Error Detection Techniques Applicable in an Architecture Framework and Design Methodology for Autonomic SoCs*

Abdelmajid Bouajila[1], Andreas Bernauer[2], Andreas Herkersdorf[1], Wolfgang Rosenstiel[2,3], Oliver Bringmann[3], and Walter Stechele[1]

[1] Technical University of Munich, Institute for Integrated Systems, Germany
[2] University of Tuebingen, Department of Computer Engineering, Germany
[3] FZI, Microelectronic System Design, Karlsruhe, Germany

Abstract. This work-in-progress paper surveys error detection techniques for transient, timing, permanent and logical errors in system-on-chip (SoC) design and discusses their applicability in the design of monitors for our Autonomic SoC architecture framework. These monitors will be needed to deliver necessary signals to achieve fault-tolerance, self-healing and self-calibration in our Autonomic SoC architecture. The framework combines the monitors with a well-tailored design methodology that explores how the Autonomic SoC (ASoC) can cope with malfunctioning subcomponents.

1 Introduction

CMOS technology evolution leads to ever complex integrated circuits with nanometer scale transistor devices and ever lower supply voltages. These devices operate on ever smaller charges. Therefore, future integrated circuits will become more sensitive to statistical manufacturing/environmental variations and external radiation causing so-called soft-errors. Overall, these trends result in a severe reliability challenge for future ICs that must be tackled in addition to the already well-known complexity challenges. The conservative worst case design and test approach will no longer be feasible and should be replaced by new design methods. Avizienis [1] suggested integrating biology-inspired concepts into the IC design process as a promising alternative to today's design flow with the objective to obtain higher reliability while still meeting area/performance/power requirements. Section 2 of the paper presents an Autonomic SoC (ASoC) architecture framework and design method which addresses and optimizes all of the above mentioned requirements. Section 3 surveys existing error detection techniques that may be used in our Autonomic SoC. Section 4 discusses implications on the ASoC design method and tools before section 5 closes with some conclusions.

* This work is funded by DFG within the priority program 1183 "Organic Computing".

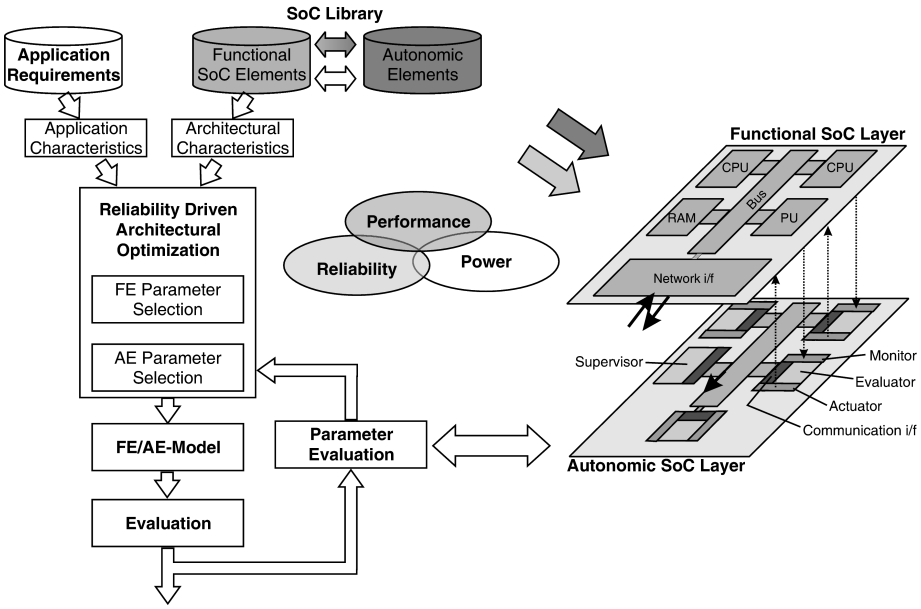

Fig. 1. Autonomic SoC design method and architecture [2]

2 Autonomic SoC Architecture and design method

Figure 1 [2] shows the proposed ASoC architecture platform. The ASoC is split into two logical layers: The functional layer contains the intellectual property (IP) components or Functional Elements (FEs), e.g. general purpose CPUs and memories, as in a conventional, non-autonomic design. The autonomic layer consists of interconnected Autonomic Elements (AEs), which in analogy to the IP library of the functional layer shall eventually represent an autonomic IP library (AE_lib). At this point in time, it is not known yet whether there will be an AE for each FE, or whether there will be one AE supporting a class of FEs.

