JOURNAL OF ELECTRONIC TESTING: Theory and Applications 18, 351–356, 2002

Reliability Properties Assessment at System Level: A Co-Design Framework

C. BOLCHINI, L. POMANTE, F. SALICE AND D. SCIUTO
Dipartimento Elettronica e Informazione, Politecnico di Milano, Italy
bolchini@elet.polimi.it
pomante@elet.polimi.it
salice@elet.polimi.it
sciuto@elet.polimi.it

Received June 14, 2001; Revised October 24, 2001

Editors: D. Nikolos, J.P. Hayes, M. Nicolaidis and C. Metra

Abstract. This paper introduces an enhanced hardware/software co-design framework allowing the designer to introduce hardware fault detection properties in the system under consideration. By considering reliability requirements at system level, within a hw/sw co-design flow, it is possible to evaluate overheads and benefits of different solutions. System specification, hardware and software concurrent fault detection design methodologies and hw/sw partitioning are the three key factors taken into account. The paper discusses these aspects providing a complete overview of the reliability co-design project.

Keywords: hardware fault detection, hw/sw co-design, partitioning, system specification

1. Introduction

The adoption of Hardware/Software Co-Design methodologies allows the designer to explore the system space for identifying the most promising solution according to the given goals and constraints. By acting at system level, the user can explore the design space in the early in the design process coping with the increasing complexity of the systems to be implemented. On the other hand, the introduction of reliability properties is postponed to lower abstraction levels where a set of traditional techniques is available for both combinational and sequential devices [1, 2]. Yet, the introduction of reliability requirements after the identification of the hardware and software components may have a strong impact on the overall system performance. Therefore, to avoid the violation of the system constraints only local modifications can be introduced. The proposed reliability co-design project aims at introducing hardware fault detection properties in the specification at system level, so that reliability issues can be taken into account to drive the binding of functionalities to either hardware or software.

The goal of this paper is to present the relevant aspects of the reliability co-design project, from the specification level to the system partitioning, by providing a set of design methodologies to deal with the system reliability constraints, defined at the specification level. The paper is organized as follows. Section 2 introduces the co-design framework, briefly presenting all the aspects of the environment. The design methodologies and the metrics to evaluate them are presented in Section 3. Section 4 then presents the proposed hw/sw partitioning methodology.

2. The Reliability Co-Design Flow

The proposed approach starts from a system specification. Let us define a *Section* as a subset of the system specification. The whole system is a particular section. A *Critical Section* is a section where the reliability

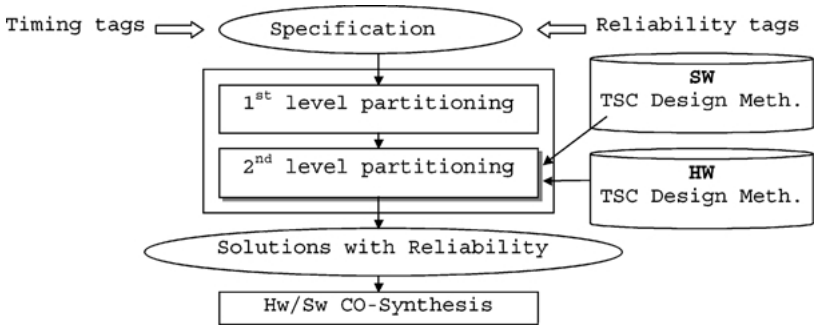

Fig. 1. The enhancement of a classical co-design flow to introduce reliability properties.

property is required. For each section it is necessary to identify the *Observable Results Set*, i.e. the set of data that the section outputs. If the considered section is a critical one then the observable result set is called *Critical Result Set* (or *Critical Results*). A *Reliable Section* is a critical section that propagates either error free critical results or faulty critical results associated with an error indication. For some methodologies the reliability properties are liable to a certain degree of uncertainty; these cases are identified as almost reliable.

The design of a critical section may require that the reliability property should be extended to other sections connected to it in order to provide reliable inputs to the critical section. Therefore, we define two properties associated with a critical section. The *Local Reliability* property of a critical section specifies that the reliability constraints involve only the related critical section. The *Global Reliability* property of a critical section specifies that the reliability constraints involve the related sections and recursively all the connected sections.

