A Framework for Reliability Assessment and Enhancement in Multi-Processor Systems-On-Chip

G. Beltrame[2], C. Bolchini[1], L. Fossati[1], A. Miele[1], D. Sciuto[1]
[1]Dip. di Elettronica e Informazione, Politecnico di Milano
P.zza L. da Vinci, 32 - 20133 Milano, Italy
{*bolchini*, *fossati*, *miele*, *sciuto*}@*elet.polimi.it*
[2]European Space Agency
Keplerlaan 1, 2200 AG Noordwijk, The Netherlands
giovanni.beltrame@esa.int

Abstract

Reliability issues play a relevant role in the design of embedded systems for critical applications; this and the always increasing performance requirements lead to the adoption of new architectural solutions, as shown by the introduction of Multi-Processor Systems-on-Chip (MPSoC). MPSoCs raise new challenges related to the complexity of the interactions among several independent cores. This paper presents a framework, based on a simulation platform, for the design of this kind of embedded systems; the framework supports the use of reliability techniques in order to address fault detection and tolerance issues. The simulation platform is also adopted for a reliability assessment task, achieved by exploiting fault injection targeting each component of the system and by monitoring the effects on the entire architecture.

1 Introduction

The continuous increase of transistor density on a single die is leading towards the production of more and more complex systems on a single chip, with an increasing number of components. This brought to the introduction of the System-On-Chip (SoC) architecture, that integrates on a single medium all the components of a full system. However, power and heat dissipation, difficulties in increasing the clock frequency, and the need for technology reuse to reduce time-to-market push towards different solutions from the classic single-core or custom technology. A solution that is gaining widespread momentum consists of exploiting the inherent parallelism of applications, by executing them on multiple off-the-shelf processor cores. Having separate cores on a single chip reduces hot-spots, allows better usage of the chip surface and provides more possibilities to exploit parallelism. This brought to the definition of Multi-Processor System-on-Chip (MPSoC). The design of MPSoC raises new challenges due to the large design space and tight design and time-to-market constraints. MPSoC are complex devices, and therefore they require some particular modeling techniques that are able to hide their inherent complexity. Nevertheless, the model has to be accurate enough to describe the entire system throughout the phases of its development, and has to provide enough flexibility to be refined iteratively up to the point where the actual device can be produced using current process technology.

1550-5774/07 $25.00 © 2007 IEEE
DOI 10.1109/DFT.2007.35

In this context, when combining independently designed modules, the enhancement and assessment of reliability becomes particularly important; for instance specific approaches are required in order to be able both to apply fault detection and fault tolerance techniques from the initial steps of the design flow and to evaluate the effects of faults in a component while interacting with the other ones composing the MPSoC. These reliability issues are becoming more and more relevant, as the incidence of soft errors grows also at ground level [7]. Such errors are caused by radiation, and they temporarily affect memory elements so that their content may be corrupted; this situation, particularly hazardous in safety-critical systems, it is serious in general, especially when considering the embedded systems' pervasiveness in today's life. A few other co-design frameworks taking into account reliability have been proposed in the past [16, 6, 2, 15], but they are suited for other architectural solutions. When considering an MPSoC architecture, the effects of a fault in a module, if not contained, may corrupt also the other components, thus particular attention has to be devoted to the design of the overall system, as a collection of critical elements.

The main contribution of this paper is the insertion of a Reliability-Aware layer inside a framework for the design exploration of Multi-Processor Systems-on-Chip, thus adding this aspect to the figure of merit used during exploration. This layer is able to introduce fault detection and fault tolerance properties in the system under consideration, by acting at high abstraction level and by accessing a dedicated library of hardware and software design techniques for reliability. The second significant aspect of our work consists in the use of an MPSoC simulation platform to perform reliability level assessment by means of Software-Implemented Hardware Fault Injection (SWIFI). This platform allows an analysis of the behavior of the designed system in the presence of failures.

