A Self-Healing Real-Time System Based on Run-Time Self-Reconfiguration

Manuel G. Gericota, Gustavo R. Alves
Department of Electrical Engineering — ISEP
Rua Dr. Antonio Bernardino de Almeida
4200-072 Porto - PORTUGAL
{mgg, galves}@dee.isep.ipp.pt

José M. Ferreira
Dep. of Electrical and Comp. Eng. — FEUP
Rua Dr. Roberto Frias
4200-465 Porto - PORTUGAL
jmf@fe.up.pt

Abstract

The new generations of SRAM-based FPGA (Field Programmable Gate Array) devices are the preferred choice for the implementation of reconfigurable computing platforms intended to accelerate processing in real-time systems. However, FPGA's vulnerability to hard and soft errors is a major weakness to robust configurable system design.

In this paper, a novel Built-In Self-Healing (BISH) methodology, based on run-time self-reconfiguration, is proposed. A soft microprocessor core implemented in the FPGA is responsible for the management and execution of all the BISH procedures. Fault detection and diagnosis is followed by repairing actions, taking advantage of the dynamic reconfiguration features offered by new FPGA families. Meanwhile, modular redundancy assures that the system still works correctly.

1. Introduction

The use of reconfigurable computing platforms in real-time systems enabled a significant speedup in performance over traditional, non-reconfigurable, implementations. New and more flexible systems are achieved using soft microprocessor cores implemented in SRAM-based FPGAs, and by mapping compute--intensive sections of an application to reconfigurable hardware. This enables a quicker response to changes in the system's environment, which is an important feature when very stringent response intervals are imposed to the system.

This degree of integration and flexibility was made possible by the reduction in the size of transistors in each new generation of semiconductor technology, which led to a greater integration and to a per unit power reduction, enabling chips to grow in size and complexity. However, new submicron scales also brought some negative aspects, namely the vulnerability to soft errors, also called single-event upsets (SEUs), which are radiation--induced transient errors caused by neutrons from cosmic rays and alpha particles from packaging material. Until now, they used to be a major concern only for space applications. But, for designs manufactured at advanced technology nodes – such as 90 nm, 65 nm, and downward – system-level soft errors became an issue also at ground level. They are now much more frequent than in previous generations [1]. By this reason, the use of fault tolerance techniques, once confined only to specific applications requiring high levels of security or operating on harsh environments, became mandatory, especially when dealing with hard real-time systems where a system failure may lead to catastrophic consequences.

Soft errors do not physically damage the chip, but the values stored in memory cells may be affected, causing incorrect data to be transmitted or an improper instruction to be retrieved by a processor. This problem has a particular impact on the reliability of SRAM-based FPGAs, currently the preferred choice for the implementation of reconfigurable computing platforms, because the structural definition of the configured functions relies on memory cells, which makes them especially vulnerable to soft errors. Additionally, the amount of embedded memory blocks available for user's applications is also increasing.

Another negative aspect due to the smaller technological scales is the increased threat of electromigration, which may result in permanent physical damages to the chip. The number of defects related to small manufacturing imperfections that are not detected by production testing has been growing as scale goes down. These defects are especially prone to electromigration phenomena, resulting, after large periods of operation, in the emergence of permanent faults.

The recent addition of new features to FPGAs, such as dynamic reconfiguration and self-reconfiguration, may help to cope with the problems mentioned above, in particular when dealing with hard real-time systems that require a high reliability level.

Dynamic reconfiguration involves the reconfiguration of a fraction of the configurable resources, without disturbing the operation of those functions that are not modified, while self-reconfiguration [2] enables currently configured functions to control the dynamic

♦ This work is supported by an FCT program under contract POSC/EEA-ESE/55680/2004

0-7803-9402-X/05/$20.00 © 2005 IEEE

reconfiguration of other areas of the same FPGA. Their combined use makes feasible the implementation of the autonomous recovering mechanism proposed here. The mechanism has the ability to restore the fault-free operation of the system when a faulty condition is detected.

The implementation of online structural test and fault tolerance strategies were largely explored in previous works [3-4]. However, those previous approaches relied on a rotate and test methodology, whose primary aim was the structural test of the FPGA. Moreover, only a small fraction of the resources were configured to be tested at a time. These solutions create a test latency that must be taken into account since it degrades the performance of the test strategy. If a defect affects the functionality of a given function, the resulting fault will be propagated until the test function reaches the defective resource. By then, the fault may already have had catastrophic consequences.

