Improving the Dependability of Embedded Systems Using Configurable Computing Technology [*]

Eduardo Bezerra [1], Fabian Vargas [2], Ahmet Ozcerit [3] and Michael Paul Gough [4]

[1, 3, 4] Space Science Centre
School of Engineering
University of Sussex
BN1 9QT, England
E.A.Bezerra@sussex.ac.uk
A.Ozcerit@sussex.ac.uk
M.P.Gough@sussex.ac.uk

[1] Faculty of Informatics
Catholic University, PUCRS
Porto Alegre – RS, Brazil
eduardob@inf.pucrs.br

[2] Electrical Engineering Dept.
Catholic University, PUCRS
Porto Alegre – RS, Brazil
vargas@ee.pucrs.br

Abstract

In this work, strategies for dependability improvement of embedded systems based in configurable computing technology are discussed. To better explore the possibilities, an embedded system for space application was chosen as a case study. The case study was first implemented in a high level of abstraction, using the VHDL language, targeting its utilisation in a situation where no fault tolerant requirements were needed. The requisites to increase the reliability and testability of this system are discussed here, as well as some expected results.

Keywords: *Computer Architecture, Real Time Computing, Fault Tolerance, Parallel Computing, FPGA, VHDL, Configurable Computing, Space Applications, and Embedded Systems*

1. Introduction

Common features of embedded systems, in general, are compactness and their application specific nature. In addition, some embedded systems have a higher demanding for processing power. A good example of this kind of system are the on-board instruments of spacecrafts, which also require fault tolerance capabilities. In order to meet these requisites, microcontrollers have been used in the design of such systems [1]. However, with the advances in the configurable computing field, it becomes possible to have systems with performance rates hundred or, in some cases, thousand times higher than traditional microcontroller based designs [2][3][4]. The first configurable computing system was proposed by Estrin in 1963 [5], but it could not be implemented before because of the technology limitations of the time. Nowadays, the best way to implement a configurable computer system is by using Field Programmable Gate Arrays (FPGAs) [2]. Some advantages of using FPGAs instead of microcontrollers in embedded systems are:

- Implementation of a real application-specific design. FPGA internal resources can be configured according to the application requirements. In a microcontroller, the application has to adapt to the resources available, and in many cases, not all resources are used;
- The Printed Circuit Board (PCB) for an FPGA may be simpler than the equivalent one for a microcontroller based design. This is because of the possibility of integrating in the same device (FPGA), external hardware components like, for example, FIFOs and state machines;
- The level of performance obtained with an FPGA is higher than with a microcontroller, even when using the same clock, because of the parallel nature of hardware.

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

The main disadvantages are the FPGA high cost and the obstacles dictated by the synthesis tools for developing systems in a high level of abstraction [6][7]. These problems are a result of the early stage of development of this technology. Despite these problems, the possibilities introduced by the configuration computing technology are an invitation for on-board instrument processing implementation [8]. The difficulty in accessing these long-life computers to execute maintenance, is a motivation for the investigation of strategies for dependability improvement [9][10].

The main objective of this work is to propose strategies for improving some dependability features of embedded systems for space applications. These strategies are based on the traditional ones used in the design of microprocessor systems, but taking in consideration features of this new technology. Configurable computing allows new possibilities, for instance, the use of a combination of strategies for fault tolerance in software and in hardware in 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. The paper is divided into three main parts. The first part consists of the sections 3 and 4, where aspects of system reliability improvement are discussed. In the second part, represented by section 5, some comments about system testability are introduced. The last part, section 6, shows reliability and performance figures expected. The case study introduced in section 2, is used as a base in all three parts. In section 7 there are some conclusions, and future directions.

2. Case Study: An Embedded System for Scientific Space Applications

The case study selected for this work is the FPGA implementation of an on-board instrumentation module of the SVALBARD sounding rocket, launched from Spitzbergen, Norway, in the winter of 1997/1998. The original module as flown consisted of a board with two DS87C520 microcontrollers (8051 family), FIFOs, state machines and software written in assembly language. In this case study this was re-implemented in VHDL [11], in order to investigate the feasibility of using FPGAs as the main processing elements, to replace microprocessors in special-purpose computer designs [12]. The main application performed by this module is an auto-correlation function (ACF) processing of particle count pulses [13]. The ACF is a statistical method that can be used to obtain information about the behaviour of a signal, revealing the presence of periodicity in a near random signal. An ACF is constructed by sampling a signal at two instants of time (t), separated by a lag (τ), measuring the signal amplitude (x), finding their product, and averaging over the time of the record (N). This procedure is better described by the equation:

$$R_x(\tau) = \frac{1}{N}\sum_{t=0}^{N} x(t).x(t+\tau) \qquad \textit{Equation 1.}$$

Although this ACF application was used to investigate the behaviour of energetic electrons at altitudes of up to 500 Km [1], the system as a whole, including the hardware and the software parts, is not too different from conventional embedded systems based in 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) as required by equation 1, and typical of DSP applications.