The reference architecture consists of the processor block (either general purpose or DSP), which executes software processes, memory and a set of co-processors (ASIC or FPGA) implementing hardware functionalities, if required. The adopted fault model is the *single functional failure*, where any number of physical faults causes a functional module to perform incorrectly. Software faults, i.e. bugs, are not considered in this work, depending on the correctness of the implementation of the code with respect to the specification; similarly, we do not guarantee the correctness of the device implementation with respect to the initial specification. In the presented methodology, software redundancy techniques are adopted to detect faults in the hardware, not in the software. The functional failure module thus includes the main processor, the co-processors, the main memory and the communication channels, the system bus and the dedicated channels for hw/hw module communication. Such a single failure model is based on a commonly adopted hypothesis: failure is detected before another module fails.

Fig. 1 shows the proposed reliable system design framework. A co-design flow taking into account reliability properties requires the addition of three main components to a standard co-design flow: (1) a system specification allowing the user to explicit reliability constraints while adopting a co-design approach; (2) the definition of device implementation methodologies fulfilling the designer requirements in terms of functionality, performance, costs and reliability; (3) a set of metrics for system partitioning, that include, in addition to costs and performance, also the evaluation of the co-design properties.

3. Metrics and Classes of Methodologies

The reliability project has investigated design methodologies for guaranteeing fault detection capabilities based on the adoption of redundancy strategies. Temporal, architectural, and information redundancy methodologies are usually adopted, preferring the latter two approaches since they provide a better coverage and a prompt detection of failures. These aspects are significant especially when working in a critical applications environment.

A suite of methodologies has been selected as a viable solution to provide the desired reliability, evaluating benefits (in terms of detection latency, fault coverage, system performance) and costs ([8]). Each of the provided solutions has been characterized with respect to these four significant parameters,

constituting the elements of the defined metrics to compare different possible solutions. The first parameter is the *detection latency*, that is the time between the instant a fault causes an observable error and the instant the error is detected; this aspect depends on the fact that the methodology monitors the results as soon as they are produced (no latency), or that other tasks are performed before the monitoring takes place (latency). The second parameter is the *fault coverage*, i.e. the percentage of detected faults with respect to the complete fault set. The third parameter is the *performance* degradation, which represents the overhead (i.e., execution time increase) caused by the components added to implement self-checking capabilities with respect to the original system. The last parameter is the *Cost* of a given solution, defined as the overhead with respect to the original system in terms of implementation cost and design cost. The goal is to weight all the elements contributing to the costs of the designed solutions. All these elements contribute to the metrics adopted for comparing different solutions to the reliability problem in the co-design flow.

The methodologies we analysed and developed can be classified first on the basis of the functionality performed and controlled (application execution, communication), then on the partitions (HW or SW) and finally on the Concurrent Error Detection (CED) techniques adopted for guaranteeing the reliability properties. Let us first consider reliability of the application execution.

The design approach considers as the basic element any functionality (or part of it) that the system must provide; with the term *nominal* we will denote the original functional system elements. The term *checking* will be used to identify the redundant elements designed to provide fault detection capabilities. The *checker* is the functional element for comparing the elaboration of the nominal and checking elements in order to detect a mismatching behavior due to failures. Each one of these three elements (nominal, checking and checker) can be independently implemented in hardware or in software, leading to several classes of solutions.

A preliminary analysis of the alternatives led to consider half of them (see Fig. 2), discarding those with an intrinsic inefficiency. For instance, we discarded the solution with the nominal architecture implemented in hardware and controlled by a software realization of the same functionality, since the checking architecture will not perform as fast as the nominal architecture thus providing an inefficient detection capability, either slowing

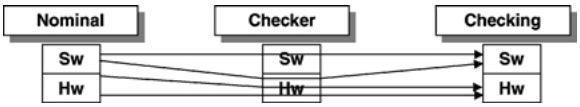

Fig. 2. Significant combinations of the hardware-software system parts.

down the original architecture or detecting faults with a high latency or at high costs for storing data between the two implementations.