This paper presents a new methodology that aims to increase the reliability of real-time systems based on reconfigurable platforms implemented in dynamically reconfigurable FPGAs. The drawback associated to previous approaches is avoided by the introduction of fault tolerance techniques.

In the next section traditional hardware redundancy techniques are briefly analyzed, followed by the presentation of the proposed methodology. Several aspects related to its practical implementation are then discussed, and future research lines are presented in the concluding section.

2. Hardware redundancy techniques

Traditionally, highly critical applications relied on hardware redundancy to increase their reliability. One of the best known of such approaches is Triple Modular Redundancy (TMR), a static redundancy technique that achieves fault tolerance without actually detecting any fault. In this method, each module, which may be a complete system, such as a computer, or a less complex unit, like a microprocessor or even an adder or a gate, is replicated three times. The voting element collects the outputs from the three sources and delivers the majority vote at its output. In this case, it is assumed that the majority voter does not fail, which is an unrealistic principle. When this assumption is not verified, the reliability of the voter element will determine the reliability of the circuit, since it will fail if the voter fails. However, the reliability of a voter in a redundant system can be improved by replicating this element as well, in a scheme that is called T-TMR [5].

In new nanometre technology, the use of fault tolerance mechanisms is essential, not only due to soft errors, but because it is unrealistic to expect that a manufacturing test will cover all possible faults. In particular, delay faults emerging from defects of resistive type, or due to crosstalk or ground bounce, are almost impossible to foresee [6].

Hardware redundancy is also a preferred choice to improve the reliability of highly critical applications based on FPGAs [7-8]. Due to their inherent configurability, FPGAs are especially suitable for the implementation of modular redundancy, since it does not require any new architectural feature and it is function independent.

If it is clear that hardware redundancy increases the reliability of a system, it is also obvious that any proposed methodology has also to take into consideration the cumulative impact of single errors, as their added effect may lead to the quick disruption of a system. The great advantage of using reconfigurability is that in the event of a module failure a diagnose-and--repair mechanism may be activated and the initial redundancy re-established. This may be done transparently and without human intervention, since physical component replacement is not needed. This means that a higher level of maintainability is achieved, without even implying the inoperability of the affected circuit, since it is protected by TMR. This is both true to hard and soft errors, despite the different repair mechanisms that must be adopted to overcome them.

3. The Built-In Self-Healing methodology

In a TMR implementation, if a module fails it should be promptly replaced to keep the reliability level. However, it is not easy to detect a fault in a TMR implementation using traditional online test strategies, due to the inherent masking properties of redundancy.

In our approach we propose the implementation of a Built-In Self-Healing (BISH) methodology, which can be divided into three tasks: detection, diagnosis and repair. These tasks are controlled by a soft microprocessor core implemented in the same FPGA, and having a compatible reliability index. Due to the usual long time interval between module failures [9], a generic soft microprocessor core that carries on other tasks related to the operation of the real-time system may be used for this purpose. At the moment, this methodology is being applied only to soft-errors, but we plan to extend its usage to hard errors, making use of active replication techniques [10].

The detection of faults is done through a scan chain that regularly captures the values at the outputs of all the modules and voters, including those of the soft microprocessor core, as shown in figure 1.

Upsets may also affect the values shifted through the scan chain, thus leading to wrong fault diagnosis and consequently to the extemporaneous activation of a repairing mechanism. However, despite representing an additional unnecessary task for the reconfiguration mechanism, it does not affect system operation. If the structural configuration of the scan chain is affected by a

fault, either due to a hard or soft error, several neighbouring bits in the scan chain will be disturbed, indicating that a simultaneous general failure in all modules of one or more functions is taking place. If this happens, and since the probability of a general failure is very low, the scan chain must be checked first. The Boundary Scan (BS) chain may also be used to capture each FPGA output [11]. As a hard-wired implementation, this scan chain is less prone to soft errors.

