Improving Reconfigurable Systems Reliability by Combining Periodical Test and Redundancy Techniques: A Case Study

Eduardo Augusto Bezerra [1], Fabian Vargas [2] and Michael Paul Gough [3]

eduardob@acm.org, vargas@computer.org, M.P.Gough@sussex.ac.uk

[1,3] Space Science Centre, School of Engineering and Information Technology, University of Sussex, UK
[1,2] Informatics Faculty/Electrical Engineering Dept., Catholic University – PUCRS, Brazil

Abstract. This paper revises and introduces to the field of reconfigurable computer systems, some traditional techniques used in the fields of fault-tolerance and testing of digital circuits. The target area is that of on-board spacecraft electronics, as this class of application is a good candidate for the use of reconfigurable computing technology. Fault tolerant strategies are used in order for the system to adapt itself to the severe conditions found in space. In addition, the paper describes some problems and possible solutions for the use of reconfigurable components, based on programmable logic, in space applications.
Key words: Reconfigurable computing, Fault Tolerance, Test, Programmable-logic-array, and Space Applications

1. Introduction

The on-board computer system of a spacecraft for long-life missions is a good representative example of an adaptive system [1,2]. This category of system, including software and hardware, has to be designed considering that post-launch maintenance is generally impractical [2], and the hardness of the space environment will certainly cause problems to the system. Traditional on-board computers based on microprocessors, adapt to the problems they face by using fault-tolerance techniques. Faults are tolerated, and vital services such as, for instance, communication to ground stations, are maintained operative. From the hardware point of view, the adaptation takes place by the isolation of faulty modules from the system, and, where it is possible, by the activation of spare modules. At the software level, the adaptation is identified, basically, when the microprocessor starts using a new program, stored on-board or up-loaded from ground.

Reconfigurable devices introduce to the hardware level, the flexibility for adaptation provided at the software level [3,4]. In the case of space applications, two important advantages of having the whole computer system, including the hardware and the software parts, implemented using only hardware components (reconfigurable hardware) are: the economy in area usage and board complexity; and the increased processing speed with lower clock rates.

On-board systems based on reconfigurable hardware will also reduce the overall design cost associated with this category of computer. Because of the application specific nature of this kind of system, its requirements can vary significantly from application to application, resulting in a completely new design for almost every new application. Reconfigurable devices are appropriate for the implementation of application specific solutions. This allows the designers to have different hardware configurations, without the need for changes in the board layout. It is another example of adaptation. In this case the hardware adapts to a new application not because of the occurrence of a fault, but because of a new mission requirement.

The drawback to the use of reconfigurable devices is the difficulty associated with the "software" development for this kind of hardware. The internal organisation of the reconfigurable devices available, makes the development of good synthesis tools for high level programming languages an arduous job. Even when using hardware description languages as, for instance, VHDL, the developer of spacecraft computers has to keep the code as simple as possible, in order to avoid problems for the synthesis tool [5]. For instance, in systems that require complex data structures or many levels of nested loops, the best solution maybe to still use conventional microprocessor based boards.

The first concern of a designer of a reconfigurable system for long-life space applications is the improvement of system dependability features such as, for example, reliability, availability, maintainability and testability [6]. Reliable devices together with fault-tolerance and test strategies are used in order to accomplish this target. In the past few years, strategies to improve the dependability features of reconfigurable computer systems, have been proposed and implemented [1,7-12]. These strategies

are mainly based on the traditional ones used in microprocessor based systems.

Following a similar approach, in this paper strategies to improve the dependability of reconfigurable computing systems for space applications are presented. An important consideration is to define the strategies that make use of the fact that reconfigurable computing permits new possibilities, for instance, the use of a combination of strategies for fault tolerance in software and in hardware at the same level of abstraction. The ideas discussed here can be used not only for space applications, but also for any other embedded system with similar dependability requirements. It is important to highlight that all strategies discussed in this work are to be applied during the system's usage and at a high level of abstraction. The fault detection activities take place first at component level and then at the network node level. The diagnosis (location) of faults inside a component is an interesting activity from the manufacturers' point of view and it is out of the scope of this work. This position can be explained because there is a huge effort from the user point of view to isolate faulty parts internally to a component. Instead of this, it is more interesting for the user to try to recover the whole component, which is possible if the application is based on SRAM-type FPGA components.

This paper is organised as follows. Section 2 describes a case study which is one of the motivations for this work. Section 3 describes the basic node of a network architecture for space applications, based on reconfigurable devices. In Section 4 there is a description of the fault model adopted, and a brief review of Single Event Upset (SEU), which is one of the main sources of problems in on-board electronics in space applications. Section 5 introduces strategies for dependability improvement, with emphasis on reliability, availability and testability improvement techniques. Section 6 describes an approach for masking connectivity faults. Section 7 presents implementation details. Section 8 discusses expected results. In Section 9 there are some conclusions, and future directions.

