPROACH 1)

Reconf. Interval [1]	Case (a)	Case (b)	Case (c)
5,000	0.60	0.83	0.19
500	0.94	0.98	0.77
50	0.994	0.998	0.97
5	0.9993	0.9997	0.997

[1] (images) with the processing time of 1 image being equal to 50 ms

1). If the stream processor is repaired very frequently, say every 5 images, the impact is rather low because at this point of time, the probability that the stream processor is still healthy is high for all three cases. However, if it is repaired less frequently, e.g. every 5,000 images, the impact becomes much more visible as evidenced by the top row in Table VIII.

Recently, the availability analysis method was validated during a proton irradiation test campaign [26]. By means of the proposed fault injection and Block RAM profiling technique, the number of sensitive memory elements could be predicted with such a precision that the estimated availability figure differed from the measured one during the test by only 0.9%.

VII. CONCLUSIONS

In this paper, a novel methodology for a systematic availability analysis of stream processors, implemented on SRAM-based FPGAs, has been presented. The proposed methodology provides a new capability allowing the detailed analysis of a circuit without a detailed knowledge of its internal structure. The methodology is implemented as a set of tools integrated together and is fully tested using a case study based on a typical satellite payload data streaming system. A new configuration memory failure classification algorithm is proposed and a novel Block RAM profiling tool is developed and incorporated allowing Block RAM failure quantification. The proposed approach to availability analysis could easily be automated and is non-intrusive, i.e. it does not require any circuit modifications. Hence, the method can serve as an essential tool during the design and qualification phase of space electronics systems that incorporate SRAM-based FPGAs, simplifying the evaluation of the appropriate FDIR mechanisms. This is the first implementation of an ESA standard availability aproach with regards to SRAM-based FPGAs that addresses a gap in the current space qualification process.

ACKNOWLEDGEMENT

Sponsorship from ESA under the NPI Programme, Airbus Defence and Space, UK and the University of Leicester is gratefully acknowledged. We wish to thank the reviewers for their constructive comments and suggestions for improvement.

REFERENCES

[1] C. Norton, T. Werne, P. Pingree, and S. Geier, "An evaluation of the Xilinx Virtex-4 FPGA for on-board processing in an advanced imaging system," in *2009 IEEE Aerospace conference*, 2009, pp. 1–9.

[2] Andrew Holmes-Siedle and Len Adams, *Handbook of Radiation Effects*. Oxford University Press, 1993.

[3] F. Siegle, T. Vladimirova, O. Emam, and J. Ilstad, "New voter design enabling hot redundancy for asynchronous network nodes," in *9th NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2013.

[4] ——, "FDIR techniques for payload streaming applications using SpaceWire-based networks," in *International SpaceWire Conference*, 2014.

[5] ECSS, "Space engineering: Space environment. ECSS-Q-ST-30-09C," ESA/ESTEC, Tech. Rep., 2008.

[6] D. McMurtrey, K. S. Morgan, B. Pratt, and M. J. Wirthlin, "Estimating TMR reliability on FPGAs using Markov models," 2008. [Online]. Available: http://contentdm.lib.byu.edu/cdm/ref/collection/IR/id/612

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

[8] J. Kastil, M. Straka, L. Miculka, and Z. Kotasek, "Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into FPGA," in *15th Euromicro Conference on Digital System Design (DSD)*, 2012, pp. 250–257.

[9] Q. Martin and A. George, "Scrubbing optimization via availability prediction (SOAP) for reconfigurable space computing," in *IEEE Conference on High Performance Extreme Computing (HPEC)*, 2012, pp. 1–6.

[10] K. Hoque, O. Mohamed, Y. Savaria, and C. Thibeault, "Early analysis of soft error effects for aerospace applications using probabilistic model checking," in *Formal Techniques for Safety-Critical Systems*. Springer International Publishing, 2014, vol. 419, pp. 54–70.

[11] ——, "Probabilistic model checking based DAL analysis to optimize a combined TMR-blind-scrubbing mitigation technique for FPGA-based aerospace applications," in *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Oct 2014, pp. 175–184.

[12] F. Siegle, T. Vladimirova, O. Emam, and J. Ilstad, "Adaptive FDIR framework for payload data processing systems using reconfigurable FPGAs," in *8th NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2013, pp. 15–22.

[13] G. Allen, G. Swift, and C. Carmichael, "Virtex-4QV static SEU characterization summary," NASA Jet Propulsion Laboratory (JPL), Xilinx, Tech. Rep., 2008.

[14] ECSS, "Space engineering: Space environment. ECSS-E-ST-10-04C," ESA/ESTEC, Tech. Rep., 2008.

[15] Carl Carmichael and Chen Wei Tseng, "Correcting single-event upsets in Virtex-4 FPGA configuration memory. XAPP1088 (v1.0)," Xilinx, Tech. Rep., 2009.

[16] N. Rollins, M. Fuller, and M. Wirthlin, "A comparison of fault-tolerant memories in SRAM-based FPGAs," in *2010 IEEE Aerospace Conference*, 2010, pp. 1–12.

[17] M. Krepa, "JPEG Encoder," 2013. [Online]. Available: www.opencores.org/project,mkjpeg

[18] B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski, "SoCWire: a network-on-chip approach for reconfigurable system-on-chip designs in space applications," in *3rd NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008, pp. 51–56.

[19] ECSS, "SpaceWire - links, nodes, routers and networks. ECSS-E-ST-50-12C," ESA/ESTEC, Tech. Rep., 2008.

[20] CCSDS, "Space Packet Protocol. Blue Book. CCSDS 133.0-B-1," Consultative Committee for Space Data Systems, Tech. Rep., 2003.

[21] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, June 2006.

[22] Cobham Gaisler AB, "LEON3 Processor," http://www.gaisler.com/index.php/products/processors/leon3, May 2015.

[23] A. Agresti and B. A. Coull, "Approximate is better than 'exact' for interval estimation of binomial proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998.

[24] J. Pellish, M. Xapsos, C. Stauffer, T. Jordan, A. Sanders, R. Ladbury, T. Oldham, P. Marshall, D. Heidel, and K. Rodbell, "Impact of spacecraft shielding on direct ionization soft error rates for sub-130 nm technologies," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3183–3189, Dec 2010.

[25] A. Zimmermann, "Modeling and evaluation of stochastic Petri nets with TimeNET 4.1," in *6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2012, pp. 54–63.

[26] F. Siegle, T. Vladimirova, C. Poivey, and O. Emam, "Validation of FDIR strategy for spaceborne SRAM-based FPGAs using proton radiation testing," in *Conference on Radiation Effects on Components and Systems (RADECS)*, 2015.