Both systems, the original design and the VHDL version, were implemented without taking into consideration any fault tolerant strategies. The main reason for that is the short mission duration, which was about 20 minutes long. The dependability improvements discussed next, are necessary in case of long-life applications where maintenance is not possible, or extremely expensive. An example of such a situation is a satellite carrying on-board scientific instrumentation including a similar ACF application to that flown on the rocket.

In the next sections, the three systems, that is, the original microcontroller implementation, the VHDL version and the fault-tolerant design, are named, respectively, SVAL, SVAL-VHDL, and FTSVAL-VHDL.

3. Improving the Case Study Reliability

Some modules of the SVAL-VHDL implementation are much faster than the SVAL respective ones, this being one of the most important advantages of using configurable computing to replace microprocessors. The performance figures are discussed later in the results section. In addition, there is a decrease in the PCB complexity as a consequence of the reduction in the number of components, which results in an improvement in the system reliability. FTSVAL-VHDL is designed to improve even more the SVAL-VHDL dependability features, without losing performance. It is obtained by including fault tolerance strategies in SVAL-VHDL. Two possibilities for the FTSVAL-VHDL implementation are described. In the first one, discussed next, fault-tolerant strategies are implemented inside the FPGA, allowing the same SVAL-VHDL board to be used. For the second one, discussed in section 4, additional external hardware is employed and, consequently, FTSVAL-VHDL version 2 has more hardware components than SVAL-VHDL, but not as many as it would be necessary for SVAL to reach the same reliability levels.

The block diagram in Figure 1 shows the basic hardware components necessary to implement both, SVAL-VHDL and FTSVAL-VHDL version 1. This implementation requires only the FPGA and a ROM memory for the configuration bitstream storage, used every time the system is initialised. The bitstream for the XQ4085XL [14], the FPGA chosen for this application, is 1,925 Kbits long, held in two 1 Mbits XQ1701L ROMs. These two components identified by the 'Q' letter, the FPGA and the serial ROM, belong to the Xilinx *QPRO* family of products for aerospace, defence and high reliability markets [15][16][17].

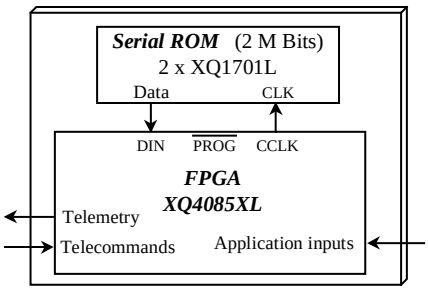

Figure 1. Block diagram of FTSVAL-VHDL (and SVAL-VHDL) hardware components.

Even employing high reliable devices, since FTSVAL-VHDL was conceived for long-life missions, additional fault-tolerance strategies are used in its design. In order to define the strategies, a very simple, but efficient, fault model was chosen for this kind of architecture. The faults considered in this fault model are *stuck-at* and *connectivity* [9]. The *stuck-at* faults are good representatives of the bit errors that can occur in SRAM based devices, as, for instance, FPGAs, which are sensitive to single event upsets (SEUs) caused by atmospheric high-energy neutrons [16][17]. The other modelled fault, *connectivity*, is responsible for more than 90% of the problems in a board and, as described later, special strategies as, for instance, bus replication and voters, are used to tolerate this problem. The strategies used to prevent and, when it is not possible, to tolerate the faults, belong to the fault model adopted, are described next.

3.1. SEU Prevention

In [16][17] there is a study showing the low SEU susceptibility of Xilinx FPGAs, and in order to improve even more the system reliability, in [17] a method to reduce the effects of SEUs in FPGA systems was proposed. Basically, in that method, 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, which allows the entire internal FPGA configuration to be read. If one input of the controller is different from the others two, then it is assumed that an SEU has occurred, and a reconfiguration of the faulty FPGA is executed. In [16] it was shown that a simple refresh operation, in this case by means of reconfiguration, is enough

to recover the device from a SEU. The main problem with the refresh recovery is the total loss of measurement data within the instrument system. 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 XQ4085XL). The block size has to be calculated according to the application time requirements.