2. Motivation for the Use of FPGAs in Space Applications

The main motivation for using FPGAs instead of microprocessors for on-board computer implementation, is the gain in performance with associated decrease in the PCB area usage. In order to find out the feasibility of using FPGAs in this class of application, a case study was developed. The case study is the FPGA implementation of an on-board instrumentation module of a NASA sounding rocket [13]. This rocket flew from Spitzbergen, Norway, in the winter of 1997/1998, carrying a scientific application computer designed at the Space Science Centre, University of Sussex [14]. The real time science measurement performed by this embedded system is an auto-correlation function (ACF) [15] processing of particle count pulses as a means of studying processes occurring in near Earth plasmas. The original module as flown consisted of a board with two DS87C520 microcontrollers (8051 family), FIFOs, state machines and software written in assembly language. Although this ACF implementation is a very specific application, the system as a whole, including the hardware and the software parts, is not too different from conventional embedded systems based on microprocessors or microcontrollers. This case study is a typical memory transfer application, with a high input sampling rate and with scarceness of processing modules. The most demanding actions for processing blocks, are the ones with multiply-and-accumulate operations (MACs), typical of DSP applications.

The test strategies proposed in this paper are designed to execute in parallel with the user application and with the fault-tolerance strategies. When considering processing time, there are no performance penalties, because there is no need, for instance, to time share tasks. Table 1 lists the number of cycles necessary to run the processes written in assembly language for the microcontrollers, and the equivalent ones written in VHDL for the FPGA. The software algorithm flown on the example mission was broken down into six contributing processes to ease the comparison.

Another important system feature improved because of the use of configurable computing technology is the reduction in the number of hardware components. The main system repercussions are the reduction in the on-board area usage, and in the power consumption. The reduction in the number of components can be achieved by using only one FPGA device configured to execute the same functionality as the whole 8051 based system. For instance, external chip FIFOs were replaced by circular FIFOs implemented using data structures in VHDL.

Considering the use of the FPGA board in an application where fault-tolerance is not a requirement, as the original case study, all of the electronics board can be implemented as a single chip, for instance, an XQ4085XL Xilinx FPGA [16,13] with two XQ1701L serial ROMs to store the bitstream. The result is a reduction from 22 to 3 chips. Extrapolating the case study to a long-life system, where fault-tolerance is required, we can use a board similar to the one shown

in Fig. 1.b. In this case the reduction in the number of components may be from 22 to about 10 chips on-board, which is still a significant result, considering that now the system has fault-tolerant capabilities.

Table 1. Performance Comparison for the Case Study.

	µcontroller (cycles)	*FPGA (cycles)*	*Rate*
P1	*4,518*	*1*	*4,518 times faster*
P2	*8..36*	*1*	*8 to 36 times faster*
P3	*18..1,018*	*1..68*	*18 to 14.97 times faster*
P4	*1,240*	*48*	*25.8 times faster*
P5	*1,334..3,438*	*132..143*	*10.11 to 24 times faster*
P6	*11,116*	*288*	*38.6 times faster*

3. System Description Overview

Field-Programmable Gate Arrays (FPGAs) are the devices used for the implementation of reconfigurable computer systems. The block diagram in Fig. 1.a shows the proposed architecture for an on-board processing system, based on FPGAs, which may be used for both scientific and commercial space applications. The two main improvements in this architecture over the architectures that have been used in most space applications previously, are the use of reconfigurable devices as the main processing elements, and the use of a network to connect the different modules. As the main objective of this paper is to discuss and to present strategies for the dependability improvement of processing elements of a reconfigurable system, we describe in this Section a basic network node.

The block diagram of the network node shown in Fig. 1.b has two external communication channels, one for connection to the network, and the other one for interfacing to the application. The first one has a fixed number of signals, as the protocol for inter-node communication is pre-defined. A microcontroller with an embedded UART is a good option for the inter-node communication. The second one is defined according to the application requirements, and is a good example of the flexibility introduced by the reconfigurable computer technology.

The main responsibilities of the processing element, FPGA module in Fig. 1.b, are to construct telemetry packets, according to the European Space Agency (ESA) standards [17], and to implement the protocol for communication on the on-board network system. The protocol used is the Controller Area Network (CAN), and the HurriCANe core, designed by ESA has been used in the design [18].

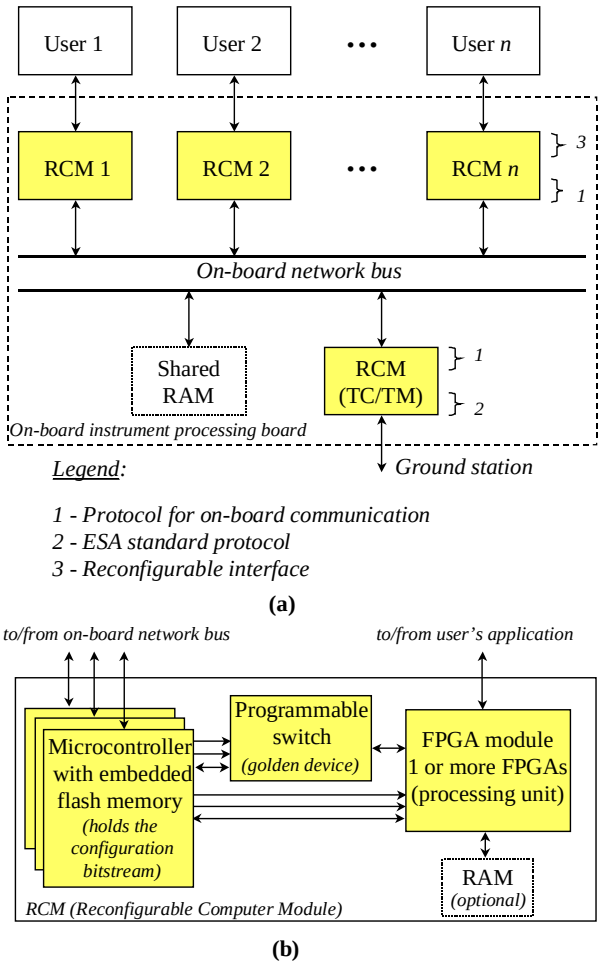

Fig. 1. Block diagram of the proposed system. (a) Network architecture. (b) Basic RCM node.

