AIS – Autonomous Integrated Systems

From Volker Schöber, Oliver Bringmann, Andreas Herkersdorf, Walter Stechele, Norbert Wehn, Matthias May, Daniel Ziener, Abdelmajid Bouajila, Daniel Baldin, Johannes Zeppenfeld, Björn Sander, Jürgen Teich, Maurice Sebastian, Rolf Ernst, Dieter Treytnar

Gefördert durch das Bundesministerium für Bildung und Forschung

In this article, the AIS cluster research project presents an overview of the results of the 6 R&D partners after three years of work. AIS proposes a new design methodology for MPSoCs for autonomous integrated systems, introducing an autonomous layer. Using this methodology, chips will be designed to react with autonomous characteristics during application. A main focus during the research was to incorporate interfaces between the design tools and the methodologies. As a result, different design methodologies to increase autonomous behaviour are presented. Autonomous elements and functions will be included to monitor and control data-paths, control-paths and communication buses to react at block or system level. The robustness will be optimised by inserting self-optimisation services including an operating system that is able to replicate, monitor and migrate tasks and services. The article will present the highlights of the partners' work and an FPGA-based prototyping platform which shows the interaction of the complementing EDA methodologies of the partners in one design.

Motivation

With the growing complexity of nanoelectronic integrated systems, the importance of non-functional requirements – like robustness or lifetime – are growing. Moreover, the area of application is often unknown or not addressed. As a result, nanoelectronic systems might fail. In classical chip designs, systems are often specified for the worst case. This will lead into over-engineered solutions in most cases. With AIS, we propose a new design methodology to create large nanoelectronic systems including communication buses, memories, computation modules, sensors, and actors besides control and data paths.

To give an example: Instead of designing chip buses – like the AMBA bus from ARM CPUs – for the worst case, future system buses are able to react under harsh environments. The encoding technique will be changed and the robustness of the chip enhanced. As a result, the new encoding technique will reduce the performance of the bus only when it is needed. Moreover, additional power consumption due to stronger encoding techniques can be reduced to a minimum. This example shows the envisioned flexibility of chips with autonomous behaviour.

The characteristics of electronic systems to react autonomously and accommodate flexibly to faults and modifications of the environment as well as the internal state require a new kind of thinking in the design-process. Not only function, area and power consumption are considered to be at the forefront, but also detecting and reacting from systems under defective operative conditions. That means sensors, evaluators and actuators in MPSoCs are going to detect sporadically appearing faults as well as analyse them and initiate actions to guarantee reliable, flawless operation. An autonomous operating system which works on the principle of self-organisation is necessary in addition to error correcting mechanisms, methods of correction and autonomous elements for an MPSoC-platform. The operating system manages hardware resources and allows the usage of the corresponding hardware-architecture by offering a machine-oriented software level (elementary operating system), which provides an API in terms of many services. This software level is going to be self-optimising and self-healing on the principles of self-organisation. For such dynamic self-healing and optimisation it is necessary to have platform data about possible ways of communication, as well as data about processors and their performance characteristics. In this way dynamic optimisation and the self-healing of software will be supported by the hardware platform. Figure 1.02 shows the areas of research done by the partners in the project.

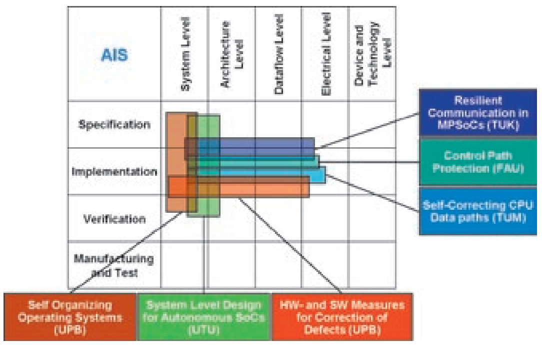
AIS
System Level
Architecture Level
Dataflow Level
Electrical Level
Device and Technology Level
Specification
Implementation
Verification
Manufacturing and Test
Resilient Communication in MPSoCs (TUK)
Control Path Protection (FAU)
Self-Correcting CPU Data paths (TUM)
Self Organizing Operating Systems (UPB)
System Level Design for Autonomous SoCs (UTU)
HW- and SW Measures for Correction of Defects (UPB)

Figure 1.02: Research areas of the AIS project partners.

Goals and organisation of the project

The goal of the project is that the designed chips will have additional functionality to react during application to find errors, faults and degradation of the reliability of single components and will be able to react for keeping the system running. To do this, autonomous elements will be introduced into the system design. These autonomous elements will both monitor data-paths, control-paths and the structure of communication on a

Forschungspartner
Uni Erlangen-Nürnberg
TU Braunschweig
TU Kaiserslautern
TU München
Uni Paderborn
Uni Tübingen

Unterstützt durch Industriepartner
Atmel Germany
Alcatel-Lucent
ChipVision Design Systems
GLOBALFOUNDRIES Dresden
Infineon Technologies
Melexis
NXP Semiconductors Germany
OneSpin Solutions
Robert Bosch
TexEDA Design
X-Fab Semiconductor Foundries

Förderkennzeichen des Vorhabens
01M3083

Laufzeit
01.12.2006–30.11.2009

functional level and also react to change operating conditions as well as faults. When fault correction occurs actions will be initiated and, if necessary, superior levels informed about the changed operating status. The robustness will be optimised due to the insertion of self-optimisation. Services will be investigated that are used in an operating system for internal self-optimisation and self-healing through replication, monitoring and migration on the basis of reliable structures of communication.

