Towards a Framework and a Design Methodology for Autonomic SoC

Gabriel Lipsa, FZI, Microelectronic System Design, Karlsruhe, Germany
Andreas Herkersdorf, Technical University of Munich, Institute for Integrated Systems, Germany
Wolfgang Rosenstiel, University of Tuebingen, Department of Computer Engineering, Germany
Oliver Bringmann, FZI, Microelectronic System Design, Karlsruhe, Germany
Walter Stechele, Technical University of Munich, Institute for Integrated Systems, Germany

Abstract

The transition from microelectronics to nanoelectronics reaches physical limits and results in a paradigm shift in the design and fabrication of electronic circuits. The conservative worst-case-approach is no longer feasible and has to be replaced by new design methods. These new design methods and tools have to guarantee reliable and robust systems in spite of unsafe and faulty functions, due to fabrication faults, soft errors and design errors.
This paper proposes autonomic or organic computing principles to be applied to hardware design methods for future SoC solutions. Incorporating self-calibration, fault tolerance or even self-healing concepts into integrated circuits systems represents a major conceptual shift, which requires new design processes and tools. In the future, guarantee of functional correctness at the chip level will include self-configuration of adaptable components and flexible interfaces supporting a flexible component composition within complex SoC systems. Herewith, we will present novel concepts how autonomic computing principles can be applied to ASoC design.

1   Introduction

The 2003 ITRS Roadmap [9] projects micro- and nanoelectronic integrated CMOS circuits to witness a continued capacity growth rate corresponding to doubling transistor count every two to three years ("Moore's Law"). These enormous chip capacities enable systems of ever increasing functional complexity and heterogeneity to be integrated on a single chip (SoC - System on Chip). Systems, which few years ago did consist of multiple components on multiple boards, now can be assembled from IP (Intellectual Property) macro library elements and integrated on a single die. The availability of such high chip capacities has also changed the key challenges of integrated circuit (IC) design. Today and in the future the primary driver will no longer be how to integrate even more transistors on a single chip, but how to develop such complex ICs with affordable cost and within reasonable time frames. Thus, the capacity problem, which dominated semiconductor scaling for decades, has transformed into a complexity problem.

This paper suggest that such a significant shift in IC design challenge demands a new conceptual approach in the IC design method to overcome the productivity bottleneck. We propose autonomic system properties – self-configuring, self-administrating, self-healing and self-protection – to be incorporated into future IC designs and be supported by corresponding tools. Section 2 briefly discusses related work in this field. Section 3 sketches properties and scenarios of autonomic ICs. It also introduces a high level overview on the autonomic SoC (ASoC) framework. Section 4 sketches the architecture of ASoC that we have in mind and the research direction that needs to be followed. Section 5 describes the design methodology for obtaining the ASoC and the research direction necessary for obtaining this methodology. Section 6 presents conclusions and future possibilities to expand this framework.

2   Previous Work

In 2001, IBM declared "Autonomic Computing" to be the most important challenge for information technology in the future [8], [10]. The German Information Technology and Informatics Society (ITG/GI) identified "Organic Computing" as a key technology for computer and system architecture of 2010 [18] and illustrates various application scenarios. In [12] requirements for new design methods and tools are described to guarantee reliable and robust systems in spite of unsafe and faulty functions on the lowest process levels. Self-calibrating, fault-tolerant or even self-healing systems

require totally new design processes and new design tools.

The *CARUSO* project [2] proposed autonomic self-x functions to be provided by the middleware layer of a multi-threaded CPU system. The *Diva* [1] and *Razor* [6] projects focus on hardware techniques in processor micro-architecture design. *Diva* introduced the concept of dynamic verification to reduce the burden of correctness in microprocessor design. *Razor* tunes the supply voltage while monitoring the bit error rate in the execution pipelines during operation. S. Mitra et al., Stanford, introduced an FPGA-based self-repair concept where faulty parts of a design are replaced on-the-fly by originally unused logic and routing resources [11].

Our contribution is complementary to *CARUSO* because we focus on autonomic principles, which are entirely embedded in the hardware layer. The techniques used in *Diva* and *Razor* can be used for applying autonomic principles to processing units.