In some applications it may be possible to transfer some or all user processing activities to the FPGA module. An example is when the on-board data-handling (OBDH) system and one (or more) of the applications are designed by the same group [19, 20]. In this case the "User" block shown in Fig. 1.a may be very simple, for instance, consisting of only analog devices, sensors and converters, as the FPGA module in Fig. 1.b may be used to execute all of the processing.

The microcontroller is responsible for the implementation of the physical layer of the CAN protocol, connecting the CAN core (located on the FPGA module) to the network bus. Another important activity performed by the microcontroller is the management of the reconfiguration and test of the FPGA module. The design and implementation of

these activities are the main objective of this work, and they are detailed in Section 5.

Other components of the Reconfigurable Computer Module (RCM) node are a RAM memory and a programmable switch. The RAM module may be attached to the FPGA module, in order to be used by some applications as a scratch area. The programmable switch is one of the single points of failure of the system (the other one is the FPGA itself), and for this reason a highly reliable and non complex device is used.

The data pins of the microcontrollers are connected directly to a voter implemented in the FPGA, and the pins used to program the FPGA are connected to the programmable switch. If a single discrepancy is detected by the voter, then the FPGA reprograms the switch in order to select one of the two healthy microcontrollers as the new configuration manager.

The microcontrollers' embedded flash memories hold the configuration bitstreams (CBs) for the FPGA module. They may be changed from the ground in case of system upgrades or bug fixes. The microcontroller works as a configuration manager allowing device(s) of the FPGA module to be initialised from the flash memories as if they were serial ROMs. The microcontroller is also responsible for refreshing the flash memories when they experience upsets. As the objective of this work is the improvement of the dependability features of the FPGA module, strategies for dependability improvement of the microcontroller and flash memories will be discussed in a future work.

4. Fault Model Adopted

As the systems based on the proposed architecture have been conceived for long-life missions, all of the electronic components must be military standard. However, as stated before, because of the hostile environment found in space and the long-life expected for the system, additional fault-tolerance strategies are used in the design, even when employing high reliability devices. In order to define the strategies, a very simple, but efficient, fault model was chosen. The faults considered in this fault model are "stuck-at" and "connectivity" (faults in interconnect resources) [6].

The main cause of stuck-at faults in space are the Single-Event Upsets (SEUs) caused by atmospheric high-energy neutrons [21-23]. Most of the FPGAs available are SRAM based, and devices implemented with this kind of technology are sensitive to SEUs. Basically, a SEU takes place when a single high-energy particle (typically a heavy ion) strikes a sensitive node in a memory cell, which causes the particle to loose energy via production of electron-hole pairs, resulting in a densely ionised track in the local region of that element. It will force the affected memory cell to stay in a fixed state, 0 or 1, and, consequently, stuck-at faults are the best option to model the bit errors that can occur.

The other modelled fault, connectivity, is responsible for most of the problems in a board [6] and, as described later, special strategies as, for instance, bus replication and voters, are used to tolerate this problem. Connectivity faults are related to I/O and connection resources in general. Some authors defend the position that tests in the processing elements cover also the connectivity faults. The strategies used to prevent and, when it is not possible, to tolerate the faults, belong to the fault model adopted, are described next in Section 5.

In [21,22] there is a study showing the low SEU susceptibility of Xilinx FPGAs. However, for some critical applications where human life is at a premium or when the whole on-board electronics is dependent on two or three core components, then, even the low SEU susceptibility must be improved. The strategies described in the next sections are used to tolerate the effects of faults resulting of SEU occurrences. Needless to say that in order to tolerate a fault, it is necessary first to detect the fault. Test strategies for fault detection are also described next.

5. SEU Prevention Strategies

5.1. Refresh operation in a Triple Modular Redundancy (TMR) FPGA system

In [22] a strategy to reduce the effects of SEUs in Xilinx FPGAs was proposed. Basically, as shown in Fig. 2, three FPGAs are configured with the same bitstream (triple redundancy), and operate in synchronism. A controller reads the three FPGA bitstreams, bit after bit, and if there are no differences, then a correct functioning with no SEU occurrence is assumed. This procedure is executed continuously, with no interference in the FPGA normal operation. Such a scheme is possible because of the FPGA's readback feature.

If one input of the controller is different from the others, then it is assumed that an SEU has occurred, and a reconfiguration of the faulty FPGA is executed. In [21] it was shown that a simple refresh operation, in this case by means of reconfiguration, is sufficient to recover the device from an SEU. The main problem with the refresh recovery is the total loss of measurement data within the instrument system, since all those FPGAs have to be resynchronised to the same input data and same position in the application

algorithm. Another problem is the time necessary for reconfiguration, and depending on the application size, it is therefore recommended to divide the system into small blocks using several small FPGAs. This is because in a small FPGA, configuration can be made in just a fraction of second (e.g. 195 ms, for Xilinx XQ4085XL). The block size has to be calculated according to the application time requirements.