The goals of the project are in accordance with the principles of self-organization, self-protection and self-healing. The components have to detect their status by themselves and report it to the operating system level. The spreading of errors and the negative impact on the reliability of the MPSoC will thus be detected and, if possible, corrected autonomously during operation. For this, one needs some precautions in the SoC design: compatible expansions in design methodology, EDA tools for analysis autonomy and reliability, and last but not least, support from hardware-oriented software like operating systems. Three behavioural levels are used to model the system characteristics needed in the future: functional, autonomous and operating system level, as shown in Figure 1.03. The autonomous behavioural level is going to build a close connection with the relating operating system. For several components the relevant parameters will be identified from the system model, and one corresponding component model will be evaluated and optimised. This modelling of single components will be considered during the separate design steps starting with specification and continuing until implementation. One objective is to analyse, explore and optimise the necessary system resources at system design level with the help of component models, to guarantee reliability due to the autonomous behaviour of the SoC during operating.

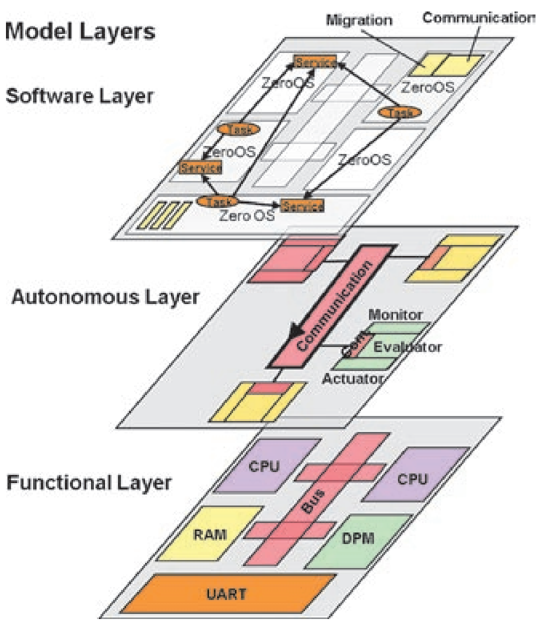

Figure 1.03: Beside the software and the functional layer, an additional layer is introduced by AIS, the autonomous layer.

To reach these goals AIS organized its research with two work packages (WP). A new kind of system design methodology for autonomous integrated systems is explored in the first work package, depicted on the left side. The second work package introduces the design of components to fulfil the previously named requirements on the component level, as you can see on the right side of Figure 1.04. With this new component design methodology and components of architecture will be dimensioned with autonomous characteristics and provided for system design. In a process of exploration and integration these components will be combined with an autonomous behaviour based operating system environment at system level in the system level WP. Beside the hardware design, the new design process will include the operating system level of the MPSoC.

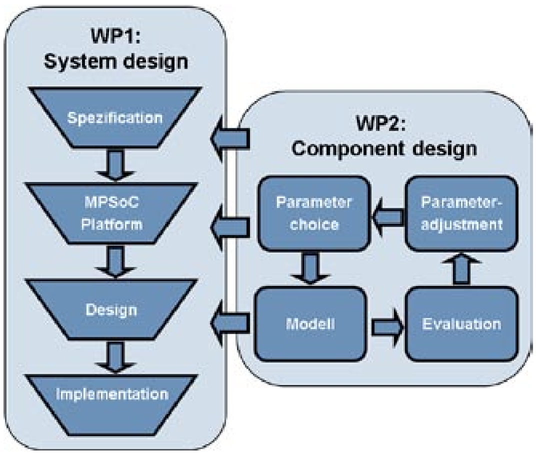

Figure 1.04: The project is organised in 2 work packages, system design (AP1) and component design (AP2).

In addition to the two WPs, the project partners will develop and exchange prototypal implementations for validation to demonstrate interoperability using a FPG prototyping platform. In the following the article presents the research partners' results beyond the WPs. At the end, the demonstration is highlighted and an outlook is presented.

Technische Universität München (TUM)

Self-correcting CPU data paths

In deep-submicron technologies, transient and timing errors are a growing problem. Transient errors consist of Single Event Upsets (SEUs) hitting storage elements and creating bit-flips in addition to Single Event Transients (SETs) where particles striking the circuit logic create glitches which can be sampled by registers and hence create data corruption. Multiple concurrent transient errors have already also been observed in SoCs.

The objective of TUM was to build a self-correcting CPU data path towards single and multiple transient and timing errors. The hardening techniques should be transparent for the software layer and more efficient than massive redundancy techniques (such as Triple Modular Redundancy). The Nicolaidis shadow register [Nicolaidis99] is an efficient transient and timing error monitor which consists of adding an extra register with a delayed clock to each register that should be protected. This detects single SET, SEU and timing errors. In order to also detect multiple SEUs, we extended this technique with an Error Correcting code (ECC) [Bouajila09]. The result is a technique using timing (shadow register) and information (ECC) redundancy that is able to detect single and multiple SEU, SET and timing errors.