The rest of the paper is organized as follows. The next section briefly discusses related work. Section 3 describes the proposed design framework, allowing the exploration of the solution space also from the reliability point of view, in order to identify the most convenient trade-off between costs and benefits, fulfilling the designer's requirements and constraints. Section 4 details the MPSoC simulation platform, supporting the proposed design framework and allowing the evaluation of the various possible implementations, and also shows how we exploit such platform for injecting hardware faults in a component model and for simulating its effects within the complete MPSoC environment. Finally, Section 5 closes the paper drawing some conclusions.

2 Related Work

In the past, only a few works [16, 6, 2, 15] proposed a hardware/software co-design flow taking into account reliability issues, and considering fault detection and tolerance properties as relevant metrics in the exploration of the solution space. Some of these proposals target classes of faults different from the Single Event Effect, which, nowadays, is considered particularly interesting.

In [6] the authors provide fault detection capabilities by applying duplication and comparison through assertions, evaluating the caused area overhead, but using no other figure of merit. With respect to [2, 15], the proposed platform is characterized by a higher complexity, and the fault injection simulator provides a valuable tool to verify the achieved reliability level. This is particularly interesting when considering the limited access to the resources structure and to the architecture of the adopted system. More precisely, the

approach presented in [2] has been generalized to be suitable for different design flows and frameworks, in relation with the architectural solution at hand.

The approach proposed in this paper, focusing on the Single Event Effect fault model, mainly differs from previous work for its application target, Multi-Processor System-on-Chip, that introduces new challenges. First of all the number of available cores in the final architecture, as well as their off-the-shelf nature, introduces new degrees of freedom in the application of reliability techniques, but also new complexity due to the interaction between IP cores, that can be manipulated and monitored only externally, with little customization possible.

3 The proposed framework

The aim of the proposed framework is to enable design space exploration for reliable MPSoCs, and to guide the designer in the evaluation of the possible design implementations toward an optimal solution. Acceptable solutions are determined on the basis of a set of possible architectures, constraints and requirements, and by accessing a library of available hardware and software techniques for the implementation of reliability properties.

The framework offers capabilities for modifying and managing the system specification to introduce design for reliability requirements; this step can be followed by the exploration of the design space: it is thus possible to identify an architecture that satisfies the given constraints and that ensures good performance both in terms of speed and power consumption. Finally, the issue of the experimental evaluation of the obtained solution, in terms of reliability level and performance, is taken into account by providing a simulation environment capable of fault injection activities and performance profiling.

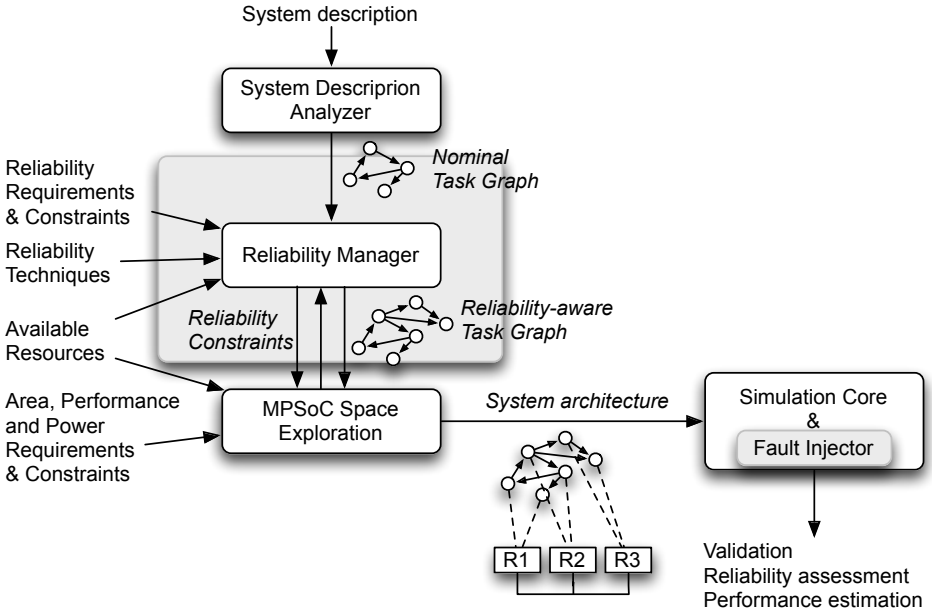

Figure 1. The proposed framework

An existent tool for MPSoC design space exploration, called PandA, has been extended by introducing a "Reliability-aware layer" similarly to the methodological approach of the

work proposed in [2] for simpler architectures. Figure 1 shows the modules of the basic framework and the extended modules (with a shaded background). The basic framework aims at determining an optimal implementation of the system under analysis given a target multi-processor architecture. This is obtained by first exploring the solution space and then by identifying, among the found solutions, the most suitable one with respect to the requirements and constraints (in terms of area, performance and power).