3 Overview of ASoC Framework

Our focus is on the chip level hardware layer of SoCs (Systems on Chip) solutions, which form the underpinning base technology for all IT infrastructure and computing equipment. As described in section 1, future SoCs will witness a continued exponential increase in transistor capacity resulting in a complexity and reliability problem. Therefore, we propose to rededicate a fraction of the abundant transistor capacity of future SoCs to implement organic computing principles for the sake of higher fault tolerance, performance, power efficiency, easier system diagnosis and the capability to autonomously adapt to changing environmental conditions – be it either externally imposed workloads or temperature variations. Such a conceptual shift in the approach to IC design requires a fresh and holistic view on the implications for SoC architecture platforms, the SoC design method and corresponding EDA design tools. As organic computing attempts to conquer complexity and reliability with additional complexity, appropriate tools support during the entire design process is of critical importance for creating trust and acceptance in the system developer community.

We need to investigate how a self-organizing, autonomic SoC architecture platform looks like and what kind of design method and tools are required to support ASoCs. By a self-organizing SoC, we understand an integrated system that will continuously try to find by itself, without external intervention, the most suitable configuration for keeping the system in reliable, fault-free functional operation while ensuring best possible performance.

Translated into the world of semiconductor IC systems, the future may look as follows: with increasing, externally imposed workloads, the clock frequency and supply voltage of individual processor cores are increased to elevate processing performance. At the same time, critical transactions on on-chip interconnect buses, for example between processors and the memory subsystem, are prioritized and the bus bandwidth of less critical transactions is reduced. Redundant building blocks, being deactivated under normal operating conditions, are activated on demand to increase system performance. In return, secondary, low priority functions or external chip interfaces may be deactivated to reduce power consumption. System monitoring and self-test units trigger diagnosis traces to be analyzed in real-time and/or written to memory in cases where functional blocks or system buses show suspicious behaviour. Such analysis can lead to the de- or reactivation of entire IC portions.

There do exist already a number of techniques, which can be applied and further developed in the context of autonomous integrates systems. For example:

- Dynamic Voltage and Frequency Scaling (DVFS) [4] does control already processor clock and operation voltage.
- Dynamic reconfigurable processing units [16] adapt their behaviours during operation to external conditions.
- Built-in-self-test concepts [3] validate circuitry at system start up and partially also during system operation.
- On-chip debugging aids like ChipScope [19], RISCWatch [7], and bus monitors [17] grant partial access to signals and system state information which aren't otherwise observable through regular chip I/Os.

There are mechanisms and strategies to support fault tolerant behaviours of complex systems [12].

Self-organization also has to deal with the phenomenon of graceful degradation of SoCs, or said in other words, to guarantee minimum SoC functionality and performance in the event of entirely or partially failing system components for which no redundant replacement is available. In contrast to degrading SoCs, self-healing concepts attempt to replace a faulty processing unit with an equivalent counter part which will adopt the functionality of the failing element. The replacement unit can either be an idle stand-by element, or a processing unit that performs other tasks prior to the error occurrence.

The self-healing concept does not just mean to fix an error, but also to prevent errors in cases where the system risks getting into a critical state (e.g.

performance wise due to component overload, or temperature wise due to excessive power consumption). The problem is that system clock frequencies in the hundreds of MHz and transaction times of few nanoseconds do not allow for reactive problem investigation in the event of a failure but require proactive problem prediction before a component actually fails. This implies that the ASoC must be able to supervise the behaviour of its constitutional components and build up fallback scenarios, which are activated under certain trigger conditions before system failure. Fallback solutions can result in deactivating secondary functional units for saving power. The fallback solution just has to be good enough for respecting the constraints and providing a reasonable quality of service. Once a fallback solution has been deployed, the self-organization process will again try to improve performance and eventually switch back to the original system configuration.

The ASoC architecture platform related aspects are closely interlocked with the ASoC design methodology (similar to what is best practise in processor micro-architecture and compiler co-design).