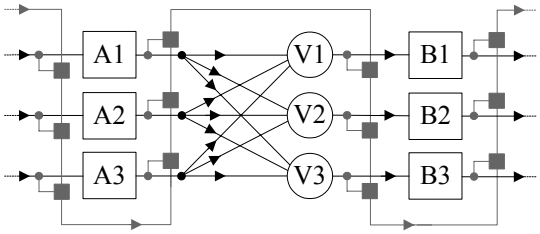

Fig. 1. Example of a T-TMR implementation with a scan chain

The captured bitstream is shifted to the internal microprocessor where it is analyzed. Since the scan chain cells completely wrap the modules and voters, it is possible to confine the origin of an error to the space between them, corresponding to the module or voter where the value was captured, and to the interconnections in-between [12].

Three possible causes for a fault to appear may be considered:

1. the faulty value is due to a soft error affecting one of the circuit registers;
2. the faulty value is due to a soft error affecting a configuration memory cell, which leads to a change in the functionality of the module or voter or in the routing of signals;
3. the faulty value is due to a permanent physical defect affecting the structure of the FPGA.

The first case may be immediately excluded if the error is captured at the output of a voter, since voters are typically implemented using combinational logic only. If it has its origin in a module, one can expect that the fault will be automatically corrected at the next register update. A new scan chain capture operation may show that the error has already been fixed and no further action is needed. If not, the second situation may have occurred.

In this case, a background task is launched to readback part of the configuration bitstream of the area where the affected module is implemented. Comparison with the original bitstream may be done by bit comparison or Cyclic Redundancy Check (CRC). If an incoherency is found, the microprocessor performs a partial reconfiguration of the area where the supposedly affected module is implemented, restoring the original configuration and eliminating the cause of the failure. The output of the module is captured again and its correctness verified. This technique is known as scrubbing, and defined as the process of re-writing the configuration memory during (and without disturbing) normal FPGA operation [13].

If no error on the configuration bitstream is detected, but the fault persists, the most probable reason is the existence of a physical defect in the array. To restore the reliability index, the affected module has to be relocated to a fault-free area and its input and output connections re-established releasing the faulty area to be tested [4]. This procedure is controlled by the microprocessor. When the defect location is identified, the defective resource is "marked down", to avoid its use in case of future reconfigurations. A list of faulty resources is maintained in memory by the microprocessor. This memory must also be protected against upsets using error checking and correction techniques based on Hamming or Hsiao codes [6].

The remaining resources that are tested OK can be reused in later replacements of any other faulty module. In this way, the available spare resources are almost entirely restored for future replacements. Figure 2 shows the diagram flow of the proposed methodology.

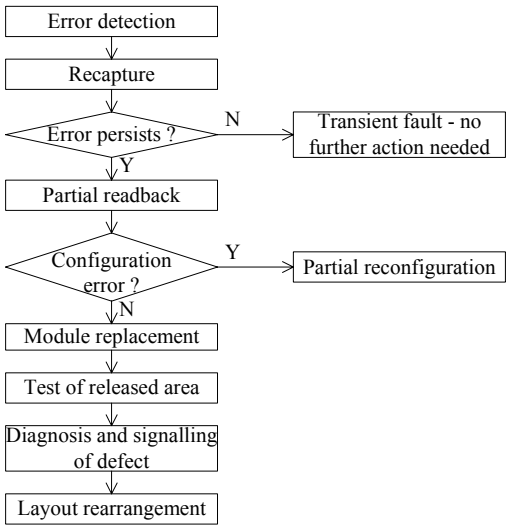

Fig. 2. Flowchart of the proposed BISH methodology

This methodology extends the reliability of each function and enables a smoother degradation of the global reliability index. Despite being a static T-TMR implementation, a faulty module or voter is dynamically repairable n times, where n depends on the cause of the failure. If the origin is not a permanent physical defect, then n is infinite. Otherwise, n depends on the initial amount of spare resources and on the location of the defects that affect the structure of the FPGA.

The microprocessor is also implemented using T-TMR to ensure a reliability index compatible with the remaining blocks. The microprocessor is divided in small functional modules, facilitating replacement in

case of fault detection, and reducing the spare space needed for relocation in case of fault detection. If the defective module is part of one of the three implemented processors, the remaining two will be responsible for the replication of the malfunctioning module. Subsequent test procedures will already be assumed by the whole three.

Self-reconfiguration is necessary to embed the whole system in a single FPGA, including the BISH features. The Virtex-II and Virtex-II Pro families have an Internal Configuration Access Port (ICAP), which enables a soft microprocessor core to control its own dynamic reconfiguration or the reconfiguration of any external modules, without stopping or disturbing the operation of the whole system.