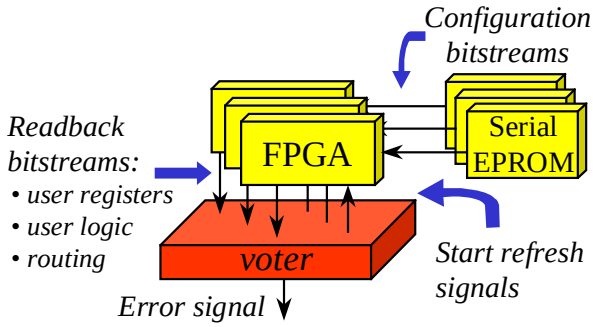

Fig. 2. A TMR FPGA system.

It is important to note that we are not proposing the use of refreshing operations as a means to prevent SEUs. This strategy has already been proposed in previous works [21][22]. What we are proposing are strategies to trigger and to start the refresh operation.
In the next three Sections, the methods proposed in this work for SEU prevention are presented. These new methods are also based on the refresh execution, but without FPGA replication.

5.2. Periodic Refresh Without FPGA Replication

As the target system is a long-life application, periods of downtime are considered in its design, and thus are possible to be interrupted and completely reinitialised after some time running, with no major problems. This strategy, shown in Fig. 3, utilises a clock generator and a counter. In the event of a rising edge pulse, generated by the clock, the counter is incremented. Every time the counter reaches the zero value the refresh operation is executed. Refresh is achieved by the counter process resetting the FPGA PROG pin, which leads to the FPGA being reconfigured, preventing SEU occurrences from affecting the system functioning. It is important to notice that in this strategy there is no test execution, and consequently, no SEU detection. The refresh is executed periodically, even if there are no SEU occurrences. In terms of hardware resources, this strategy is less expensive than the TMR one, but depending on the application, it can be expensive in terms of system availability.
The appropriate refreshing frequency has to be calculated according to the application characteristics. For example, in a hypothetical application where a new processing cycle starts every 20 hours, a 15 Hz clock generator and a 19 bits counter could be used to trigger the refresh operation, which will happen every 19.4 hours.

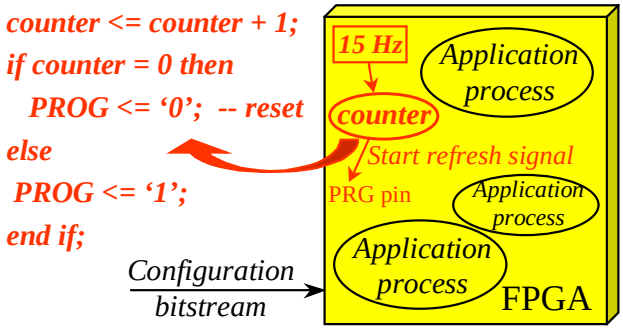

Fig. 3. Using a counter to start the refresh operation.

5.3. Signature Analysis-Driven Refresh Without FPGA Replication

Another option for SEU prevention is the use of a signature analysis method [6], to identify when a refresh operation is necessary. For applications where periods of downtime and loss of data are not allowable, this strategy is more efficient than the clock/counter one. In this work two different architectures have been implemented. In the first one it is assumed that the FPGA module (Fig. 1.b) has two FPGAs, one for processing and a smaller one for testing activities. In the second architecture the FPGA module has only one FPGA, and the testing activities are performed by a microcontroller. From the system reliability point of view, both architectures have advantages and disadvantages, which are discussed in Section 9. A short explanation of the implementation of signature analysis methods in both architectures is given next.

FPGA Module has Two FPGAs
Fig. 4 shows a block diagram of the first version of the FPGA module of Fig. 1.b. In this version the processes responsible for processing the user's application and for starting the readback and refresh operations are located on FPGA A. FPGA B, the smaller one, is responsible for the testing activities.
As FPGA area is an expensive resource, the method proposed here uses the LFSR/PSG approach for signature generation and analysis [24]. A Linear Feedback Shift Register (LFSR) is a shift register with combinational feedback logic around it that, when clocked, generates a sequence of pseudo-random patterns [24]. In our case, we are considering the use of a primitive polynomial, in order to generate all the

$2^n - 1$ possible combinations, where **n** is the degree of the polynomial. A Parallel Signature Generator (PSG) is an LFSR with exclusive-or gates between the shift registers, implementing a generator polynomial used to compact a given sequence of bits. Using this approach, the same piece of hardware is used both, to trigger the start of the readback (LFSR mode) and to generate and analyse the FPGA bitstream signature (PSG mode).

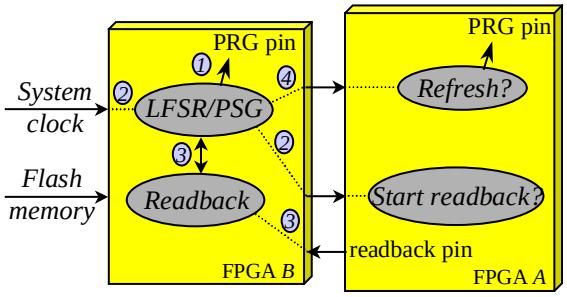

Fig. 4. The LFSR/PSG approach.