Academic and industrial research institutions have just started to investigate what autonomic behaviour means at different levels of system abstraction. In the particular context of SoC design, specific research directions for building a framework in ASoC are required:

- The investigation and development of a new SoC design method, capable of dealing with graceful degradation and redundancy in distributed integrated systems. This includes the ability to model and optimize structural changes within the SoC architecture in the event of failure or suboptimal performance.
- The investigation and development of a corresponding ASoC architecture platform, which extends the state-of-the-art IP library design approach towards self-organization and exploits the capabilities of the new design method and tools. We believe that the ASoC design method and ASoC architecture platform must be developed in a seamlessly interlocked manner (similar to microprocessor architecture and compiler co-design) in order to be effective.
- The investigation and development of new on-chip component supervision and validation techniques for individual and interdependent functional macros, which could only partially be verified before manufacturing. Since functional test coverage of multi-hundred million transistors macros cannot be exhaustive, we need to search

for efficient means to evaluate the dynamic behavior of SoC IP macros in real-time.
- The investigation and development of concepts for dynamic and coupled power-performance management in SoCs.
- The investigation and development of approaches for flexible and dynamic hardware-software (HW-SW) repartitioning. An equivalent software process shall replace defective hardware, or dynamically loaded hardware configurations shall replace a low performing software process.

4 Autonomic SoC Architecture

Fig. 1 shows the proposed ASoC architecture platform. The ASoC is split into two logical layers: The functional layer contains the IP component or Functional Elements (FEs). FEs are either general purpose CPUs, memories, on-chip busses, special purpose processing units (PUs) or system and network interfaces as in a conventional, non-autonomic design. The autonomic layer consists of Autonomic Elements (AEs) and an interconnect structure among the AEs. In analogy to the functional layer IP library, the AEs shall eventually represent an autonomic IP library (AE_lib). At this point in time, it is not known yet whether there will be a one-to-one correspondence between each functional IP component and its autonomic counterpart, or whether there will be one autonomic IP element supporting a class of functional IP components. Furthermore, there may but need not be an AE per functional macro.
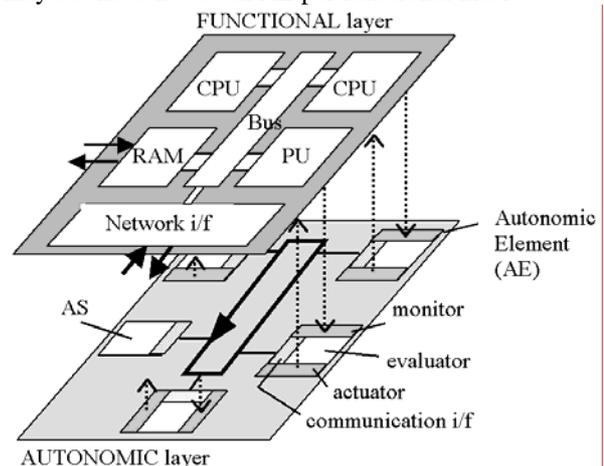

Fig. 1 Two-layer autonomic SoC platform

There will also be a special type of AE, an autonomic supervisor (AS), which has no counter part on the functional layer. This AS element acts as a central controller in the otherwise fully distributed AE architecture. The AS is a preferred candidate for monitoring the correct operation of and interaction between other AEs. Thus, the self-healing aspects of

ASoC are extended to the autonomic layer itself as AEs may also fail. Alternative to a centralized AS implementation, a distributed realization of the AS functionality spread across a set of AEs would also be conceivable. In order to keep complexity bounded and for the sake of a clean functional partitioning, we will start with investigating a centralized AS. The AE interconnect structure within the autonomic layer – Autonomic Interconnect Structure (AIConS) – may but need not be identical to the on-chip interconnect at the functional layer. An interconnect scheme with a different topology (e.g. switch or ring) might better serve the AE-specific information exchange.

Each AE contains a monitor or observer section, which senses signal or state information from the associated FE, an evaluator, which merges and processes the locally obtained information with state from other AEs and/or memorized local knowledge, and an actuator, which 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 distributed (decentralized) observer-controller architecture.

AEs and FEs form closed control loops, which can autonomously alter the behaviour or availability of resources on the functional layer. Control over clock and supply voltage of redundant macros can provide additional processing performance or replace on-the-fly a faulty macro with a "cool" stand-by alternative.

Likewise, temporarily switching off external chip interfaces when not needed, or narrowing down wide on-chip communication busses under low load conditions, contributes to system power savings.

The capability to administer and store state information locally in the AEs (e.g., to represent a sliding window history trace on pertinent observations on the functional macro) is essential for establishing a controlled emergent behaviour of the global ASoC system. Emergent system behaviour has the inherently attractive potential to make the system react in ways, which haven't been considered or architected during the design process, a property that is tightly associated with truly autonomous systems. In general, emergent behaviour may also imply undesired and/or chaotic reactions. In nature, such tendencies are naturally eliminated (only evolutions which are in harmony with their environment survive). In technical systems, precautions against undesired emergence have to restrict alternatives leading to controlled forms of emergence only. In our ASoC platform, AEs take decisions based on the stored status plus the communicated information from other AEs. Thus, the local state in an AE is not sufficient to predict the local action taken and the global system behaviour. Since only local actions

which are "in harmony" with the local functional macro are allowed, this approach implicitly guarantees controlled emergence.