Starting from this study of fault models and error monitors, we designed and built a fault-tolerant CPU data path where a corrective micro-rollback is performed every time an error is detected. As the monitor detection latency is only one cycle, we don't need to have expensive FIFOs storing multiple previous states in order to achieve rollback as described in [Tamir90]. Our correction scheme is able to re-execute errant operation by "going-back" only one cycle, therefore we need only history registers to store the previous inter-pipeline registers state (see Figure 1.05) [Stechele07] [Bouajila06]. This is a major contribution in comparison to the costly classic rollback, which stores requires a rollback to the last checkpoint potentially made "thousands of cycles" before, and has therefore has large error correction performance overheads. In our scheme error detection and correction require only two cycles, independent of where the error occurred and whether it is a single or a multiple error.

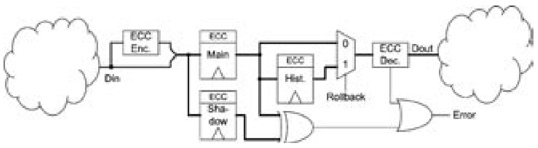

Figure 1.05: Inter-pipeline register providing multiple error detection (shadow+ECC), correction is being performed through micro-rollback using history registers content

We implemented our protection scheme in a Leon3 VHDL soft core. The VHDL code has three different working modes:

» Only error detection: only single/multiple error monitors are implemented, we only detect errors. This mode can be useful in applications which tolerate a maximum error rate.
» Single error detection and correction: the monitor uses shadow register (no ECC will be implemented in the Figure 1.05); error correction is achieved through micro-rollback thanks to the last cycle state stored in history registers.

Single and multiple error correction: the monitor consists of shadow registers and ECC, (see Figure 1.05), single and multiple errors are detected, error correction is achieved through a rollback from the state stored in the history registers. This self-correcting CPU processor was tested both in simulation and on an FPGA prototype using fault injection while running Mibench standard tests [Mibench]. The two-cycle penalty per error was confirmed. For instance for a 5 % error rate, the performance penalty is only 10 % on the CPI (cycle per instruction) as shown in Figure 1.06 [AIS-M2 2 2-TUM-LIS-Q09]. Our investigations have been presented to our industrial partners and we are investigating technology transfer opportunities with several business units of Infineon Munich.

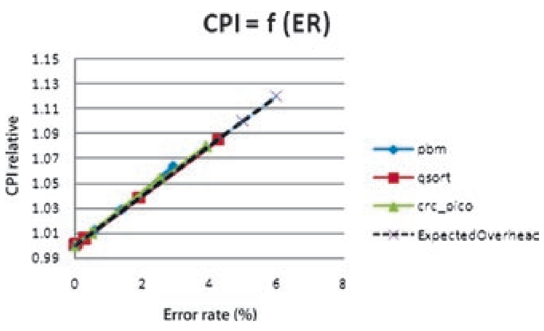

Figure 1.06: Relative CPI variation through error Rate

University of Erlangen-Nuremberg (FAU)

Control Path Protection

Single event effects and disturbances in the control path lead to errors which manifest themselves in a corrupted control flow. This may cause the carrying out of wrong instructions. Detecting control flow errors and therefore avoiding the execution of wrong or faulty instructions is a problem of growing importance with respect to reliability and security. Control path protection can be achieved by checking the control flow of a programme under execution, and if necessary, initialising correction measures, for example the re-execution of the last instructions. A quite general definition of control flow checking may be given as follows: Control flow checking denotes the task to test whether a sequence of programme counter values is correct with respect to a given programme specification.

In embedded SoCs, a CPU often executes only a few specified programmes over its lifetime. So, it is beneficial to analyse these subroutines for control flow instructions statically. It is possible to extract information about the programme flow from an executable programme during compilation time and use this infor-

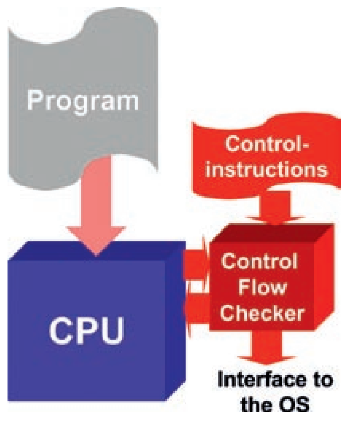

Figure 1.07: Concept of autonomously interacting control flow checker that can monitor the programme counter and correct a corrupt control flow.

mation to check the control flow of the processor with a so-called control flow checker at run-time (see Figure 1.07). On the other hand, hardware overhead and easy integration into processor design and application development flows are of the utmost importance in many cost-sensitive applications, particularly in embedded systems.