Each AE contains a monitor or observer section, an evaluator and an actuator. The monitor senses signals or states from the associated FE. The evaluator merges and processes the locally obtained information originating from other AEs and/or memorized knowledge. The actuator executes a possibly necessary action on the local FE. The combined evaluator and actuator can also be considered as a controller. Hence, our two-layer Autonomic SoC architecture platform can be viewed as a distributed (decentralized) observer-controller architecture. AEs and FEs form closed control loops which can autonomously alter the behavior or availability of resources on the functional layer. Control over clock and supply voltage of redundant macros can provision additional processing performance or replace on-the-fly a faulty macro with a “cool” stand-by alternative.

Although organic enabling of next generation standard IC and ASIC devices represents a major conceptual shift in IC design, the proposed ASoC platform represents

a natural evolution of today's SoCs. In fact, we advocate to reuse cores as they are and to augment them with corresponding AEs.

In order to study the feasibility of our ASoC framework, we already started looking in-depth into how to design such SoCs. We adopted a bottom-up approach in which we design autonomic building blocks and connect them to build an Autonomic SoC. The ultimate objective is to understand how to form an Autonomic IP library and, thus, how to design Autonomic SoCs in a systematic and well-established top-down design flow.

3 Existent CPU concurrent error detection techniques

The human body needs to sense the state of its different organs, e.g. pain or temperature, to let the immune system handle the problem or to try to ask for external help, e.g. medicines [1]. In analogy to the human body, the SoC needs to detect errors for example to allow a CPU to re-execute an instruction or to ask for a replacement CPU.

There are three main concurrent error detection techniques: hardware redundancy, information redundancy and time redundancy. In our survey, the efficiency of each technique is measured by 1) how many different types of errors can be detected (so we need to define a fault model and then to evaluate the fault coverage), 2) how much overhead (in terms of area, performance and power) the concurrent error detection technique induces, and 3) how feasible IP-reuse is, i.e. is it possible to achieve error detection in an already existent CPU by adding a separate monitor?

In [3], an (extended) fault model classification is presented. In fault-tolerant integrated circuits literature, faults are usually classified as permanent [4, 5], timing [6], transient [7] or design (logic) errors. The most widely used fault model is the single error fault, in which different errors don't occur simultaneously. The fault coverage [3] of a detection technique is given for a specific fault model. Fault coverage is either given by analytical (formal) methods or by simulation.

A. Hardware redundancy techniques: Hardware redundancy has good fault coverage (transient, timing and permanent errors). However, the area and power overheads are big. In the particular case of duplication, the monitor will be the duplicated circuit and the comparator. In spite of its big overheads, we could use this technique in our ASoC project because the percentage of logic parts in modern SoCs is less than 20%.

B. Information redundancy techniques: There are two different approaches using information redundancy. The first approach synthesizes HDL descriptions [4] to generate so-called path fault-secure circuits. The second approach tries to build self-checking arithmetic operators to achieve the fault-secure property [5]. The main drawback of the first approach towards our ASoC framework is the difficulty in separating between the functional and autonomic layers. Self-checking designs give fault-secure arithmetic operators for stuck-at faults (permanent faults) with low area overhead but their use requires redesigning existing IP-libraries.

C. Time redundancy technique: The time redundancy technique was proposed to detect transient errors. Transient errors can be modeled by SEU (Single Event Upset) and SET (Single Event Transient) [8].

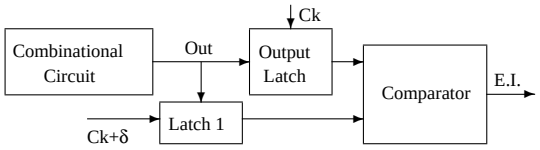

Fig. 2. Time redundancy technique [7]

The idea of Nicolaidis [7] is to benefit from the fact that a transient error will occur only for a short duration. Hence, if we sample a primary output at two successive instants separated by a duration larger than that of the SET pulse, we will be able—in theory—to detect all SETs. Figure 2 presents this scheme.