By analysing the identified designs, which differ for the adopted concurrent error detection techniques, we selected those with the best trade-off in terms of cost and effectiveness. Four solutions have been considered:

- *Nominal SW, checker SW, and checking SW*, where the code dedicated to error detection tasks (both checking and checker) is executed either by the system processor or by a dedicated one. The selected approaches are *Self-Checking Sw* [12], *Assertions* [3], *Dual-Processor* [8], *and VLIW* [8, 10];
- *Nominal SW, checker HW, and checking SW*, where the checking functionality is implemented in software while the checker is hardware. Three different CED techniques have been identified: *Interface for Functional Redundancy Check* [5], *DMA Checker* [8], and *VLIW with Hw checker* [8];
- *Nominal SW, checker HW, and checking HW*, where hardware implementations of CED capabilities check critical sections implemented in software. The best solution identified is based on a *Dynamically configurable checker* [8].
- *Nominal HW, checker HW, and checking HW*, where the CED capabilities for modules implemented in hardware are also implemented in hardware. The selected techniques are: *Duplication* [8], *TSC scheduling* [6], and *TSC devices* [9].

Having defined the design methodologies belonging to the space of solutions analysed, it is necessary to provide reliability of the communications. More precisely, it is necessary to guarantee that any fault on communication lines is detected. Either hardware redundancy (lines duplication) or information redundancy (data encoding) can be adopted, independently of the communication channels being used. The checking functionality can be implemented in hardware or software. We explored the different possibilities (communications between procedures implemented in hardware, or between different kinds of procedures, hw/sw,

sw/sw, etc.) analysing costs and benefits. The best cost-effectiveness is obtained by keeping encoded data in memory, letting the CPU work only with unencoded data, while pairs of hw sections communicate by means of dedicated lines (duplicated, encoded, or a combination of both) [8].

4. System Partitioning

Once the system is specified and the critical sections are tagged, the first-level partitioning task identifies the solution space or part of it. This process is extremely complex and time consuming, due to the large number of possible alternatives and to the fact that, although heuristics and tuned estimation functions have been defined, it is the final co-simulation of the suggested system implementation that confirms the viability of the identified solution.

Given these premises, the reliability aspects add a significant number of parameters to the partitioning step for the selection of the final implementation, making this task too complex. In order to cope with the complexity of the partitioning step when reliability goals are also included, a two-level approach is here proposed. A first partitioning is performed which takes into account only the classical aspects and cost functions, meeting the usually stringent time constraints [7]. Given one solution, a second-level partitioning considers the additional reliability constraints, analyses the possible approaches, within the set of defined methodologies, which fulfil them and provides the solution that has the best trade-off (if it exists). Fig. 3 presents this two-level partitioning where a transparent background identifies the classical elements of a co-design flow, whereas the shaded areas point out the elements related to the reliability goals. It is worth noting that if the given solution has no possible allocation of critical sections fulfilling all reliability constraints either another solution has to be chosen from the design space or, if no solution exists fulfilling both classical and reliability constraints, the first level partitioning has to be repeated with a relaxed set of constraints and directives. The second-level partitioning, aimed at providing an acceptable allocation of hardware and software sections to the necessary tasks, must fulfil the initial, classical constraints considered by the first-level partitioning and has to meet the reliability constraints. These reliability constraints, explicitly stated in the system specification, refer to the strength of reliability required by each section, that may be *hard* or *soft*. A *hard reliability* requirement imposes that 100% fault coverage must be guaranteed for the section under consideration. A *soft reliability* requirement relaxes the constraint, to specify that any fault coverage is acceptable. This strength of the reliability enforcement is the main constraint