Our solution is a control flow checking architecture consisting of an additional checker unit and some few modifications of the processor pipeline. Direct jumps and branches are checked with the information extracted from the compiled code. We developed and analysed two different methods for efficiently storing this information in an on-chip memory. The correct start and the target address of a direct jump or branch are compared with the current values from the processor pipeline. Returns can be verified in this approach by introducing an additional hardware stack. Furthermore, if an error is detected, the re-execution procedure will be initiated. In this case, the address of the erroneous instruction is forwarded to the fetch stage of the CPU pipeline. The result is a re-fetch of the instruction which caused the error. The detection of errors happens during the execution of the erroneous instructions, so we have the possibility to react immediately and prevent those incorrect instructions from being executed. With this technique we therefore have no performance impact on the CPU and the compiled programme code remains unchanged, which makes our approach completely transparent to the programme developer. This approach was also presented in [ZT08] and [ZT09].

The hardware overhead for the additional checker was evaluated on the AIS demonstrator. It was verified that lookup tables and flip flop overhead amounted to less than 5 % of the CPU core requirements. So the only overhead results from the additional memory needed to monitor the control flow instructions. The challenge of integrating our approach into the common AIS demonstrator was the interaction with the data path protection developed by the project partner TUM. Both protection methods are integrated into one CPU where the error correction of one method affects the error detection of the other method. Therefore, both protection methods must be coordinated with each other. Furthermore, for the interaction with the organic operation system (developed by the project partner UPB), our programme analyser had to be adapted. Finally, an interface for the communication of the number of detected and corrected errors from the control path checker to the operating system was introduced.

University of Kaiserslautern (TUK)

Resilient Communication in MPSoCs

Recent case studies have shown that exploiting application-level resilience can drastically reduce the area and energy overhead for implementation-level resilience. E. g. in [MAY08] an LDPC decoder case study was performed. This study demonstrated that the overhead was smaller than 20 % for an error resilient LDPC architecture by exploiting application resilience. Similar investigations were performed in [BAU07, GEO06]. Application resilience requires adaptive implementation resilience, i. e., the strength of the error-resilience has to be adapted during run-time.

Thus TUK extended the AMBA [ARM99] advanced high-performance bus (AHB), which is the central communication scheme in many MPSoCs, by adding an application specific error protection scheme. This error protection scheme is run-time reconfigurable by the applications running on the Leon3 processors. So, the strength of the error-protection mode, and thus the energy and performance overhead, is adapted to the type of data sent over the bus during run-time. Up to sixteen different general or application specific error protection modes can be implemented. Various techniques with simple error detection and correction codes are provided, e. g., parity bits with ARQ, repetition codes with voting, and Hamming codes provide soft error protection. Timing errors are detected by synchronously flipping one data bit every second clock cycle in the master and the slave error detection and correction EDC unit [MAY08], [WOR04]. The active protection mode on the read/write data bus depends on the address of the transfer.

The AMBA bus extension has been integrated into the common demonstrator of the AIS project. The master EDC units log the number of errors that were detected on the data bus. Based on this information, the self-organizing operating system ORCOS from project partner UPB can change the error protection scheme on the data bus or migrate tasks to another module.

University of Tübingen (UTU)

System level design for autonomous SoCs

In order to be able to design MPSoCs which can cope with the challenges that are posed by future CMOS technology nodes, a major shift in the applied SoC design methods was necessary. While performance and energy consumption were already taken into account as optimization goals at system level, the inclusion of reliability was still an open problem. The foundation for the proposed design process is made by an executable transaction level system model written in SystemC, that allows the exploration of design decisions/parameter sets with respect to performance, energy consumption, power dissipation, temperature and reliability. The behaviour of the system can be influenced by using these design decisions and parameter sets. The model reflects the most important properties of the three logical system layers addressed in AIS (see Figure 1.02), such as the capability of the operating system to migrate tasks or the policies for power reduction that are implemented inside the hardware units as accurately as possible.

An approach for detecting errors during run-time includes failure rate registers at certain locations of a module together with modern error monitors, such as Razor [Aus03] or error correcting codes (ECC). In the registers, the numbers of errors which are detected by the monitors during a certain time interval are stored. Based on these numbers a reliability measure can be calculated using the technique presented in [Ber06]. When the reliability measure reaches an unacceptable value, actuators are triggered and/or the corresponding information is sent to the next higher block in the module hierarchy. If, for example, a large number of errors are counted over multiple intervals, the evaluator possibly concludes that the advanced aging of the component is responsible for this. When this information is propagated within the system, this may lead to the migration of tasks carried out by the operating system. The challenge of how the faults, detected by the monitors on the chip, can be propagated and aggregated during run-time, in order to make conclusions about the entire system and its subsystems, respectively, is also dealt with during the system design process. In doing so, it is taken into account, that errors occurring are possibly not visible at system level due to potential error masking.

Reliability considerations are often temperature-driven, which means that steady state or time-dependent temperatures of the components under consideration are the initial point for a further analysis. To obtain component temperatures their power consumptions have to be known. These in turn are derived from activity information. Design alternatives are evaluated at ESL with respect to several design objectives, lately also including temperature. However temperatures are dominated by local power effects – a fact, that has not been sufficiently considered at ESL until now. There is a lack of appropriate power models. Therefore, a methodology for the power analysis of embedded processors at a high level of abstraction was developed which can be used for accurate application-specific temperature and reliability considerations [San09]. Application characteristics only available at system level are combined with abstracted gate level power information.