In order to define when the readback starts, first the LFSR/PSG process (Fig. 4, ①), working as a simple LFSR, generates all $2^n - 1$ pseudo-random patterns. When the output of the LFSR has a pattern matching a pre-defined seed, then the operation mode is changed to readback. At this moment a signal to start the readback is sent to FPGA A (Fig. 4, ②).
After each 8 cycles of the system clock, a register in the LFSR/PSG process is loaded with the contents of a 8 bits shift-register located in the Readback process (Fig. 4, ③) which holds the last 8 bits received from FPGA A. This shift-register is controlled by the Readback process, which is also responsible for filtering the bits "*unusable data*", "*RAM bits*" and "*capture bits*", not used for purposes of signature generation [25]. The reason for that is because these bits change dynamically during the FPGA utilisation and are not suitable for comparison with the "gold" signature. The "gold" signature is generated on ground, using the same PSG method, from the original bitstream used to configure FPGA A, and is stored on-board, in FPGA B.
When the readback is concluded, the LFSR/PSG process in FPGA B uses the calculated signature to compare to the on-chip stored, pre-defined one. If the test fails, then a "start refresh" signal is sent to FPGA A (Fig. 4, ④) in order to "clean" possible SEUs. The FPGA B refreshes itself after the end of all FPGA A readback/refresh executions.

FPGA Module has One FPGA
The main difference between this approach and the latter is that the testing activities are now implemented in software and they run on a microcontroller. This approach could be implemented in the architecture shown in Fig. 1.b, but a simpler one with only one microcontroller and one FPGA has been used here because the FPGA board with three microcontrollers and a programmable switch is still in the design stage.
Using a microcontroller for the readback and refresh operations is much easier than using another FPGA. The microcontroller already has fixed hardware blocks that can be used to build the test strategy, with no need for hardware reuse. A Cyclic Redundancy Check (CRC) based strategy has been implemented instead of the LFSR/PSG method. A microcontroller's timer is used to trigger the readback, and a standard CRC algorithm is used to calculate the signature of the bitstream. On the FPGA side, the operation is the same as the one described in the last Section.

5.4. Signature Analysis With Continuous Readback Execution

In both signature analysis methods described in the last Section, the test for SEU occurrences is executed periodically. Another option for the test is to execute the readback continuously, as it does not affect the normal FPGA operation. When considering the use of two FPGAs, this method is less expensive in terms of hardware, as part of the LFSR/PSG process is not necessary. As the readback is executed continuously, then there is no need for sending a signal to the "start readback" process resident on FPGA A.
It should be noted that a drawback of this technique is the increase of power consumption since the "always on" readback process resident in FPGA B (or microcontroller) and the continuous readback task that is performed by FPGA A increase the overall system power consumption due to the increased switching activity.

6. Masking Connectivity Faults

The SEU prevention strategies described in the last Section, are very efficient for fault prevention in processing modules, as operation units are implemented using the FPGA SRAM based look-up tables (LUTs). Control units of processing modules, are partially implemented using flip-flops. They are one of the points not covered by this work, since many well known fault-tolerant strategies to improve control (and data) flow reliability can be found in the literature [6].
Reliability improvement in the processing modules is worthless if the input data correctness is not guaranteed. The proposed strategy is shown in the block diagram in Fig. 5. In this scheme a majority voter receives the same data from three different

FPGA input pins, and if at least two of them are equal, then the data are sent to the application, otherwise, an error signal is set, invalidating the data. The block diagram was partially generated by *Synplify* from a VHDL code.

The strategy is used to mask faults in the external FPGA pins, and in the internal FPGA routing resources. It is assumed that the same sensor output is connected to three different FPGA pins, sending the same data to the voter. Using three different sensors, which characterises a triple modular redundant (TMR) implementation [6], may be possible but it will depend on the data being collected. In most of the cases, different sensors send different data to the voter and, even if the data are correct, it may result in a wrong interpretation by the voter. This happens because different sensors can detect different physical phenomena, at the same instant of time.

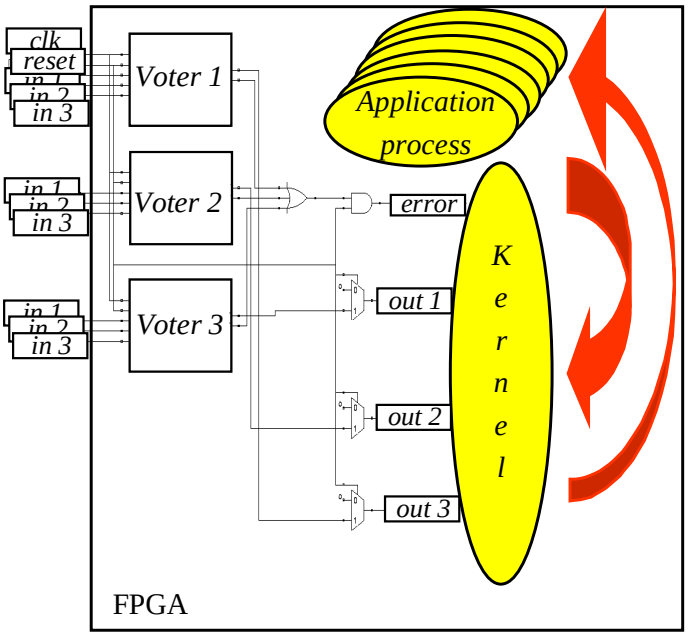

Fig. 5. Using replicated inputs/voter to mask connectivity faults.