The AE control loop works entirely in the hardware domain (no software in the loop) to ensure control loop reaction times of few system clock cycles, i.e., nanoseconds. Autonomic software processes, e.g. as described in [2], [8], should make use of information gathered by AEs. While AEs operate autonomously in a decentralized manner, they may be initiated and dynamically configured from a central system control point.

We thought of AE elements for the CPUs, memory and network/system interface. The AE of the processing units will be able to control the voltage and frequency, will switch the state of the processing unit from stand-by mode to active in case some other CPU fails or will shut down the CPU in case of failure. For the memories we apply two concepts: local self-healing memories and distributed self-healing memories. For the first ones, the memory AE maintains for a restricted number of erroneous memory locations an address-transformation unit that maps "erroneous addresses" into a memory region, which is not known to the regular memory map and has been verified to function properly. The distributed self-healing memories approach assumes cooperative functional masters on the on-chip interconnect bus. The Memory AE identifies erroneous memory locations with the above procedure and broadcasts the faulty address locations via the AE interconnect structure. The network AEs can be used as an alert mechanism for other SoC functional units to prepare for increased workload.

Very important will be the AIConS (Autonomic Interconnect Structure), which will link all the AEs with the central AS. Today, high-capacity on-chip busses are the ubiquitous standard for functional layer interconnects. NoC (Network on Chip) architectures with higher capacity and the capability to differentiate between local and (chip) global transactions are a potential successor technology for conventional busses. Three alternatives are possible for the AIConS: shared bus, multi-port switch and shared ring. All of them present advantages and disadvantages concerning connectivity and complexity in regard to the self-organizing algorithm. Shared bus interconnects are well understood, have a simple physical structure and natively support broadcasting (of inter-AE messages), but they allow only one AE-to-AE/many transaction at a time and the bus arbiters are quite complex when fair capacity sharing or real-time support for transactions comes into play. Multi-port switches have a higher aggregate interconnect capacity than busses and support simultaneous point-to-point connections among

multiple AEs, but they are quite complex and broadcasting is difficult. The shared ring is not too complex and allows broadcasting, but point-to-point communication depends on the IP blocks and it requires a protection switching mechanism in case a ring segment fails.

Although organic enablement 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. We regard this property as instrumental for gaining industry acceptance for this approach. Major investments have been made into the establishment of IP component libraries at the functional layer. The capability to reuse existing cores as they are and to augment them with corresponding AEs preserves this investment and enables a gradual evolution path towards increasing organic content in SoCs. Furthermore, this strategy allows subsequent IP core generations to dissolve the logical separation between FE and AE IP components and merge both parts, either partially or entirely, into an FAE when indicated.

In concern to the ASoC architecture some questions need to be answered:

- Is the clear separation between autonomic and functional layer really feasible and realistic? In other words, can we come up with AE definitions and prototype implementations, which make no prerequisites on the functional IP components to provide self-organisation and controlled emergence in system behaviour?
- Will there be specific AEs per functional IP component, or is it possible and more efficient to come up with a generic AE serving several functional IPs? A refinement of this question is: Can at least the central evaluator element of the AEs be reused among a class of IP cores while the monitors and actuators are IP specific?
- What is the right level of granularity for stand-by elements at the functional level? For example, will an entire CPU be replaced in case of detection of a permanent physical defect, or is it possible to deactivate and replace sub-blocks within the CPU, e.g., individual execution pipelines or ALU (arithmetic logic units).