University of Paderborn

Self- organizing operating systems for MPSoCs

While the reliability inside fault-tolerant MPSoCs can be increased by inserting error detection and error correction codes directly inside processor and bus structures some situations still need the support of higher-level mechanisms. Thus one goal inside AIS was to develop a reflective operating system which, on the one hand eases the access to hardware resources, and on the other hand increases the overall reliability of self-healing and self-optimizing components. One task of the reflective real-time operating system ORCOS is to manage the underlying hardware transparently, which is done by providing an API close to the POSIX standard to the application programmer. As a major design goal the configurability of the system offers a high performance while being very small in means of binary and memory footprint. Reflective components for monitoring the execution of services and tasks as well as replication and migration technologies have been integrated as well.

Through close interaction with the autonomous hardware the operating system is capable of analysing and handling faults during the execution of a service and can prevent future execution failures by using the replication or migration mechanisms provided without any interaction of the application programmer. In order to guarantee the operability of the system a transparent communication structure has been developed which ensures the consistency of communication channels between tasks and services even after a service has been migrated from a faulty node to a node with less failures.

As communication between such kinds of distributed systems is a major issue, communication protocols for routing and migration have been analysed and simulated for a large set of nodes. While it was possible to reduce the number of transmission errors by using biologically inspired algorithms for routing packets between nodes it was observed that small changes in the configuration of such a distributed system may result in completely different behaviours, which make the use of simulations essential for ensuring the correctness of these algorithms.

The combination of the hardware approaches together with the reflective operating system on the demonstration platform showed the usability of the overall approach.

Technische Universität Braunschweig

Hardware and software measures for correction of defects

The introduction of special system functions to react to internal errors in computational cores as well as in the communication infrastructure has an impact on a wide variety of non-functional aspects. In real-time systems compliance with application-dependent timing constraints is of essential importance for the correct functionality of the whole system. In safety related real-time systems deadline failures entail the same consequences as incorrect functional behaviour. Since safety related systems, e.g. in aerospace or automotive applications, are a main target of the semiconductor industry, errors that affect timing guarantees are of premier interest in the AIS project. The goal in AIS is to develop methods to accurately quantify the effects in the context of typical safety standards, such as IEC61508, and come up with solutions to improve safety and availability.

The reliability of fault-tolerant MPSoC-based real-time systems has been explored from two different points of view. On the one hand the communication infrastructure has been considered. Based on a given error correction technique, a timing analysis methodology has been developed to derive the reliability, i. e. the probability of failure-free operation during a certain period of time. Two different mechanisms are covered by the analysis and compared with each other: error correcting codes and error detecting codes with retransmission of erroneous messages until a message has been transmitted correctly. Results show that selecting the best strategy is no trivial decision. The best strategy strongly depends on system and application characteristics, pointing out the need for analytical methods for taking optimal design decisions.

Furthermore the reliability of fault-tolerant computational cores has been investigated. For that purpose a check -pointing scheme has been specified in such a way that safety-critical tasks are duplicated and the results of the original tasks compared with those produced by the replica. Whenever an error occurs time-consuming rollback is necessary. All these fault tolerance mechanisms might have effects on the system's functionality due to impacting the logical as well as temporal correctness of the system, so that the application of special analysis algorithms is necessary to verify the final reliability as required by the safety standards.

Another topic that has been included in these research activities is the design of systems where functions of different safety levels are mapped to the same MPSoC system. Such systems with mixed safety criticalities will be dominant in future embedded systems, where many functions will be merged on the same IC. To minimize system overhead it is necessary to verify safety levels individually. This is difficult where resources are shared, such as cores and buses. It has been shown that it is feasible to integrate communication channels with widely different safety requirements into a single bus and formally verify the safety levels independently. For that purpose the formal analysis methodology mentioned above has been adopted to the design principles of safety standards like IEC 61508 (design w.r.t. SIL).

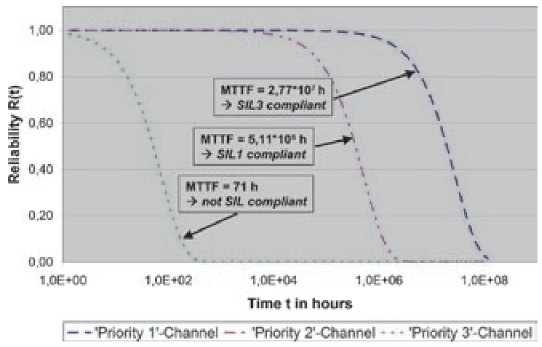

Figure 1.08: Mixed-criticality design according to IEC 61508

Within the context of real-time system analysis a demonstrator platform has been applied to illustrate the effects of errors on system timing. For that purpose errors have been injected during the transmission of large data chunks over the AMBA bus, leading to retransmissions and missing potential deadlines.