The SEU prevention method proposed here is based on the refresh execution, but without the need of FPGA replication. As the case study 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 method can be used in SVAL-VHDL (Figure 1) with no need of additional hardware. The internal FPGA 15 Hz clock generator and a 19 bits counter, written in VHDL and implemented in the FPGA is used, together with the SVAL-VHDL processes. In the event of a rising edge pulse, generated by the 15 Hz clock, the counter is incremented. Every time the counter reaches the zero value, which happens about each 19.4 hours, 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.

3.2. Strategies for Connectivity Faults

The SEU prevention strategy described in the last section, is very efficient for fault prevention in the processing modules, as the operation units are implemented using the FPGA SRAM based look-up tables (LUTs). The control units of the processing modules, are partially implemented using flip-flops, and are one of the points not covered by this work.

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 Figure 2. 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 is sent to the application, otherwise, an error signal is set, invalidating the data. The block diagram was partially generated by Synplify [7], from a VHDL code with three instances of the VOTER entity listed in Figure 2. This entity was developed for synchronous input data. For asynchronous inputs, the clock signal in the sensitivity list has to be replaced by the input signals.

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
entity VOTER is
  port (CLK_IN      : in  std_logic;
        RESET_NEG_IN : in  std_logic;
        IP1_IN      : in  std_logic;
        IP2_IN      : in  std_logic;
        IP3_IN      : in  std_logic;
        OUT_OUT     : out std_logic;
        ERROR_OUT   : out std_logic
       );
end VOTER;
architecture VOTER_BEH of VOTER is
begin
  ERROR_OUT <= '0' when RESET_NEG_IN = '0' else
           '1' when ((IP1_IN /= IP2_IN) or
                     (IP1_IN /= IP3_IN) or
                    (IP2_IN /= IP3_IN)) else
           '0';
   VOTER_PRO: process (CLK_IN, RESET_NEG_IN)
   begin
    if RESET_NEG_IN = '0' then
       OUT_OUT   <= '0';
    elsif CLK_IN'event and CLK_IN = '1' then
       if (IP1_IN = IP2_IN) or
          (IP1_IN = IP3_IN) then
          OUT_OUT   <= IP1_IN;
       elsif IP2_IN = IP3_IN then
          OUT_OUT   <= IP2_IN;
       end if;
    end if;
  end process VOTER_PRO;
end VOTER_BEH;
```

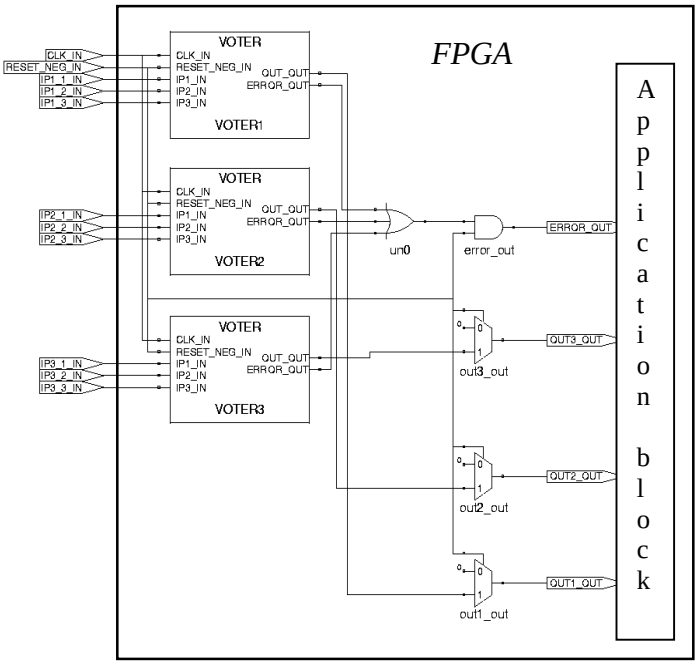

Figure 2. VHDL code for one voter, and block diagram of the voters and application inside the FPGA.

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, 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 is 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.

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 Figure 2, the input IP1_1_IN may be located in the pin 40, whilst the input IP1_2_IN 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 configuration file (bitstream) 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 SVAL-VHDL board shown in Figure 1, because the voters are implemented in the same FPGA along with the application, as shown in Figure 2. This strategy masks permanent, transient or intermittent faults efficiently.

4. Improving the Case Study Reliability with Additional External Hardware

The strategies described in the last section are used to increase the case study reliability, with no need for extra external hardware components on the board (see Figure 1). All strategies are implemented in the FPGA together with the application. The main advantage of this course of action is the possibility of having the whole system implemented in the same design input format, in this case, the VHDL language. A unique description format is ideal for the design process to apply fault avoidance techniques, as, for instance, the use of design rules and to simplify the revision activities executed during the whole design cycle [9].