5 Design Methodology for Autonomic SoC

The increased complexity of ASoC design can be handled by a tailored new design methodology for ASoC. The methodology has to consider fault tolerance as an additional parameter beside, area, performance and power consumption, and will

provide a technique to build SoC architecture with autonomic or organic properties. This architecture has to be optimized for application specific SoC from today. We propose a methodology for a library and template based approach. The library will consist of parameterized templates. There will be templates for the Functional Layer and for the Autonomic Layer.
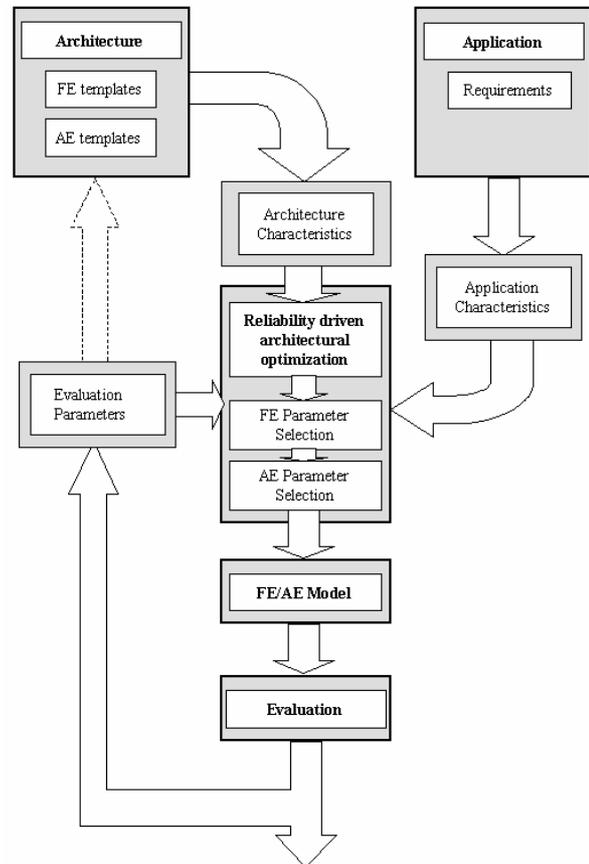
The design flow accounted to this methodology, is depicted in the Fig. 2.

In short, the methodology will consider the characteristics of the architecture and the requirements from the application. Making use of them, we will obtain a model, consisted of AE and FE templates (AE/FE Model). Upon this model will be performed the reliability driven architectural evaluation and optimization.

Fig. 2 Design Flow

The **FE templates** are the elements from the library, which will be used in the Functional Layer of the ASoC. These elements are the normal functional elements from today's SoC and will be modified to support the Autonomic Layer to watch, diagnose and repair or replace them in case of failures. Templates are provided for processing units, busses, memories and network interfaces. All these templates will be characterized by different properties. These properties can be: area, clock frequency, performance, distribution of the failure rate in time (reliability) and

parameters like number of each type of FE, width (for busses) or capacity for memories.

The **AE templates** are the newly obtained elements for the monitor, actuator and evaluator. These elements will be chosen in addition to FE templates, the choice of the FE influencing the selection of AE's. The characteristics of the AEs are different that those of the FEs. The AEs have to be distinguished at three different levels. The first level is the hardware level and the characteristics are clock frequency and area, the second level is the level, which performs the self-organization (at the evaluator level), and the last level refers to the interconnection between the AE and the corresponding FE. Several AEs can be built for a single FE, each having different characteristics and will be used for different applications.

Also a different investigation of the AS – the Autonomic Supervisor, and of the performed self-organizing algorithm, is necessary. The analysis of the self-organizing algorithm refers to the complexity, resource usage, response time and the solution it provides. Also, the analysis must consider the emergent properties of the system and how the self-organizing algorithm influences the emergent behavior. Different types of self-organization algorithm will be investigated and an appropriate representation has to be found for its parameters and the properties.

The **application requirements** are also inputs to the design methodology. An application analysis will be performed and will search for the execution times, resource usage, priorities, dependencies between tasks and constraints [13], [14], [15]. The analysis will provide the a-priori knowledge for the self-organizing algorithm, which has to guarantee the application constraints. For example, the algorithm might learn the system behavior, when it is not very busy and it might not respond correctly when the system, for some reason, becomes busy. This has to be prevented, by using the a-priori knowledge to constrain the self-organizing algorithm and so to constrain the emergence behavior of the entire system. All these characteristics, found after the application analysis, will be used by the self-organizing algorithms for keeping the system still functional in case of failure of one or more FEs. What will be important in the application requirements is the time to function, which will say for how long the system must be functional at the required constraints.