Demonstration of AIS design approach using a FPGA platform

An MPSoC demonstrator is implemented to demonstrate the efficiency of the various techniques in a realistic environment. This demonstrator is implemented on a Gaisler GR-CPCI-XC4V development board. The Xilinx Virtex-4 LX100 FPGA [XIL] on the board contains multiple Leon3 processor cores [GAI], an SDRAM controller, and several peripheral units connected via an AMBA AHB. The MPSoC consists of autonomous hardware units for error-resilient processing and interconnects as shown in Figure 1.09. The different units can monitor and analyse sporadic disturbances and trigger adequate reactions autonomously. Different techniques are integrated into the MPSoC for a holistic protection of the system: a self-correcting data path (TUM) and control flow checking (FAU) in the LEON3 processor cores and a run-time configurable data protection of the AMBA AHB (TUK). The self-organizing operating system ORCOS (UPB) monitors the error rates on the different processors and can migrate tasks form erroneous to reliable processors. The integration of the different methods of the project partners into the common demonstrator strengthened cooperation in the AIS project considerably.

The application running on the demonstrator is a channel decoding system, i. e., a turbo decoding system. Turbo decoders are a hot topic in wireless communication and an important component in baseband receivers for wireless communication systems such as WiMAX, CDMA2000, UMTS, and 3GPP LTE. Channel decoding corrects errors induced during data transmission over the wireless channel. Turbo decoding is based on an iterative algorithm with high computational complexity and high communication bandwidth. A very interesting aspect of this application is the fact that the decoding algorithm belongs to the class of probabilistic algorithms. These algorithms have a certain error-resilience. Thus, we can not only investigate implementation resilience, but also the interrelationship between application and implementation resilience with this demonstrator and we can explore and verify the various error protection techniques under various failure scenarios.

The application mapping is shown in Figure 1.09. Data generator, Turbo encoder, wireless channel, and error monitor are implemented as dedicated hardware module inside the FPGA. The core unit, the Turbo decoder, is implemented in software running on multiple Leon3 CPUs. Data to and from the turbo decoder is transmitted over the AMBA bus. Additionally, the main memory (SDRAM) is connected to the AMBA over the memory controller.

The operating system ORCOS is capable of monitoring the health status of an MAP decoder service. It uses the

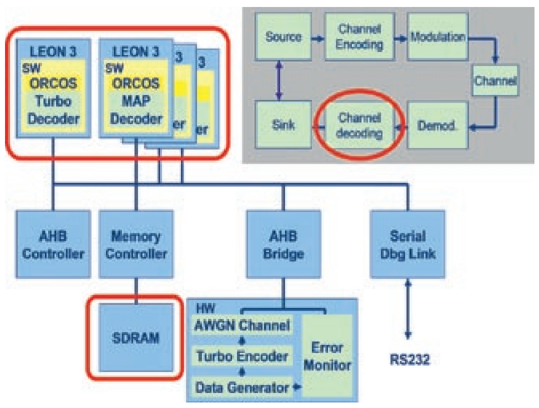

Figure 1.09: Mapping of the channel coding system on the demonstration platform

monitoring interfaces of the error detection and error correction hardware. The Turbo decoding task is to send requests to the individual MAP decoder tasks running on the Leon cores using the transparent communication framework of the operating system. During execution the operating system may decide, based on the error values provided by the hardware, to migrate a decoding task from one processor to a more reliable processor. Although the former decoder will not be available any more the transparent communication framework ensures the connectivity of the Turbo decoder to the MAP decoder unit. Three scenarios are emulated to evaluate various error protection techniques:

» In normal operation, no errors are injected. This is our reference.
» In the failure operation, errors are injected into the AMBA bus communication, the control and data paths of the Leon3 CPUs by using pseudo random generators based on LFSRs.
» Autonomous error handling in processors, communication and operating system is activated during autonomous operation.

Our demonstrator can be considered as a rapid prototyping system for error-resilient MPSoCs. It allows fast design space exploration by emulating various error protection techniques with varying failure rates on the microarchitectural level. The impact on the system behaviour can be evaluated with respect to overhead in area and latency. It is important to point out that exploration on the microarchitectural level is mandatory for analysing the impact of transient errors in all parts of the hardware. A simulation at this level would result in extremely long simulation times because the system behaviour has to be modelled at microarchitectural level and has to be monitored over a long time period.

In the TUM self-correcting CPU data path, error detection is based on the Nicolaidis shadow register technique [Nicolaidis]. Error correction uses a customized micro-rollback [Tamir90]. As the error detection latency is only 1 cycle, storing the last cycle state is enough to retry the errant operation. Therefore we added history registers. Through pipeline micro-rollback, the errant operation will be retried. Any error will be detected and corrected with a fixed penalty of 2 clock cycles. This correction penalty is independent of the stage at which the error occurred. This has been checked in ModelSim and in the FPGA prototype.

The number of needed clock cycles for re-execution of an erroneous control instruction using the control path protection of FAU depends on the currently executed instruction. On a simple program counter increment (no control flow instruction) we are able to correct the error in one additional clock cycle, whereas a correction of an erroneous return instruction needs five clock cycles. Furthermore, cache misses due to falsified branch or jump targets have also an impact on the latency.

We presented the demonstrator at the exhibition of the DATE'09 conference in Munich, at the edaWorkshop'09 in Dresden (Figure 1.10), and at the "Lange Nacht der Wissenschaften" in Nuremberg, Fürth, and Erlangen. Furthermore, the demonstrator will be presented in an accepted paper on the DATE'10 conference [MAY10].

Figure 1.10: Demonstration of the AIS results during the edaWorkshop09.