In Figure 3, additional hardware components are included in the design in order that the system features improve even more. This also adds some flexibility to the system, taking advantage of the FPGA reconfigurability facilities. In this board the serial ROM is triplicated, with the bitstream for an application replicated in the three configuration memories, or, alternatively, it is possible to have three different bitstreams, representing three different applications. These two cases result in two different functioning modes, which are described next.

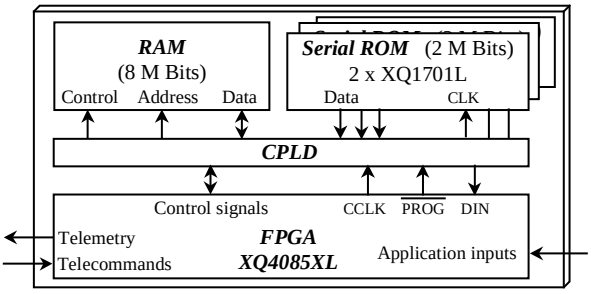

Figure 3. Block diagram of FTSVAL-VHDL, with extra external hardware components.

4.1. Case 1: Serial ROMs with the same configuration bitstream

In order to prevent SEU effects, the FPGA reconfigures itself at regular intervals, using an internal timer, as described before. Using the readback FPGA feature, the configuration bitstreams are also sent to the ground station to be compared to a correct bitstream. In case of an unsuccessful reconfiguration or negative result from the comparison, the FPGA attempts to reconfigure itself for a second time to avoid transient fault effects. If the second attempt is also unsuccessful, then the FPGA tries to use the redundant serial ROMs. For the worst-case scenario, a configuration bitstream is brought from the ground station, and stored in the 8 Mbits RAM. Once the RAM is loaded, the FPGA can be configured with the new bitstream. As shown in Figure 3, the RAM is controlled by a simple CPLD device, which is used to emulate a serial RAM, as the component used is a parallel one. For instance, when a new bitstream is received by the FPGA, it sends the bits to the CPLD, which is responsible for converting them to parallel and storing the bytes into the RAM. When the FPGA starts a reconfiguration procedure, and if the CPLD had already selected the three serial ROMs, then the CPLD reads the RAM, and sends the bitstream to the FPGA, converting the bytes into bits, following the FPGA CCLK configuration clock pulses.

The CPLD is responsible for the memory system management. However, when access to the RAM is necessary, the CPLD has first to ask a microkernel, implemented in the FPGA, for the new bitstream. This microkernel is implemented in VHDL, being part of the bitstream stored in the serial ROMs. Needless to add that, should all three bitstreams become completely corrupted, then the system crashes, because in this case it is not possible to store a new bitstream in the RAM. As ROM memories are more resistant to radiation than RAMs [18][19], then the main use of the RAM component in Figure 3, is for system upgrades. A new version of the ROM bitstream can be uploaded from the ground station, which characterises the use of hardware in the same way as software.

Important problems with relation to fault tolerance are the single points of failure. In the board in Figure 3, the CPLD and the FPGA are single points of failure, and the only prevention adopted here is the selection of high reliable parts.

4.2. Case 2: Serial ROMs with separate configuration data

In this mode, the serial ROMs include separate configuration data streams for each experiment. The same storage device is used to refresh the FPGA as long as the current experiment is active. When the experiment environment is required to be changed, the new configuration data is retrieved from the related memory device and a regular refresh operation is carried out from that memory from that moment on. In the case of an unsuccessful refresh operation the configuration bitstreams are brought from the ground station and stored into RAM (as in Case 1). Therefore, this mode of work cannot provide a fault tolerance service as efficient as in case 1. However, in this mode the system has more options to function with increased code space efficiency. Any fault upon refresh causes the RAM device to be loaded from a ground station and to be used subsequently throughout the mission. This mode of functioning is an actual example of a reconfigurable computing application, as the same FPGA device is used to implement several hardware configurations, according to external requests.

5. Improving the Case Study Testability