Simulations [9] on adders and multipliers showed that this scheme detects around 99% of SETs (an SET can escape from detection because of reconvergent paths). It also detects SEUs and timing errors. The area overhead depends on the circuit area per output parameter. Razor, a similar time redundancy technique for detection and correction of timing errors was suggested in [6].

These time redundancy techniques are very interesting for our ASoC framework and could be integrated in a systematic way to protect circuits against transient and timing errors. Also, the separation between the functional and autonomic layer is quite simple; for instance the monitor for Nicolaidis scheme (Fig. 2) will include the extra latch and the comparator.

D. Combination of Hardware and Time Redundancy: The Dynamic Implementation Verification Architecture (DIVA) [10] incorporates a concurrent error detection technique to protect CPUs. We can classify it either as a time redundancy and/or a hardware redundancy technique. The baseline DIVA architecture (Fig. 3a) consists of a (complex) processor without its commit stage (called DIVA core) followed by a checker processor (which consists of CHKcomp and CHKcomm pipelines, called DIVA checker) and the commit stage.

The DIVA checker checks every instruction by investigating in parallel (Fig. 3a) the operands (re-reading them)and the computation by re-executing the instruction. In case of an error, the DIVA checker, which is assumed to be simple and reliable, will fix the instruction, flush the DIVA core and restart it at the next program counter location. Physical implementation of a DIVA checker has an overhead of about 6% for area and 5% for power [11].

We believe that in very deep sub-micron technologies a checker which is reliable enough is difficult to achieve and could also result in a large area overhead. We suggest a modified version of DIVA in which both the DIVA core and checker re-execute an errant instruction, so that the checker no longer needs to be reliable.

The modified DIVA (Fig. 3b) only checks the CPU computation and protects the DIVA core/memory interface and the register file with error correcting codes (ECC). Therefore, the DIVA checker no longer needs to access the register file and data cache, eliminating structural hazards between the DIVA core and checker. The fault coverage of both DIVA versions includes transient, timing, permanent and logic errors. In both DIVA versions, it is mandatory that the error rate is bounded not to decrease perfor-

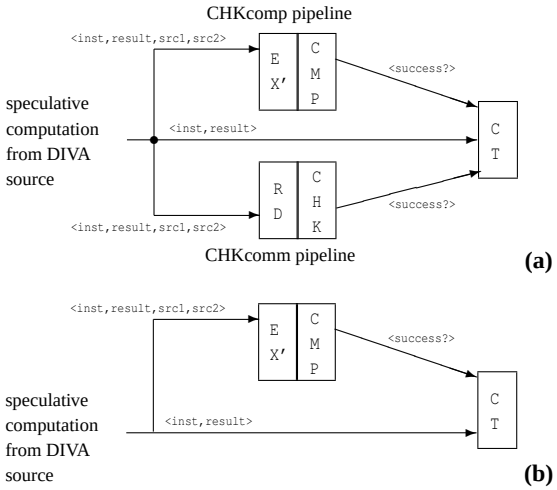

Fig. 3. DIVA architecture. **(a)** Baseline DIVA architecture [10], **(b)** Modified DIVA architecture

mance. DIVA cannot be inserted in a systematic way to protect existent CPUs because there are no standards in designing CPUs. Hence, a designer should study the CPU architecture and implementation and then separate the commit stage to be able to insert the DIVA checker. Nevertheless, we should mention that the separation between the functional layer and the autonomic layer is clear (the DIVA checker is the monitor). Also, DIVA enables us to re-use existent CPUs by identifying their commit stage.

We decided to use the modified version of DIVA to build autonomic CPUs because it allows us to separate functional and autonomic layers, permits IP-reuse and has the best fault coverage when compared to other detection techniques: it detects transient, timing, permanent and logic errors with a fault coverage close to 100%. The drawbacks of the other techniques are either big overheads (duplication, path fault-secure circuits), limited fault coverage (transient and timing-error detection oriented techniques (Nicolaidis and Razor), or restrictions to just stuck-at detection (Self-checking designs)).