Summary and Outlook

The results of the AIS project demonstrates the advantages of designing complex nanoelectronic chips with autonomous behaviour. A new abstraction layer – called the autonomous layer – is proposed. This layer contains information to handle the intrinsic state of the SoC, like the number of soft errors occurred in the data path. Using the autonomous layer in a design platform makes a new design philosophy for the industry possible: **Creating a design environment for SoCs with autonomous behaviour**. Instead of creating autonomous behaviour by specifying the functional requirements the design flow helps to reach the goals of the target market. The AIS project shows the usability of the interfaces to combine different measures that have been explored by the partners individually.

To give an example: Hardware- related measures like self-correcting CPU data paths are aggregating the number of errors within one module. The operating system ORCOS reads out the accumulated events of mismatches. Based on a given threshold, ORCOS migrates critical tasks to other hardware resources with fewer errors. As a result in a given MPSoC environment, the system will migrate safety-critical applications to hardware resources with the highest reliability measured during applications. It increases the depths of autonomous behaviour by enabling the interaction and combination of individual approaches, like:

» Control path protection,
» Self-correcting CPU data paths,
» Resilient communication buses,
» System level design for autonomous SoCs,
» Self-organising operating systems for MPSoCs, and
» Reliability of fault-tolerant MPSoC-based real-time systems.

The use of such techniques is useful in many applications where cost and computational complexity have to be taken into account. As in the DRAM or solid-state drives (SSD) market, the industry is looking for design approaches to use unreliable technologies for reliable applications. The results of AIS are promising for these new design methodologies for a new class of reliable MPSoCs with unreliable hardware. To give few examples where autonomous behaviour can help:

» In future automotive application, there will be a large variety of communication. Safety critical buses in automotive applications will be the "nervous system" to manage and control fully electric cars. Due to costs and energy consumption, industry is looking for mechanisms to reduce the number of parallel buses within one car and to raise the reliability of data transmission. Sending signals with different priorities and individual guaranteed response times is a prerequisite to mixing data communication within one bus so as to reduce costs and energy.
» Security and Chip card applications will need to improve the security level in future applications. The system application needs a stronger link to the intrinsic state of the hardware. The number of hardware monitors will increase. Protection mechanism of control and data path will improve the security of such applications.
» CPUs with high performance data buses already use sensors inside the chip. Multicore applications are already state of the art. Today for the first time applications are using CPU and GPU together to increase the utilisation of the hardware. A new class of operating systems – like the ORCOS demonstrated – provides the opportunity to improve this trend and to link hardware resources and heir intrinsic state together to improve the performance and reduce the margins of worst case scenarios.

Based on the promising research results the AIS-Project partners propose to extend existing industrial design flows. To utilize the results in an industrial environment the research partners are open to identifying an evaluation scenario together with the industrial partners.

Kont@kt (AIS):
Dr. Volker Schöber
fon: (05 11) 7 62 – 1 96 88
schoeber@edacentrum.de

Publication of the project and references

[AIS-M2 2 2-TUM-LIS-Q09] A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, „Evaluation von AE in einen CPU-Datenpfad", AIS Milestone report , M2.2.2-TUM-LIS-Q09, 28.02.2009

[ARM99] ARM: AMBA specification, ref. 2.0, May 1999, http://www.arm.com/.

[Aus03] T. Austin et al.: „Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation", Symp. On Microarchitecture (MICRO-36), December 2003.

[BAU07] J. Bau, Q. Jacobson, R. Hankins, B. Saha, A. Tabatabai, and S. Mitra, „Error Resilient System Architecture (ERSA) for Probabilistic Applications," in IEEE Intl. Workshop on Silicon Errors in Logic – System Effects, apr 2007.

[Ber06] A. Bernauer, O. Bringmann, W. Rosenstiel , A. Bouajila, W. Stechele, A. Herkersdorf: „An Architecture for Runtime Evaluation of SoC Reliability", In INFORMATIK 2006 – Informatik für Menschen, volume P-93 of GI-Edition – Lecture Notes in Informatics, pages 177-185, Köllen Verlag.

[Bouajila06] A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, A. Bernauer, O. Bringmann, W. Rosenstiel, „Organic Computing at the System on Chip Level", Proceedings of the IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2006). Springer, October, 2006

[Bouajila09] A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, „Multi-bit Soft- and Timing Error Detection for CPU Pipelines", Workshop 2009 – Electronic Design Automation (EDA), Dresden, Germany, 26.05.2009 – 28.05.2009

[GAI] Gaisler Research, http://www.gaisler.com/.

[GEO06] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, „Probabilistic Arithmetic and Energy Efficient Embedded Signal Processing," in International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES'06), Oct 2006, pp. 158-168.

[MAY08] M. May, M. Alles, and N. Wehn, „A Case Study in Reliability-Aware Design: A Resilient LDPC Code Decoder", in Proc. IEEE Design, Automation and Test in Europe (DATE '08), Munich, Germany, Mar. 2008, pp. 456–461.

[MAY10] M. May, N. Wehn, A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, D. Ziener, J. Teich, „A Rapid Prototyping System for Error-Resilient Multi-Processor Systems-on-Chip." Accepted for publication at DATE 2010, Dresden, Germany.