The design activity takes as input a system specification described with a high level language (C, C++ or SystemC). The first step, performed by a module called *System Description Analyzer*, translates the specification into an internal representation to be easily managed and modified, a graph of tasks, called *task graph*, where nodes represent basic tasks of the system description (i.e. each of them is composed by some of the instructions of the original specification). The obtained task graph is fed to the next module that explores the system design space: by taking into account requirements and constraints, this module tries to find an optimal binding among the nodes of the task graph and the available architectural resources. Finally, an architecture simulator is provided for an experimental evaluation of system performance; simulation is also used to tune the parameters of the system (for example, cache size and memory latency) to reach the desired goal.

Considering the presented framework, reliability issues are taken into consideration by including an additional layer for the reliability design space exploration and by enhancing the simulator for supporting fault injection. Figure 1 highlights additional modules with a gray background area, whereas Figure 2 details the introduced layer.

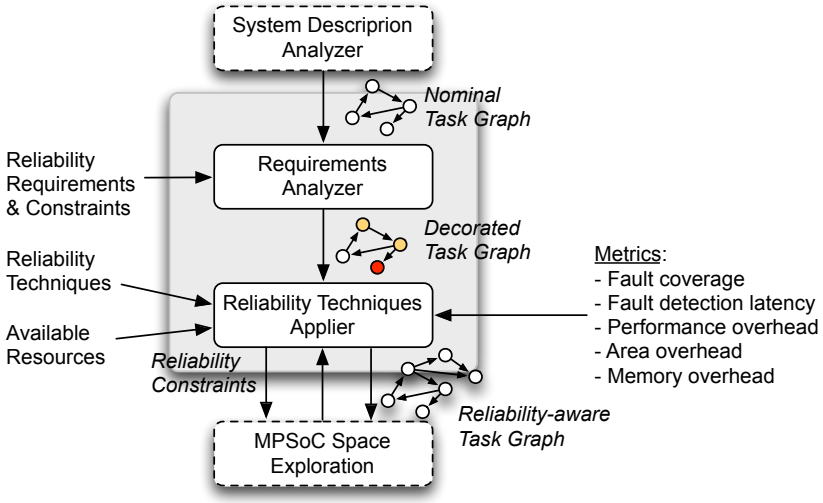

Figure 2. The reliability layer

This layer is composed of two main modules, a *Requirements Analyzer* and a *Reliability Techniques Applier*. The former supports the designer in specifying the parts of the entire systems that are critical, and for which fault detection and/or tolerance properties need to be provided; the designer can introduce such criticality indications either in the initial specification (and the tasks associated with reliability requirements will inherit them) or on the nominal task graph. The module produces a *Decorated Task Graph*, where each task is characterized with additional information about the properties that have to be guaranteed and with the required level of reliability (detection, tolerance, recovery, etc.).

Requirements and constraints on communication are derived by the module, by analyzing the requirements expressed on tasks and the direct specification on the nominal task graph.

In the next step, the *Reliability Technique Applier* analyzes the decorated task graph and manipulates it by applying available reliability techniques. During this phase, several metrics can be adopted to estimate system quality and to guide design space exploration; considering reliability properties assessment, the identified parameters are fault coverage and detection latency, while costs are evaluated in terms of area and performance overhead.