For this fault masking strategy to be efficient, the three input signals for the voters must be located, preferably, in three distant pins. For instance, in the block diagram in Fig. 5, the input *"in 1"* may be located in the pin 40, whilst the input *"in 2"* is located in pin 80. The pin locations are chosen by the designer, using a constraints file, before the placement and routing (PAR) execution. The netlist generated by a synthesis tool, from a VHDL source code, has no pin location and routing information, and this netlist is used as input to the PAR tool. In some cases it may be necessary to edit the CB generated by the PAR tool, and change, manually, the position of the components of a voter, in order to approximate them to the input pins. The pins' location and the delay for individual routes can be specified in a constraints file. Moreover, the PAR tool may place a voter very close to a pin, but very distant from another one, having both time delays according to that defined in the constraints file, but with very different times between them. A solution to avoid the need for manual intervention, is to define very short delays in the constraints file. The problem with this solution is that, depending on the design complexity, and the size of the FPGA chosen, the constraints specified may not be achievable. This strategy for connectivity faults masking can be employed in the RCM node shown in Fig. 1.b, because the voters are implemented in the same FPGA along with the application, as shown in Fig. 5. This strategy masks permanent, transient or intermittent faults efficiently.

7. Prototyping Environment and Implementation Details

Prototypes of the proposed strategies (Sections 5.2, 5.3, 5.4 and 6) have been implemented using two FPGA developing boards, one with a Xilinx XC4010XL and another one with a Xilinx Virtex XCV50. The board with the Virtex FPGA was used in all implementations. The XC4010XL FPGA was used as the tester described in Section 5.3 ("FPGA module has two FPGAs").

The Virtex board has a PIC microcontroller, which is used for configuring and testing the FPGA. The bitstream for configuring the FPGA is stored in the PIC's embedded flash memory. The strategies described in Section 5.3 have not been fully tested because of the absence of a board with three flash memories and a programmable switch. However, all the procedures described, except for the flash memory selection, have been verified and proven to be functioning in a satisfactory way.

A vacuum chamber could have been used at the Space Science Centre in the testing activities, but there were no spare FPGA boards available when this paper was written. The solution adopted was a deterministic fault injection by using Xilinx development tools. For instance, the flow used for the fault injection and system verification of Section 5.3 ("FPGA module has one FPGA") is as follows:

1. Generation of the CB and respective signature;
2. Download the CB into the PIC's flash memory;
3. Load PIC with the test program and the signature;
4. PIC configures the FPGA and starts the testing activity;
5. Change the state of a LUT's bit using Xilinx Floor Planner Editor in the host computer;
6. Download the faulty CB into the FPGA using a

JTAG cable (PIC holds the signature of the healthy CB);

7. Next time PIC calculates the CRC, the injected fault is detected and the healthy CB located on the PIC's flash memory is used to refresh the FPGA.

Deterministic fault injection has limitations in comparison to a random strategy using a vacuum chamber and radioactive material, but it is sufficient to show the efficiency of the proposed strategies considering the fault model adopted.

8. Numerical Analysis of the Signature Analysis Method

The RCM node shown in Fig. 1.b has been analysed in numerical terms using reliability evaluation techniques [6]. For this analysis two situations are considered.

In the first situation, the three flash memories hold three different CBs. This scenario represents a real reconfigurable computing system, because the FPGA functionality can be altered, on-the-fly, according to the application requirements. From the fault-tolerance point of view it is not a good approach as, in the case of an SEU occurrence in one of the flash memories, the respective application has to stop, and wait for a good CB be up-loaded from the ground station. The reliability of this situation is found from Equation 1.

$$R_1(t) = 1 - (1 - R_{flash}(t)) \qquad \text{Equation 1.}$$

Since the reliabilities of all components, but the flash memories, are constant, they have not been included in the numerical analysis. The reliabilities of the flash memories are not constant, because their contents may be changed when a new CB is uploaded.

In the second situation, the three flash memories hold the same CB, which characterises a TMR system. The vote is executed, implicitly, by the microcontroller or by FPGA B, using the signature analysis method described in Sections 5.3 and 5.4. As this test strategy is not capable of fault location, then, in case of a fault detection, it is not possible to identify whether the problem was in the flash memory or in the tester. In any case, the FPGA A (Fig. 4) is reconfigured with a CB from another flash memory. If the error persists, then the diagnosis is a permanent fault in FPGA A, and the module has to be by-passed. On the other hand, if with the new CB no error is detected, then the respective flash memory is considered faulty, and then it needs to be refreshed in order to try to clear any occurrence of SEUs. The reliability of this situation can be found from Equation 2.

$$R_2(t) = 1 - ((1 - R_{flash1}(t))* (1 - R_{flash2}(t))* (1 - R_{flash3}(t))) \qquad \text{Equation 2.}$$

To demonstrate the reliability improvements when using replicated CBs (R_2), we consider a hypothetical situation where the failure rate (λ) is identical for each flash memory. For this study a failure rate of 0.0001/hour was chosen, to allow for the generation of quantitative information for comparison purposes. Considering that $R(t) = e^{-\lambda t}$ and $\lambda_{flash1} = \lambda_{flash2} = \lambda_{flash3} = \lambda_{flash}$ then Equations 1 and 2 can be re-written as the following Equations.