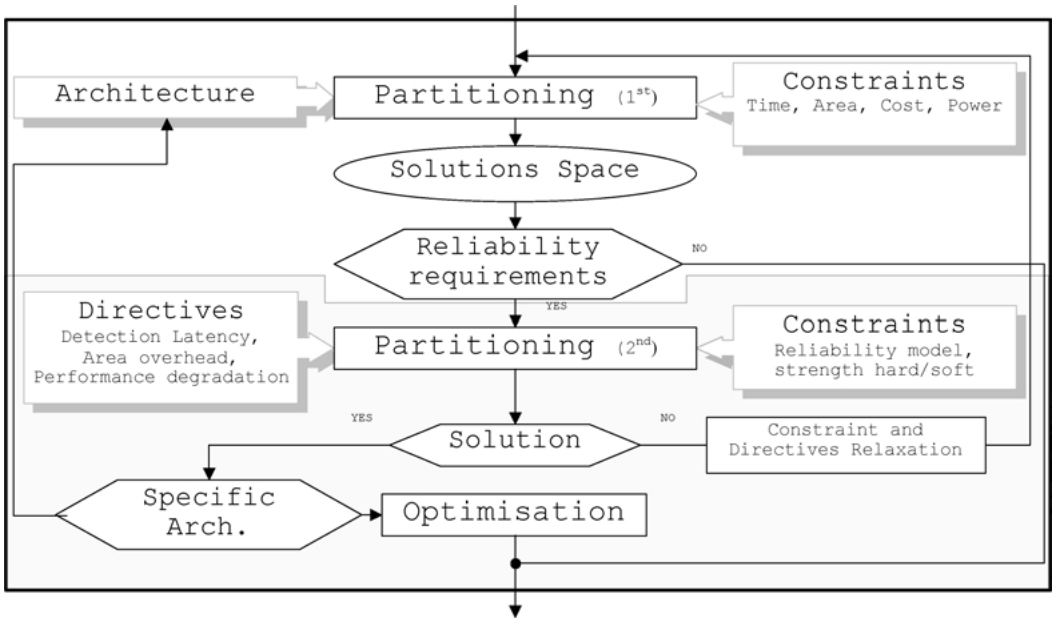

Fig. 3. The enhancement of a classical co-design partitioning to introduce reliability properties.

guiding the second-level partitioning; the parameters considered in the cost function for selecting among the available CED methodologies are the *fault coverage, detection latency, area overhead* and *performance degradation.*

All the constraints associated with these parameters are rather qualitative than quantitative, influenced by the functionality of the specific section and its implementation solution. The definition of the quantitative aspect is included in the cost function, and it is used when different solutions have to be compared.

In the second partitioning step, each task of the selected solution is associated with the reliability methodology that best fits the designer's expectations; the key element of the decisional process is constituted by the set of directives defined on each task (latency, area, performance degradation and constraint strength). However, the set of solutions locally identified does not typically represent the solution of the problem as a whole. For instance, if a task requires the *assertion* method (weak reliability constraint) while another task is associated with *IFRC* (strong reliability requirement), the first method is overlapped by the second one so that the first task is checked twice. Furthermore, the first method is covered by the second one, thus making the application of assertions a useless cost overhead. As a consequence of the methodological aspects concerning reliability, the partitioning problem consists of both (i) defining a criterion for the identification of the relation between the constrained task and the most suitable CED method and (ii) optimising the result produced by the assignment criteria with respect to the global solution. The optimisation task has to be applied concurrently with the method identification since the effects of dominance reduce the solution space.

4.1. Reliability Model Identification

The criterion for the identification of the most suitable CED method uses the set of directives and constraints that have been applied to the system specification. Fault coverage, detection latency, area overhead and performance degradation can identify a qualitative or quantitative description of the system components properties for CED characterizations in relation with a designer specification. In particular, for all of them it is possible to identify a correct evaluation of the parameter (e.g., 0 performance degradation)—hard—or a qualitative estimation (minimum, medium or maximum)—soft—. A *fuzzy* tag identifies the required effort for the identification of anomalies during the operational time of the device (e.g. in a system section considered marginally critical). Conversely, a *crisp* tag (100% fault coverage, 0 detection latency) represents a hard system constraint that has to be enforced at any cost. Note that, for soft requirements, a *maximum* requirement includes methodologies belonging to the *medium* or *minimum* partitions; and a *medium* requirement includes *minimum.*

By taking into account the above expressed issues and considering the intrinsic properties of the methodologies that can be applied, it is reasonable to observe that crisp tags force a partition on the methodologies set. Furthermore, since the applicability of a methodology to a specific task depends on its hardware/software characteristic, a further partition is induced. In particular, 100% fault coverage induces the partitions $\pi_{\texttt{hard\_fc}}$ and $\pi_{\texttt{soft\_fc}}$, 0 detection latency induces the partitions $\pi_{\texttt{hard\_dl}}$ and $\pi_{\texttt{soft\_dl}}$ while, 0 performance degradation induces the partition $\pi_{\texttt{hard\_pd}}$; $\pi_{\texttt{soft\_pd}}$. In turn, such partitions are subdivided in relation with hardware and software aspects.

The analysis of the different properties of the selected design methodologies has led to the definition of partitions related to the hardware/software (πhw and πsw), the three parameters for classifying design methodologies (i.e. $\pi\,\texttt{sw}_{\texttt{fc}}$ or $\pi\,\texttt{hw}_{\texttt{fc}}$ for fault coverage) and, finally, the hardness and softness of the tag (i.e. $\pi_{\texttt{hard\_fc}}$ and $\pi_{\texttt{soft\_fc}}$). As a result, a complete partitions have been identified, with respect to the selected suite of CED design methodologies [8].