The second module accesses a repository where various techniques (acting on the hardware as well as on the software) for guaranteeing fault detection/tolerance properties are stored. These techniques are applied by introducing redundancies in the tasks and/or by introducing constraints which require the binding of tasks to reliable components by construction, i.e., resources that are intrinsically fault tolerant such as Totally Self-Checking (TSC) checkers. As an example, when Triple Module Redundancy is applied on a task, the corresponding node is triplicated and a node representing a voter is included; furthermore constraints are added to guarantee that the three replicas are managed by different resources and that the voter must be TSC itself. As another example, if fault detection only is required, task duplication is considered as a viable technique to be applied with respect to space or time redundancy; software code replication is another available technique, modifying the task internally. Each one of these solutions is characterized by costs, performance and depends on the available resources.

The result is a modified, *Reliability-aware* task graph that contains the original information about the system specification together with the information about reliable properties and the set of binding constraints. These results are the inputs to the MPSoC *Space Exploration* module, part of the basic flow, that produces the final system implementation.

In order to verify the correctness of the design and to compare different implementations with respect to the adopted figures of merit (e.g., area, performance and reliability), a hardware simulation platform, called ReSP, has been developed. This tool, enhanced and instrumented, is also used to evaluate the reliability of the final implementation, as detailed in the following section.

4 ReSP: The MPSoC simulation platform

We analyzed several simulation platforms found in literature, considering the opportunity to include them in the framework, but no one satisfied completely our necessities. Their limitations are mainly related to a lack of flexibility, hence reducing the possibility to introduce features for managing reliability issues and fault injection.

StepNP [11] is a platform featuring instruction-set simulators wrapped in SystemC: even if it provides introspection mechanisms, StepNP does not use the TLM standard [3] and it has a limited set of available components. MPARM [1] is another simulation platform targeted to multi-processor systems; it is provided with a large support and availability of components but it does not feature a coherent control-and-view model subsystem that would allow component introspection. Platform Architect [5] is a SystemC-based design environment provided with a graphical user interface and a standardized component library. Although it is a powerful design tool for SoCs, Platform Architect lacks the definition of a parallel programming model, and it does not specify guidelines for the implementation of complex communication architectures like NoCs, making it not perfectly suitable for

MPSoCs. Other simulation platforms are not suitable because dedicated to specific issues: GRAPES [10] is mainly focused on memory architectures and synchronization issues, and Simics on software developments [9].

Most of these, and other simulation platforms, are based on TLM; this was introduced in the past years as a modeling style to describe on-chip communication channels at a higher abstraction level with respect to Register Transfer Level (RTL) and as a solution to manage the increasing complexity of modeling Multi-Processor Systems-on-Chip. With TLM, IP modules can be modeled at a functional level and the system bus behavior can be viewed as an abstract channel independent from the target bus architecture or protocol implementation.

The limitations of the reviewed platforms led us to the development of a new simulation platform, called ReSP (*Re*flective *S*imulation *P*latform), used to support, test and evaluate the proposed methodology in relation to MPSoC reliability properties.

4.1 Reliability analysis via fault injection

ReSP is provided with reflective capabilities that allow us to query and modify the internal structure of the hardware components at runtime without the need to instrument the system description used to define the final implementation. This characteristic is ideal to be exploited for enabling fault injection campaigns first by modifying the components' internal state and then by simulating the system behavior as in presence of a hardware failure. In this way it is possible to significantly reduce the time required for setting-up the experimental environment, and since no extra elements are introduced to inject faults, no uncontrolled/masking effects are caused.

The simulation platform structure is shown in Figure 3: it is composed of two main modules, an *Architecture Builder* for building the required system architecture on the basis of a set of available components and a *Simulator Core & Fault Injector* that executes the simulation and provides features for faults injection.