$$R_1(t) = e^{-\lambda t} \qquad \text{Equation 3.}$$

$$R_2(t) = e^{-3\lambda t} - 3e^{-2\lambda t} + 3e^{-\lambda t} \qquad \text{Equation 4.}$$

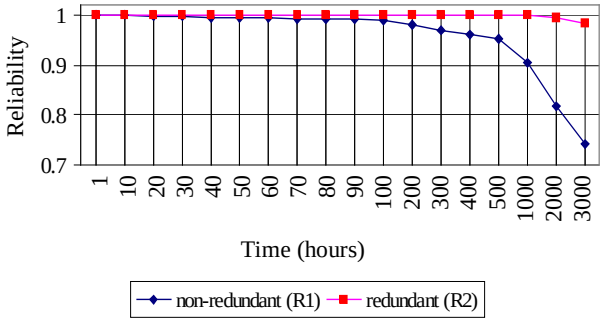

Fig. 6. The reliability responses for the two situations.

The graph in Fig. 6 was plotted from Equations 3 and 4. From this graph it is possible to observe that the reliabilities of each case remain almost the same value at the end of 30 hours of work, and in an acceptable range until the end of 100 hours of work. However, the reliability differences between each architecture become more distinctive as the time progress. For example, at the end of 3000 hours of operation (125 days), the reliability for the redundant case (R_2) is 1.3 times better than the non-redundant one (R_1).

9. Final Considerations

The design of reconfigurable computer systems, for space applications, using SRAM-based FPGAs, depends not only on high reliability military devices being available commercially, but also on the definition of strategies for fault tolerance and on-board testing.

This work introduced the use of Built-In Self Test (BIST) techniques and traditional fault-tolerance strategies together with reconfigurable computing technology, in order to improve some dependability features of on-board computers used in space applications.

The proposed strategies are described in Sections 5.2, 5.3, 5.4 and 6. The strategies adopted for our particular implementation are described in Sections 5.3 and 6, FPGA module with only one FPGA. To implement the signature analysis methods, an external tester is necessary. From the reliability point of view, an advantage of using an FPGA as a tester is that the whole system, including the testing algorithms and the user's application could be implemented at the same level of abstraction and using the same description language. This facilitates the use of formal verification methods, improving the system reliability at the design stage. On the other hand, the tester FPGA is an extra single point of failure, and for this reason the second strategy described in Section 5.3 can be considered, in our case, as the one providing more reliability to the system.

Another reason for having selected that strategy is the savings in the PCB area usage and power consumption, which are extremely important concerns when designing on-board computers for space applications. In some applications where power consumption is not crucial, the strategy described in Section 5.4 provides higher levels of reliability.

Performance figures as, for instance, FPGA area usage are not presented in this paper because of the rapid advances in the FPGA industry both at the hardware (FPGA internal organisation) and software (synthesis tools) levels.

A comparison between the proposed strategy and previous works in not straightforward. Most of the strategies for testing FPGAs found in the literature, target the manufacturer's side, where the objective is to locate the fault inside the FPGA. From the user's point of view this information is irrelevant. The few works targeting the user's side have considerable differences to the proposed one, for instance, in the low overhead approach [27]. In that approach, the FPGA is partitioned in tiles, and each tile has logic blocks used as spares. It is an expensive approach in terms of FPGA area usage, but it is more effective than our proposed strategy in case of permanent faults.

The strategies described in this paper deserve a deeper investigation, in order to be used in the design of an adaptive on-board instrument processing system, entirely based on reconfigurable technology. During the case study implementation, a series of problems related to the development of FPGA based systems arose. For instance, the synthesis tools available for high-level languages (e.g. VHDL behavioural and Verilog) are still not efficient, and a VHDL developer has to follow strict rules to obtain good results [5,26]. An FPGA configuration bitstream generated from a high-level language is space consuming, and represents a lower performer circuit when compared to one generated from schematic diagrams or low level languages such as VHDL structural.

Another concern is the time necessary for Electronic Design Automation (EDA) tools to generate CBs. In time critical systems, such as space applications, effective development facilities are important because of the short time available for making remedial changes to a faulty application. In the past several missions were saved as a result of the rapid problem identification, followed by the development of a solution, ground tests and timely transmission of the new software to the spacecraft computer. In addition to the selection of efficient EDA tools, another investigation to be done is related to the hardware description language subject.

The original design had only FPGAs (see FPGA module on Fig. 4), but the programming effort in VHDL necessary to implement the physical layer of the network protocol, and the FPGAs reconfiguration management was too high. A microcontroller has been added to the original design of the RCM node, in order to facilitate the software development. At this point it is important to make clear that it is not a hardware/software co-design project. The two parts of the system are completely independent, with a very well defined interface. The software part (microcontroller) is developed and tested considering the existence of two external objects. On one side there is the object network bus used by the microcontroller to transmit and receive bytes to/from the network. On the other side there is the object FPGA module, which receives/transmits bytes from/to the microcontroller and also it is tested and refreshed according to the strategies defined in this work. The hardware part, FPGA module, has the microcontroller as an external object used to feed the module with bytes from the network and to send bytes to the network. The tests and reconfigurations performed are completely transparent to the FPGA module.

From the fault tolerance point of view, there are several single points of failure at the node level in the proposed system. However, fault tolerance strategies at the network level can be used to identify and to isolate faulty nodes, keeping the system working. These strategies are the subject of a future work.