The start point of any fault-tolerance strategy is the fault detection, and the test is the mechanism used to do so. As described in the previous sections, the test is executed at two occasions and with two different objectives. In figure 2 the test is executed by the voter in order to guarantee correct input data to the system. Later, the functionality of the processing modules is verified on ground by comparing the bitstream received to a good one. This test approach is a major concern, because of the necessity of the bitstream transmission. When the time spent on this transference could be used to transmit useful data.

Another approach is to execute the test of the processing elements on board. In this case, the voters can ensure the reliability of the input data, but the integrity of the processed data during computation

is not guarantied. To cope with this problem, a periodical test can be executed by an external component (FPGA or microprocessor), or by internal components, for instance, one VHDL test entity for each VHDL processing entity. In both cases, the main problem is the amount of memory necessary for the test vectors storage. In order for the on board test strategy to be a viable option, a minimal test vector set, with a high fault coverage should be used.

The generation of an efficient minimal test set, and its posterior execution, as well as the use of fault detection facilities, is significantly eased, when design for testability (DFT) and design for reliability (DFR) techniques are used in all stages of the system development. In [20][21] a methodology was proposed to optimise system design towards reliability. The targets were the embedded systems developed using hardware/software (HW/SW) co-design techniques. The main idea behind that approach was to use reliability and HW cost constraints during the HW/SW partitioning, in order to select the best subset of all possible system partitions for the design. In that work only those parts mapped to HW and the communication channels are made reliable. Next, the system testability estimation procedure is based on an adaptation of the weak mutation analysis technique, and estimates, at a high-level of description, the system testability against transient or permanent HW faults.

Therefore, the methodology described in [20][21] can be used in the FTSVAL-VHDL design, not only for **reliability estimation**, but also for a **minimum test vector set generation**, for on-board testing. Using concepts from evolutionary computing, more exactly, from genetic algorithms [22], the idea is to define a minimal test set that can identify mutated chromosomes. In this case, a VHDL program is a chromosome, and syntactic alterations in the program represent malign genes in mutated chromosomes, which must be detected. To define a minimal test vector, it is necessary to generate a large number of mutated programs, and a large set of test vectors for the original VHDL program. Test vectors are considered mutation-adequated for a program, if they can distinguish the program from programs that differ from it by small syntactic changes [23].

Using the test vector set defined for the VHDL behavioural description, is a good option for the on-board test. It is also shown in [20][21] that for a given input test vectors set, the fault coverage obtained based on the mutation analysis of the VHDL description is always equal to or lower than the one obtained by means of the stuck-at fault model, at the gate level structure. These results show that the weak mutation analysis provides a *conservative measure* of fault coverage when compared to the one at the gate level structure. Such a condition allows us to take advantage of the proposed technique, which can be easily incorporated into typical system-level design flow, such as the case of the FTSVAL-VHDL prototype.

6. Numerical Analysis and Expected Results

6.1. The Reliability Evaluation of the System for Case 1 and Case 2