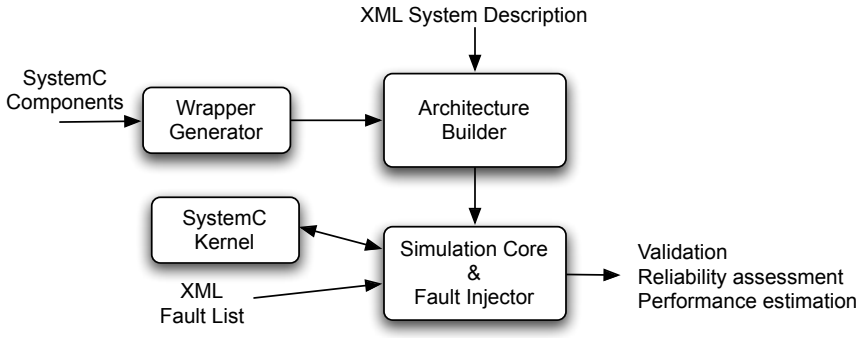

Figure 3. The architecture of the ReSP hardware simulation platform

The simulator is built using the Python programming language [12]; being scripted and interpreted, this language possesses powerful reflective features: introspection inside architectural components is allowed, i.e., it is possible to read and modify the internal state of an object at runtime, without instrumenting the code. This feature was exploited to provide the simulation platform with both debugging and fault injection capabilities. More precisely, this aspect is used to model the effect of SEU faults in terms of bit-flips, acting

on the values stored in memory elements. In general, it is possible to model any kind of functional failure causing an erroneous value at any point during the application execution.

In order to make hardware components, written using SystemC and TLM libraries, compatible with the simulation core, the facilities provided by the Boost.Python and Py++ to automatically create Python wrappers around the C++ objects have been used.

Thanks to the reflective capabilities of Python, SystemC components can be easily integrated with ReSP without the need of modifications or creation of specific wrappers; this favors external IP reusability and the description of new hardware architectures by composition of already existing components. Currently, the simulation platform includes the following components:

- *processors cores* written using the ArchC [14] Architectural Description Language; we possess both the functional and cycle accurate versions of the PowerPC, Leon2 and ARM7 RISC processors;
- *interconnections* in terms of bus and Networks-On-Chip;
- *memory hierarchies* including simple memories and caches.

In order to implement debugging functionalities, the SystemC kernel was extended by providing the possibility of pausing the simulation either when specific events are raised or when a specified amount of time elapsed. After the system halts, introspection can be used to check the internal state of the components (for example the values of the registers of a processor), eventually also modifying it; simulation may then be restarted. This same feature has been exploited to support fault injection and to derive reliability assessments, as discussed in the next section.

4.2 Reliability Analysis

The fact that it is possible to suspend execution, modify an internal value stored in a memory element and resume execution, allows to simulate the fault effects, thus being able to evaluate the behavior of the system in presence of errors and its capability to detect the fault, and recover from it, or just mask it. The classes of faults that can be simulated strictly depend on the abstraction level adopted by the component description; when the available model is functional the injected and simulated faults are considered at behavioral level, whereas if the component is described at RTL level, the considered fault set is defined at logical level.

When a fault injection campaign is executed, the simulator builds a golden model (i.e., a copy of the architecture) in addition to the architecture under test; the fault becomes observable when, after fault injection, the output of the golden model differs from the output of the model where the fault is injected. Then, simulation can be automatically stopped and it is possible to explore the internal state of the architecture and compare it with the state of the golden model. In case the fault does not generate an error, simulation will be stopped when a timeout expires. Another feature of the simulator, useful for reliability consideration, is step-by-step execution: simulation is manually advanced by only a specified amount of clock cycles, thus it is possible to see in detail how the fault propagates through the whole architecture.

It is worth noting that, due to the impossibility to apply all possible input patterns, after a fault is injected, the simulation is resumed for a chosen and finite time, in which the fault should manifest itself before being declared as “not-observable”. This solution, adopted by

all fault injection tools, does not guarantee that the fault is not observable. It is thus very important to be able to closely inspect the effects of a "not-observable" fault, in order to have some insights on the real potential of that fault.

The presented mechanisms have been implemented to work in a semiautomatic way; two features are fundamental in reducing efforts for the simulation setup: a) golden model creation and b) fault list management. The automatic creation of the golden model allows the developer to avoid the manual specification of the structure of reference architecture (which is identical to the one under test) and the comparators among the signals of the two systems. As the fault list is concerned, it is possible to specify, through an XML file, the list of faults to be injected (each fault is specified by the variable to be changed, the mask to be applied for changing the variable value, and the clock cycle at which the injection has to be performed); the simulations, one for each fault, are, then, sequentially executed. Results are displayed at the end: for each simulation, the report shows if the fault has been activated (i.e., a difference was observed among the reference golden model and the one under test) and if it has been detected (in this case there are the details of the detection timestamp and the signal with the erroneous value).