4 Design Methodology and architecture for ASoC

A successful design of Autonomic SoCs needs a well-tailored design methodology that explores the effect of the AEs to cope with malfunctioning subcomponents. Our ASoC design methodology (Fig. 1) follows the established platform-based design approach, where a predefined platform consisting of a set of architectural templates is optimized for a given application with respect to several design constraints like area, performance and power consumption. In the context of this paper, the traditional process is extended by adding the autonomic layer to the platform model and considering reliability as an additional parameter. Therefore, the evaluation process now has to deal with the effects of the AEs—which include algorithms to support self-optimization—as well as

with the AEs' relationship to system reliability. The evaluation process results in an optimized set of FE/AE parameters including provision of an a priori knowledge for the evaluators at the autonomic layer.

The design methodology will decide where and how many of the aforementioned error detection units the ASoC will need to meet the application's reliability requirements. The error detection units will be part of the AE_lib.

As the resulting ASoC will be able to change parts of its design during run time it will need design time information. In particular, the ASoC will need a priori knowledge about the application's behavior when an error occurs and, more importantly, about how the system has to self-modify to handle the error. The design methodology will gather this knowledge by injecting errors according to the error model into the application and architecture model and analyzing the consequences. The knowledge will be implemented distributed over the AEs in a self-organizing algorithm.

However, it won't be feasible to explore for all possible combinations of errors. For the explored error situations the self-organizing algorithm can react as given by the a priori knowledge. For the unexplored error situations it must be able to derive applicable measures but still meet the application constraints like temperature, timing and power consumption. The XCS classifier system presented by Wilson et al. [12] is capable to do this.

Registers attached to the error detection units will count the detected and corrected errors within a sliding time interval. When the counter exceeds some threshold, the self-organizing algorithm will take the necessary measures to correct for the error. It is also possible to provide a way to make the reliability information accessible to the application. With this, not only the hardware but also the application can adapt to varying reliabilities of some components, e.g. by rescheduling tasks to some more reliable CPU.

5 Conclusions and Outlook

This paper presented an Autonomic SoC architecture framework and design method. We are going to build an Autonomic CPU based on the LEON processor [13]. This will help us to evaluate the design effort, overheads and the gain of reliability achieved by our method. Later, we will build autonomic memory and autonomic interconnect; the ultimate objective is to get an ASoC architecture and design method integrating biology-inspired concepts.

References

1. A. Avizienis, Toward Systematic Design of Fault-Tolerant Systems, *IEEE Computer* **30**(4), 51–58 (1997).
2. G. Lipsa, A. Herkersdorf, W. Rosenstiel, O. Bringmann and W. Stechele, Towards a Framework and a Design Methodology for Autonomic SoC, in: 2nd ICAC (2005).

3. A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Trans. on Dependable and Secure Computing* **1**(1) (2004).
4. N. Touba and E. McCluskey, Logic Synthesis of Multilevel Circuits with Concurrent Error Detection, *IEEE Trans. CAD* **16**(7), 783–789 (1997).
5. M. Nicolaidis, Efficient Implementations of Self-Checking Adders and ALUs, in: Proc. 23rd Intl. Symp. Fault-Tolerant Computing, pp. 586–595 (1993).
6. D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, T. Mudge and K. Flautner, Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation, in: Proc. 36th Intl. Symp. Microarch., pp. 7–18 (2003).
7. M. Nicolaidis, Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies, in: Proc. 17th IEEE VLSI Test Symposium, pp. 86–94 (1999).
8. S. Mitra, N. Seifert, M. Zhang, Q. Shi and K. S. Kim, Robust System Design with Built-In Soft-Error Resilience, *IEEE Computer* **38**(2), 43–52 (2005).
9. L. Anghel and M. Nicolaidis, Cost Reduction and Evaluation of a Temporary Faults Detecting Technique, in: Proc. DATE, pp. 591–598 (2000).
10. T. M. Austin, DIVA: A Dynamic Approach to Microprocessor Design, *Journal of Instruction-Level Parallelism* **2**, 1–6 (2000).
11. C. Weaver and T. Austin, A Fault Tolerant Approach to Microprocessor Design, in: Proc. Intl. Conf. Dependable Systems and Networks, pp. 411–420 (2001).
12. S. W. Wilson, Classifier Fitness Based on Accuracy, *Evolutionary Computation* **3**(2), 149–175 (1995).
13. LEON VHDL code is available at `www.gaisler.com`.