The proposed system has been analysed in numerical terms in three different modes using reliability evaluation techniques [9]. Since the reliabilities of the FPGA, CPLD and the rest of the components in the system are constant, they have not been included in the numerical analysis. Case 1 can be considered as four identical memories working in a parallel manner. A permanent fault in the currently active memory causes the next memory to be used for the refresh operation. On the other hand, in case 2 the RAM memory device may be used if any fault occurs in the ROM device. The RAM module, in this system, is used mainly in case of system upgrades because, as stated before, it is known that RAMs are more susceptible to radiation than ROMs and, consequently, the probability of a ROM presenting a defect before a RAM is very low.

To discuss the reliability improvements when using replicated information, it is considered a hypothetical situation where the failure rate (λ) is identical for each memory component. For this study it was chosen a failure rate of 0.0001/hour, to allow us generate quantitative information for comparison purposes. The reliability of each case can be found from the following expressions [9] for case 1 and case 2 respectively.

$$R_{case1}(t) = 1 - [(1 - R_{rom1}(t))(1 - R_{rom2}(t))(1 - R_{rom3}(t))(1 - R_{ram}(t))] \quad \textit{Equation 2.}$$

$$R_{case2}(t) = 1 - [(1 - R_{rom}(t))(1 - R_{ram}(t))] \quad \textit{Equation 3.}$$

If $\lambda_{rom1} = \lambda_{rom2} = \lambda_{rom3} = \lambda_{ram} = \lambda_{rom}$, and $R(t) = e^{-\lambda t}$ then,

$$R_{case1}(t) = 4e^{-3\lambda t} - e^{-4\lambda t} - 6e^{-2\lambda t} + 4e^{-\lambda t} \quad \textit{Equation 4.}$$

$$R_{case2}(t) = 2e^{-\lambda t} - e^{-2\lambda t} \quad \textit{Equation 5.}$$

As seen in Figure 4, the reliabilities of each case remain almost the same value at 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 10000 (416 days) hours of operation, case 1's reliability is 1.4 times better than case 2 and 2.33 times better than the non-redundant architecture.

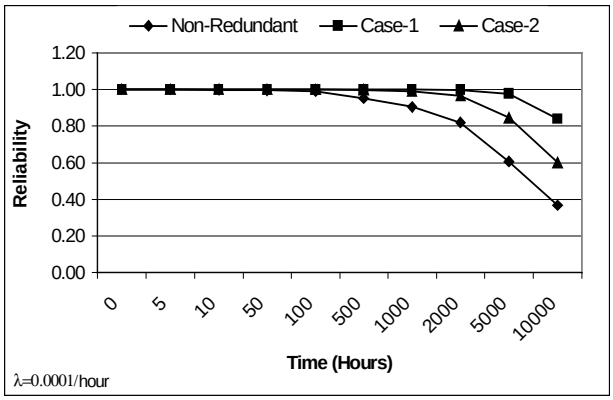

Figure 4. The reliability responses for each architecture against time.

6.2. Expected Performance Results

The main motivation for using FPGAs instead of microprocessors for on-board computer implementation, is the gain in performance with a decrease in the PCB area usage. Table 1 shows a comparison of the number of cycles (*T*) necessary to run some of the SVAL application processes in the original SVAL implementation (8051 implementation, written in assembly), and in SVAL-VHDL (FPGA implementation, in VHDL). The number of cycles required by FTSVAL-VHDL to run the ACF routines, is the same as SVAL-VHDL, as all the extra processes used to introduce the fault-tolerance capabilities to the system, are implemented to execute in parallel with the application. There are no performance penalties, because there is no need, for instance, to time share tasks.

	microcontroller	***FPGA***	***Rate***
Process 1	*4,518T*	*1T*	*4,518 times faster*
Process 2	*8T .. 36T*	*1T*	*8 to 36 times faster*
Process 3	*18T .. 1018T*	*1T .. 68T*	*18 to 14.97 times faster*
Process 4	*1,240T*	*48T*	*25.8 times faster*
Process 5	*1,334T..3,438T*	*132T..143T*	*10.11 to 24.0 times faster*
Process 6	*11,116T*	*288T*	*38.6 times faster*

Table 1. Performance comparison for the ACF application.

7. Conclusions and Future Work

Some possibilities for dependability improvement introduced by the configurable computing technology, were discussed in this paper. In sections 3 and 4 were described strategies for preventing SEU effects and to mask connectivity faults of the case study. In section 5 a new approach to estimate system testability and to determine the minimum test vector set based on an adaptation of the weak mutation analysis technique was discussed. In section 6, the expected system reliability and performance improvements were shown.

The strategies described in this paper deserve a deeper investigation, in order to be used in the design of a fault-tolerant on-board instrument processing system, entirely based on configurable computing. During the case study implementation (SVAL-VHDL), 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 [24]. 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 configuration bitstreams. 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. A possibility for future FPGA designs is to use Java with pre-optimised cores [25]. An important point to highlight, is that the use of a unique description format, as stated before, can improve the system dependability with the use of the above strategies from the very early design stages [9][26]. After selecting the language and the EDA tools, the next step will be the implementation of an FTSVAL-VHDL prototype, in order to determine the feasibility of the fault-tolerant strategies proposed here.