We are currently using this approach to validate the results on the fault-error relation and the associated evaluated fault coverage obtained in the fault injection campaign reported in [13], according to the fault injection environment defined in [4]. That environment consists of an FPGA board used to emulate an instrumented model of the Leon2 processor [8], and of a Personal Computer that hosts the FPGA board and runs the software managing the fault injection experiments. We have performed a preliminary fault injection campaign on the same system composed by Leon processor connected to a memory through a bus; the connection was obtained by means of ReSP. Table 1 presents the results of this experimental session (the "HW Detected" column reports cases where an interrupt has been asserted because of an error detection or a timeout expiration).

Application	Register	Faults	No Error	Error		
				HW Detected	SW Detected	Not detected
ELPF	Reg. Bank	2000	1787	51	152	10
	Other Regs	1600	1366	12	216	6
Kalman	Reg. Bank	2000	1540	185	271	4
	PC Reg.	1600	1184	62	353	1
TOTAL		10800	8853	488	1408	51

Table 1. Fault injection: experimental results

5 Conclusions and future work

In this paper we present a framework for exploring the design space of MPSoCs; this framework supports the introduction of reliability techniques and the analysis, by means of fault injection, of the achieved reliability level.

The introduction of reliability properties from the early phases of the design flow allows the exploration of the solution space by considering fault detection and tolerance properties as metrics, to be added to performance and power consumption ones. The proposed fault injection platform is characterized by a high level of controllability of all aspects of the

fault injection process; it also provides a detailed report of fault effects, for a more precise analysis of the reliability level, and information on the faults that are not covered.

References

[1] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. MPARAM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing System 41*, pages pag. 169–182, 2005.

[2] C. Bolchini, L. Pomante, F. Salice, and D. Sciuto. Reliability Properties Assessment at System Level: A Co-Design Framework. *Journal of Electronic Testing: Theory and Applications*, 18(3):351–356, 2002.

[3] L. Cai and D. Gajski. Transaction level modeling: an overview. In *Proc. 1st Intl Conf. Hardware/Software Codesign and System Synthesis*, pages 19–24, 2003.

[4] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante. An FPGA-Based approach for speeding-up fault injection campaigns on safety-critical circuits. *Journal of Electronic Testing: Theory and Applications*, 18(3):261–271, 2002.

[5] CoWare. http://www.coware.com.

[6] B. Dave and N.K. Jha. COFTA: Hardware-software co-synthesis of heterogeneous distributed embedded system architectures for low overhead fault tolerance. *IEEE Trans. on Computers*, 48(4):417–441, 1999.

[7] F. Ziegler et al. Terrestrial cosmic rays and soft errors. *IBM Journal of Research and Development*, 40(1), 1996.

[8] J. Gaisler. *The LEON2 IEEE-1754 (SPARC V8) Processor*, 2003. http://www.gaisler.com.

[9] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[10] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. An efficient synchronization technique for multiprocessor systems on-chip. In *Int.l Workshop on MEmory performance: DEaling with Applications, systems and architecture*, 2005.

[11] P.G. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A System–Level Exploration Platform for Network Processors. *IEEE Design and Test of Computers*, pages 2–11, November–December 2002.

[12] Python. http://www.python.org.

[13] M. Rebaudengo, L. Sterpone, M. Violante, C. Bolchini, A. Miele, and D. Sciuto. Combined software and hardware techniques for the design of reliable ip processors. In *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, DFT*, pages 265–273, 2006.

[14] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. ArchC: A SystemC-Based Architecture Description Language. *sbac-pad*, 00:66–73, 2004.

[15] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W.L. Hung. Reliability-centric hardware/software co-design. In *Proc. Int. Conf. on Quality of Electronic Design*, pages 375–380, 2005.

[16] F. Vargas, E. Bezerra, L. Wulff, and D. Barros Jr. Optimizing HW/SW codesign towards reliability for critical-application systems. In *Proc. 7th Asian Test Symposium*, pages 52–57, 1998.