Acknowledgements

This research was partly supported by CNPq – Brazilian Council for the Development of Science and Technology, and PUCRS – Pontific Catholic University of Rio Grande do Sul, Brazil.

References

[1] A. Fukunaga et al., Evolvable Hardware for Spacecraft Autonomy, NASA JPL Technical Reports, Snowmass, Colorado, USA, 1998. http://techreports.jpl.nasa.gov/

[2] N. Muscettola et al., "Remote Agent: To Boldly Go Where No AI System Has Gone Before," IJCAI'97 - Japanese Conf. on Artificial Intelligence, 1997, Nayoga, Japan, pp. 48-60.

[3] W. Mangione-Smith et al., "Seeking Solutions in Configurable Computing," Computer, Vol. 30, No. 12, pp. 38-43, Dec. 1997.

[4] J. Villasenor et al., "Configurable Computing Solutions for Automatic Target Recognition," Proc. of IEEE Workshop on FPGAs for Custom Computing Machines, Apr. 1996, Napa, CA, pp. 70-79.

[5] E.A. Bezerra and M.P. Gough, "A Guide to Migrating from Microprocessor to FPGA Coping with the Support Tools Limitations," Microprocessors and Microsystems, Vol. 23, No. 10, pp. 561-572, Mar. 2000.

[6] D.K. Pradhan, Fault-Tolerant Computer System Design, Prentice-Hall, New York, 1996.

[7] J. Lach, W.H. Mangione-Smith and M. Potkonjak, "Efficiently Supporting Fault-Tolerance in FPGAs," Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (FPGA'98), Feb. 1998, Monterey, pp. 132-141.

[8] F. Kocan and D.G. Saab, "Dynamic Fault Diagnosis on Reconfigurable Hardware," Proc. ACM/SIGDA Design automation Conference (DAC'99), 1999, New Orleans, Louisiana, pp. 389-395.

[9] E.A. Bezerra, F. Vargas, M.P. Gough, "Merging BIST and Configurable Computing Technology to Improve Availability in Space Applications," Proc. 1st IEEE Latin American Test Workshop (LATW'00), Mar. 2000, Rio de Janeiro, Brazil, pp. 146-151.

[10] C. Stroud et al., "Bist-Based Diagnostic of FPGA Logic Blocks," Proc. of the Int. Test Conference (ITC'97), Nov. 1997, Washington, DC, pp.539-547.

[11] K. Huang and F. Lombardi "An Approach to Testing Programmable/Configurable Field Programmable Gate Arrays," Proc. of the 1996 IEEE VLSI Test Symp., 1996, pp.450-455.

[12] F. Vargas, and M. Nicolaidis, "SEU-Tolerant SRAM Design Based on Current Monitoring," XXIV Int. Symp. on Fault-Tolerant Computing (FTCS'94), Jun. 1994, Austin, Texas, pp. 106-115.

[13] E.A. Bezerra et al., "A VHDL Implementation of an On-board ACF Application Targeting FPGAs," Proc. Military and Aerospace Applications of Programmable Logic Devices Conf. (MAPLD'99), 1999, The Johns Hopkins University, USA, pp. 290-296.

[14] M.P. Gough, "Particle Correlator Instruments in Space: Performance Limitations Successes, and the Future," American Geophysics Union, Santa Fe Chapman Conf., USA, 1995.

[15] K. Beauchamp and C. Yuen, "Digital Methods for Signal Analysis," George Allen & Unwin, New York, 1979.

[16] Xilinx, "The Programmable Logic Data Book," Xilinx, San Jose, 1999.

[17] ESA, "Packet Utilisation Standard, Issue 1," European Space Agency (ESA) PSS-07-0, Noordwijk, The Netherlands, 1992.

[18] W. Lawrenz, "CAN System Engineering – From Theory to Practical Applications," Springer-Verlag, New York, 1997.

[19] ESA, "Cluster: mission, payload and supporting activities," European Space Agency (ESA) SP-1159, Noordwijk, The Netherlands, 1993.

[20] ESA, "Development of On-Board Embedded Real-Time Systems – An engineering Approach," European Space Agency (ESA) STR-260, Noordwijk, The Netherlands, 1999.

[21] O. Mattias et al., "Neutron Single Event Upsets in SRAM-Based FPGAs," Xilinx High Reliable Products, Internal Report, 1999, http://www.xilinx.com/products/hirel_qml.htm

[22] P. Alfke, and R. Padovani, "Radiation Tolerance of High-Density FPGAs," Xilinx High Reliable Products, Internal Report, 1999, http://www.xilinx.com/products/hirel_qml.htm

[23] G. Lum and G. Vandenboom, "Single Event Effects Testing of Xilinx FPGAs," Xilinx High Reliable Products, Internal Report, 1999, http://www.xilinx.com/products/hirel_qml.htm

[24] W.W. Peterson and E.J. Weldon, "Error Correcting Codes," MIT Press, Cambridge, MA, 1972.

[25] Xilinx, "Virtex Configuration and Readback," Xilinx Application Note 138 (XAPP 138), San Jose, Mar. 1999.

[26] IEEE, "Draft Standard For VHDL Register Transfer Level Synthesis," IEEE, USA, 1998.

[27] J. Lach et al., "Low overhead fault-tolerant FPGA systems," IEEE Trans. VLSI Systems, Vol. 6, No. 2, pp. 212-221, 1998.