The second level partitioning takes into account the *hard* parameters first to select suitable CED techniques, and uses the *soft* parameters to select among them. More precisely, for each critical task, on the basis of its allocation in hardware or in software, the π partitions fulfilling the *hard/soft* requirements are selected and the intersections of their subsets (the CED techniques) provide the set of suitable techniques. Furthermore, *hard* requirements are fully met by all methodologies in the $\pi_{\texttt{hard}}$ so all methodologies belonging to the partition are selected. On the other hand, *soft* requirements cover a spectrum of possibilities with respect to the solutions: more precisely, methodologies belonging to the $\pi_{\texttt{soft}}$ can achieve either a medium or high degradation. As a consequence, a first screening of the methodologies in the $\pi_{\texttt{soft}}$ partition can reject some solutions on the basis of the specified *soft* requirement (*medium, minimum, maximum*).

The global solution determining for each task the CED technique actually adopted is pursued by means of a process of solution extraction and simulation, to verify that the constraints of the first partitioning are still met. This process takes into account the fact that there are techniques with a global effect which prevail over those with a local impact. As an optimisation policy, the final solution does not include overlapped methods in order to achieve a significant efficiency.

5. Conclusion

This paper introduced different aspects of a hw/sw co-design flow supporting the introduction of concurrent hardware fault detection properties in the system being designed. The user, by means of ad-hoc tags and directives those sections (tasks) of the system, specifies the need to be reliable so that they either perform correctly or an error signal is raised. A set of hardware and software design methodologies have been studied, both traditional and innovative, to provide the desired fault detection capabilities. A library of possible approaches has been identified so that, during the partitioning step, the allocation of the nominal sections to hardware or software is followed by the allocation of the reliability sections in the fulfilment of the user's constraints and fault detection requirement. This design flow has been implemented in the TOSCA co-design environment, considering OCCAM II and SystemC as specification languages. Different real-world embedded systems examples have been analysed, to prove the effectiveness of the proposed approach [4].

References

1. N.K. Jha and Wang, "Design and Synthesis of Self-Checking VLSI Circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 6, pp. 879–887, June 1993.
2. D.P. Siewiorek and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.
3. Z. Alkhalifa, V.S.S. Nair, and J.A. Abraham, "Design and Evaluation of System-Level Checks for On-Line Control Flow Error Detection," *IEEE Trans. Parallel and Distributed System*, vol. 10, no. 6, pp. 627–641, June 1999.
4. C. Bolchini, L. Pomante, F. Salice, and D. Sciuto "Reliability Properties Assessment At System Level: A Co-Design Framework," Tech. Rep. 2001-85, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 2001.
5. C. Bolchini, F. Salice, and D. Sciuto, "Designing Reliable Embedded Systems Based on 32 bit Microprocessors," in *Proc. IEEE 7th Int. On-Line Testing Workshop*, 2001, p. 137.
6. C. Wah and A. Orailoglu, "High-Level Synthesis of Gracefully Degradable ASICs, "in *Proc. European Design and Test Conference, 1996 (ED&TC'96)*, pp. 50–54.
7. G. De Michell and R.K. Gupta, "Hardware/Software Co-Design," in *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349–365, March 1997.
8. L. Pomante, "System Level Concurrent Error Detection," Tech. Rep. RT-00003, Cefriel-Politecnico di Milano, Milano.
9. D.K. Pradhan, (Ed.), *Fault Tolerant computing, Theory and Techniques*, vol. 1. Englewood Cliffs, NJ, U.S.: Prentice Hall, 1986.
10. F. Rashid, K.K. Saluja, and P. Ramanathan, "Fault Tolerance Through Re-Execution in Multiscalar Architecture," in *Proc. Int. Conference on Dependable Systems and Networks (DNS 2000)*, 2000, pp. 482–491.
11. F.F. Sellers, M.Y. Hsiao, and L.W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill: New York, 1986.
12. H. Wasserman and M. Blum, "Software Reliability via Run-Time Result Checking," *Journal of the ACM*, vol. 44, no. 6, pp. 826–849, November 1997.