To be able to implement the proposed methodology the soft microprocessor core shall be able to manipulate directly the FPGA configuration bitstream. This is necessary to create partial reconfiguration files for scrubbing and replication procedures. To support this feature, a software tool is being developed, based on the JBits software – a set of Java classes that provide an Application Programming Interface (API) to access the Xilinx FPGA bitstream [14]. This tool will create the partial configuration files and will carry out the partial and dynamic reconfiguration of the FPGA through the ICAP interface. Consequently, the microprocessor shall be prepared to run this software. Two solutions are being considered: the use of a generic microprocessor; or the use of a Java processor. In the first case, a generic soft processor core will run a Java Virtual Machine (Java VM) developed specifically for that microprocessor and to support the set of Java classes used by the reconfiguration tools. The disadvantage of this solution is the amount of memory needed to hold the JAVA VM.

The second hypothesis, the use of a Java processor, seems to be the most adequate solution for the inclusion of the BISH feature, since the software necessary to its implementation is developed using Java. These will also speed up its execution, reducing time latency between detection and correction of any fault. The disadvantage of this solution is that any other applications concerning the operation of the system, not related to the BISH feature, have to be rewritten in JAVA, which, in some cases, may not be feasible. However, this should not be a problem when developing a completely new product.

4. Conclusions

In this paper a new methodology aimed to increase the reliability of real-time systems is presented. Apart from the presentation of the whole project and of the detailed description of some of the solutions already assumed, the proposal presents a set of issues that are being studied and must be sorted out to ensure its complete success.

Further research is necessary and several issues related to the use of TMR in reconfigurable systems have yet to be considered. Current work is being done towards their resolution and integration into the project.

References

[1] S. Mitra, N. Seifert, M. Zhang, Q. Shi, K. S. Kim, "Robust System Design with Built-In Soft-Error Resilience," *Computer*, vol. 38, no. 2, pp. 43-52, February 2005.

[2] B. J. Blodget, S. P. McMillan, P. Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs," *Proc. DATE - Designers' Forum*, pp. 399-400, 2003.

[3] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriya, V. Verma, "Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications," *Proc. Intl. Test Conf.*, pp. 973-982, 1999.

[4] M. G. Gericota, G. R. Alves, M. L. Silva, J. M. Ferreira, "Active Replication: Towards a Truly SRAM-based FPGA On-Line Concurrent Testing," *Proc. 8th IEEE Intl. On-Line Testing Workshop*, pp. 165-169, 2002.

[5] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. San Francisco, CA: Morgan Kaufman Publishers, 2001.

[6] M. Nicolaidis, L. Anghel, "Concurrent checking for VLSI," *Microelectronics Journal*, Vol. 49, Nos. 1-2, pp. 139-156, November 1999.

[7] C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs," *XAPP 197 Application Note*, Xilinx, Inc., 37 p., 2001.

[8] N. R. Saxena, S. Fernandez-Gomez, Wei-Je Huang, S. Mitra, Shu-Yi Yu, E. J. McCluskey, "Dependable Computing and Online Testing in Adaptive and Configurable Systems," *IEEE Design and Test of Computers*, Vol. 17, No. 1, pp. 29-41, Jan.-March 2000.

[9] P. L. Murray, "Re-Programmable FPGAs in Space Environments". *Available at:* http://www.seakr.com/data/Unsorted/reprogrammable_fpga_in_space1.doc

[10] M. G. Gericota, G. R. Alves, M. L. Silva, J. M. Ferreira, "Run-time Defragmentation for Dynamically Reconfigurable Hardware," in: *New Algorithms, Architectures and Applications for Reconfigurable Computing*. Springer, 2005. ISBN 1-4020-3127-0.

[11] *IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std 1149.1)*, IEEE Std. Board, 2001.

[12] J. H. Lala; R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 25-40, January 1994.

[13] C. Carmichael, M. Caffrey, A. Salazar, "Correcting single-event upsets through Virtex Partial Configuration," *XAPP 216 Application Note*, Xilinx, Inc., 12 p., 2000.

[14] S. A. Guccione, D. Levi, P. Sundararajan, "JBits Java based interface for reconfigurable computing," *Proc. 2nd Military and Aerospace Appl. of Prog. Devices and Technologies Conf.*, 1999.