After obtaining the architecture (templates) characteristics and the application characteristics, the **reliability driven architectural optimization** will give notice of how many resources are needed in order for the application to work. If in a today's SoC a failure occurs, the performance of the system will suffer going to the point when actually the system can no longer work. The ASoC has to be able to handle failures. Our solution is the use of redundancy and reorganization. Because now failures are taken into consideration, more resources will be considered. The additional resources are redundant and will be used in case of failures. The problem is how to compute the number of extra resources in order to satisfy given reliability constraints. We know that the system has to work for at least a certain period of time, given in the application requirements, and also we expect that the FE will have failures and it is known the distribution of the failure rate. Having this information, a relation for computing the degree of redundancy has to be developed.

We will provide, then, a tool to help the designer to check if the selected resources, busses, and memories and processing units, will work under the required constraints.

The reliability driven architectural optimization has two situations. First, the initial configuration of the system has to be made and the second situation is after the evaluation step, which provides a quality measure, through the evaluation parameters. The selection of the initial configuration can be done in two ways. One way is to design a normal SoC, without taking into consideration the redundancy and fault-tolerance. This model will be then evaluated. But, there is a big drawback, because the evaluation phase will run too many times for big systems or systems having elements with low reliability. The advantage of this method is that we can use existing methods for designing the SoC. The other way is to start with creation of an already redundant architecture. This architecture has to be obtained and for this we propose to start from application analysis and to take into consideration the reliability. After the evaluation step, the architecture can be accepted or not. If the architecture is not accepted, a new architecture has to be refined until all requirements are fulfilled.

After the selection of both the FE and the corresponding AE, an **FE/AE Model** is obtained. Upon this model, an **evaluation** has to be performed to see if it complies with the requirements. If the architecture will not be accepted, the results of the evaluation are used as **evaluation parameters** for selecting another architecture. The evaluation of the FE/AE model will contain three parts: the exploration of the architecture, the change of the system's state after a certain failure and the evaluation of the new state. Our proposal is to build a dynamic fault tree for exploring the architecture. The evaluation has to consider along with the established characteristics in the SoC design, the new characteristics, which are referring to reliability, distribution of failure rate and time to function. Here the phenomenon of emergence

has to be considered and analyzed. This phenomenon appears due to the interdependencies between the AE inside the autonomic layer and the interdependencies between the autonomic layer and the functional layer. The evaluation process will use the information about the emergent properties of the system and it will be done in a probabilistic manner based on failure rate. The evaluation process will take into consideration also the self-organization process with dynamic binding and scheduling.

The fault tree will be constructed under the assumption that a failure takes place at each processing element. So, after a certain failure, dynamic optimization will be performed over the new configuration. This means that dynamic binding and re-scheduling will be performed and these operations must be considered also in the evaluation process. From each node of the tree the sons of this node will be built, each son meaning a possible failure of one of the working processing elements. The nodes in the tree represent states, in which the system goes after a certain failure, except for the root, which will be the initial state, when the elements are running normal, without any failure. A certain node will have information about the state of the system and the probability for the system to reach that state, in that order of failures, given by the path from the root to the node.

From the existing approach on dynamic fault tree analysis [5], we bring two new concepts. First, the obtaining of the new state or leaf, we have to take into consideration the self-organizing algorithm and the current state in order to obtain the new state. The second concept that we are bringing is an analysis performed over the state, which will be chosen.

The new state is obtained from the old one, after a certain failure occurred and then after the self-organizing algorithm performed the dynamic optimization. An important issue is the solution provided by the self-organizing algorithm. If this algorithm finds the best possible solution to obtain the best performance, the tree will be more like a directed graph because some of the child nodes will be the same.

Another possibility is if the self-organization algorithm doesn't provide the best solution but an optimized one, close to the best one. This approach is more realistic because the self-organization has to be performed in real-time and the best solution can be too difficult to compute with the time and the resources available. Still the tree obtained with this algorithm will not be a complete tree, but a graph like in the previous case. Because the algorithm will try to find an optimized solution it is probable that many times the nodes will have the same successors.

The analysis of the self-organizing algorithm helps us to explore in which state the system will go after a failure. Before expanding a certain state, it has to be evaluated to see if it still complies with the application requirements. The state evaluation will consist from:

- Communication analysis [13], [14];
- Execution time analysis [15];
- Conflict analysis for communication;

These analyses will give measures about the usage of the processing units, about how many resources are needed in the communication and if the state is still meeting the timing constraints.