References

[1] Gough, M.P. **Particle Correlator Instruments in Space: Performance Limitations Successes, and the Future.** American Geophysics Union, Santa Fe Chapman Conference, 1995.

[2] Mangione-Smith, W. et al. **Seeking Solutions in Configurable Computing**. IEEE Computer, pp. 38-43, Nov. 1997.

[3] Villasenor, J. and Mangione-Smith, W. **Configurable Computing**. Scientific American, pp. 66-71, Jun. 1997.

[4] DeHon, A. **Reconfigurable Architectures for General-Purpose Computing**. PhD Thesis, Artificial Intelligence Laboratory. MIT, USA, 368p. Oct. 1996.

[5] Estrin, G. et al. **Parallel Processing in a Restructurable Computer System.** IEEE Transactions on Electronic Computers, pp. 747-755, Dec. 1963.

[6] Xilinx. **Synthesis and Simulation Design Guide.** Xilinx, 314p. 1998.

[7] Synplicity. **Synplify Better Synthesis – User Guide release 5.0**. Synplicity, 1998.

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

[9] Pradhan, D.K. **Fault-Tolerant Computer System Design**. Prentice-Hall; 544p. 1996.

[10] Moreno, J.M. et al. **Feasible Evolutionary and Self-Repairing Hardware by Means of the Dynamic Reconfiguration Capabilities of the FIPSOC Devices.** In Lectures Notes in Computer Science - v. 1478: Sipper, M. et al.(Eds.), Evolvable systems: From Biology to Hardware. Proceedings, IX, pp. 345-355. 1998.

[11] Perry, D.L. **VHDL - Second Edition** McGraw-Hill Series on Computer Engineering 390p. 1996.

[12] Bezerra, E. A. **Space Instruments: Migrating from Microprocessor to FPGA**. Space Science Centre; School of Engineering; University of Sussex, UK; Internal Report II, 18pp. April, 1999. <www.sussex.ac.uk/engg/research/space/>
[13] Beauchamp, K. and Yuen, C. **Digital Methods for Signal Analysis** George Allen & Unwin, 316p. 1979.
[14] Xilinx **The Programmable Logic Data Book** San Jose, 1999. <www.xilinx.com>
[15] Lum, G. and Vandenboom, G. **Single Event Effects Testing of Xilinx FPGAs**. Xilinx High Reliable Products; Internal Report; 5p. 1999. <www.xilinx.com/products/hirel_qml.htm>
[16] Mattias, O. et al. **Neutron Single Event Upsets in SRAM-Based FPGAs**. Xilinx High Reliable Products; Internal Report; 4p. 1999. <www.xilinx.com/products/hirel_qml.htm>
[17] Alfke, P. and Padovani, R. **Radiation Tolerance of High-Density FPGAs**. Xilinx High Reliable Products; Internal Report; 4p. 1999. <www.xilinx.com/products/hirel_qml.htm>
[18] Normand, E. **Single Event Upset at Ground Level**, IEEE Transactions on Nuclear Science, vol. 43, pp. 2742-2750, 1996.
[19] Olsen, J. et. al. **Neutron-Induced Single Event Upset in Static RAMs Observed at 10km. Flight Altitude**, IEEE Trans. on Nuclear Science, vol. 40, pp. 74-77, 1993.
[20] Vargas, F.; Bezerra, E.; Wulff, L.; Barros, D**. Optimizing HW/SW Codesign Towards Reliability for Critical-Application Systems**. 4th International IEEE On-Line Testing Workshop. Capri, Italy, pp.17-22; 6-8 July, 1998.
[21] Vargas, F. Bezerra, E., Terroso, A. and Barros, D**. Reliability Verification of Fault-Tolerant Systems Design Based on Mutation Analysis**. Proceedings of the SBCCI 98 - Brazilian Symposium on Integrated Circuit Design; Buzios, Rio de Janeiro, Brazil; 1998.
[22] Mitchell, M. **An Introduction to Genetic Algorithms.** MIT Press, 209 pp. 1998.
[23] Weiss, S.N.; Fleyshgakker, V.N. **Improved Serial Algorithms for Mutation Analysis**. International Symposium on software Testing and Analysis – ACM-ISSTA, Cambridge-MA, pp.149-158; Jun. 1996.
[24] IEEE **Draft Standard For VHDL Register Transfer Level Synthesis** IEEE, 1998.
[25] Chu, M. et al. **Object Oriented Circuit-Generators in Java.** BOOM, BRASS Object-Oriented Module-generators; 1998. <www.cs.berkeley.edu/projects/brass/BOOM>
[26] **From Hdl Descriptions to Guaranteed Correct Circuit Designs.** Proceedings of the Ifip Wg 10.2 Dominique Borrione (Editor); 1987.