6 Conclusions and Outlook

Secure design of reliable systems is a new challenge to master nanoelectronics when it hits physical limits. The successful transfer of an autonomic SoC design framework and method to industry requires the definition of open interfaces within the autonomic layer, between these two layers, and towards higher layer software structures. The platform based SoC approach reduces development effort by making use of proven IP macros. In order to prevent isolated, proprietary solutions, the ASoC framework must seamlessly fit into the SoC design methodology and form an open commonly available industry standard.

For the time being, our target is an ASoC framework where the full spectrum of autonomic behavior is considered by the designer prior and during system architecture development. In a second step, we can image to apply the autonomic layering concept recursively on itself. This will result in optimized AE operations and inter-AE communication structures developing towards truly emergent system behavior.

7 Literature

[1] Austin, T. M.; Diva: "A Reliable substrate for deep submicron micro-architecture design". - In: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture, Pages 196-207, IEEE Computer Society, 1999

[2] Brinkschulte, U ; Becker, J ; Ungerer, T: "CARUSO – an Approach Towards a Network of Low Power Autonomic Systems on Chip for Embedded Real-Time Applications", WPDRTS04 Workshop, Santa Fee, NM, April 2004.

[3] Benso, A. , et al.: "Online and Offline BIST in IP-Core Design", IEEE Design & Test of Computers, September-Oktober 2001, pp. 92 – 99.

[4] Choi, K.; Soma R.; Pedram M.: Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times, DATE 04, pp. 4-9, Paris, February 16-20, 2004

[5] Joanne Bechta Dugan and Tariq S. Assaf. Dynamic Fault Tree Analysis of a Reconfigurable Software System. *The 19th International System Safety Conference*, Huntsville, Alabama, September, 2001.

[6] Ernst, D.; Kim, N. S.; Das, S.; Pant, S.; Pham, T.; Rao, R.; Ziesler, C.; Blaauw, D.; Austin, T.; Mudge, T.; Razor: "A low-power pipeline based on circuit-level timing speculations". - In: Micro Conference, December 2003.

[7] IBM Corporation, "RISCWatch Debugger User's Guide", Version 5.1, May 2003.

[8] Horn, P: "Autonomic Coumputing: IBM's Perspective on the State of Information Technology", IBM Corporation, Oct. 2001, http://www.research.ibm.com/ autonomic/manifesto/

[9] "International Technology Roadmap on Semiconductors 2003"

[10] Kephart, J.; Chess, D.: „The Vision of Autonomic Computing, IEEE Computer Magazine, January 2003, pp. 41 – 50.

[11] Mitra, S.; Huang, W.-J.; Saxena, N.R.; Yu, S.-Y.; Mc Cluskey, E.J.: "Reconfigurable Architecture for Autonomous Self-Repair", IEEE Design and Test of Computers, May-June 2004, pp. 228-240

[12] De Micheli, G.: „Designing Robust Systems with Uncertain Information", Asia and South Pacific Design Automation Conference (ASPDAC 03), January, 2003

[13] Siebenborn, Bringmann, Rosenstiel: Communication Analysis for System on Chip Design, DATE February 2004, Paris, France, Proceedings of the Design Automation and Test in Europe Conference (DATE) 2004, Pages: 648-653

[14] Siebenborn, Bringmann, Rosenstiel: Communication Analysis for Network-on-Chip Design, International Conference on Parallel Computing in electrical Engineering (PARALEC), 2004 Dresden

[15] A Siebenborn, O. Bringmann, W. Rosenstiel: Worst-case Performance Analysis of Parallel, Communicating Software Processes, Proceedings of the Tenth International Symposium on Hardware/Software Codesign 2002, Vol., 2002 Pages; 37 – 42. CODES, May 2002, USA

[16] Schaumont,P.; Verbauwhede, I.; Keutzer, K.; Sarrafzadeh, M.: „A quick safari through the reconfiguration jungle", Design Automation Conference (DAC-01), pp. 172–177, New York, June 18–22 2001. ACM Press.

[17] Stollen, N. , Leatherman, A, Ableidinger, B , Edgar, E: "Multi-Core Embedded Debug for Structured ASIC Systems", DesignCon2004, February 2-5, 2004, Santa Clara, USA.

[18] VDE/ITG/GI – Positionspapier: „Organic Computing: Computer- und Systemarchitektur im Jahr 2010", Arbeitsgruppe Organic Computing

[19] Xilinx Corporation, "ChipScope Pro Software and Cores User Manual", v6.2, February 13, 2004