[Mibench] http://www.eecs.umich.edu/mibench/

[Nicolaidis] Nicolaidis, „A Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies", in 17th IEEE VLSI Test Symposium, 1999.

[San09] B. Sander, J. Schnerr, O. Bringmann: „ESL Power Analysis of Embedded Processors for Temperature and Reliability Estimations", International Conference on Hardware/Software Codesign and System Synthesis, Grenoble, France, 2009.

[Seb08] M. Sebastian, R. Ernst. „Modelling and Designing Reliable On-Chip Communication Devices in MPSoCs with Real-Time Requirements". In 13th IEEE International Conference on Emerging Technologies and Factory Automation. Hamburg, 2008.

[Seb09] M. Sebastian, R. Ernst. „Reliability and Safety Guarantees in Modern MPSoCs with Real-Time Requirements". edaWorkshop 2009. Dresden, 2009.

[SebE09] M. Sebastian, R. Ernst. „Reliability Analysis of Single Bus Communication with Real-Time Requirements". In 15th Pacific Rim International Symposium on Dependable Computing, 2009.

[Stechele07] W. Stechele, O. Bringmann, R. Ernst, A. Herkersdorf, K. Hojenski, P. Janacik, F. Rammig, J. Teich, N. Wehn, J. Zeppenfeld, D. Ziener, „Concepts for Autonomic Integrated Systems", eda-Workshop, Hannover, June 19-20, 2007

[Tamir90] Y. Tamir, Marc Tremblay: „High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback". IEEE Trans. Computers 39(4): 548-554 (1990)

[WOR04] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, „On-Chip Self-Calibrating Communication Techniques Robust to Electrical Parameter Variations," IEEE Design & Test of Computers, vol. 21, no. 6, pp. 524–535, Nov. 2004.

[XIL] Xilinx, http://www.xilinx.com/.

[ZT08] D. Ziener and J. Teich, „Concepts for Autonomous Control Flow Checking for Embedded CPUs", In Proceedings of the 5th International Conference on Autonomic and Trusted Computing (ATC-08), pp. 234-248, Oslo, Norway, June 23-25, 2008.

[ZT09] D. Ziener and J. Teich. „Concepts for run-time and error-resilient control flow checking of embedded RISC CPUs". Int. Journal of Autonomous and Adaptive Communications Systems, Vol. 2, No. 3, pages 256-275, 2009, Inderscience Enterprises Ltd.

SANITAS – Sichere Systeme auf Basis einer durchgängigen Verifikation entlang der gesamten Wertschöpfungskette

BMBF-Projekt zur Verbesserung der Verifikation entlang der Wertschöpfungskette für die exemplarische Anwendung an der Industrieautomatisierung gestartet.

Die Beherrschung hochautomatisierter Fertigung von äußerst komplexen Produkten, die oft höchste Anforderungen an die Betriebssicherheit erfüllen müssen, macht den Standort Deutschland heute einmalig und auch im Vergleich zu Niedriglohnländern als Entwicklungs- und Produktionsstandort wettbewerbsfähig. Der Erfolg hängt dabei wesentlich davon ab, dass die Sicherheitseigenschaften der Produkte, Systeme und der Fertigungsanlagen, auf denen sie hergestellt werden, durch eine lückenlose Verifikation garantiert werden können. Das vom BMBF seit dem 1.10.2009 unter dem Förderkennzeichen 01 M 3088 geförderte Forschungsvorhaben SANITAS erforscht und entwickelt eine ebenenübergreifende Systemverifikationsmethodik auf Basis virtueller Modelle. SANITAS bezieht dabei alle Ebenen der Produktentwicklungskette vom mikro-/nanoelektronischen Teilsystem bis zum Endprodukt in die Verifikation mit ein. So wird erstmalig eine durchgängige Verifikation entlang der gesamten Entwicklungskette bis hin zur Fertigung zur Verfügung gestellt.

Man stelle sich ein in naher Zukunft durchaus realistisches Szenario vor, in welchem ein mobiler Serviceroboter als elektronischer Assistent für alltägliche Handgriffe im Haushalt zur Verfügung steht. Anstatt jedoch zuverlässig einen frisch gebrühten Kaffee zu servieren, kollidiert der Helfer auf seinem Weg aus der Küche mehrfach und verschüttet so die Hälfte des Getränks. Die andere Hälfte geht verloren, als die Tasse knapp neben der Tischplatte abgestellt wird.

Was im skizzierten Zukunftsszenario für den privaten Endanwender einfach nur ärgerlich ist und ihn eventuell davon abhalten wird, weitere Roboter zu erwerben, besitzt im Kontext der industriellen Fertigung bereits heute eine deutlich dramatischere Dimension: In heutigen automatisierten Fertigungsanlagen ist ein Betrieb von Industrierobotern nur in Sicherheitskäfigen oder abgeschirmten Räumen möglich, um so eine Gefährdung der beteiligten Bedienkräfte aus-

Gefördert durch das Bundesministerium für Bildung und Forschung

Zusammensetzung des Projektkonsortium

Projektpartner
Fraunhofer-Gesellschaft, IIS
Forschungszentrum Informatik
Infineon Technologies AG
Micronas AG
Robert Bosch GmbH
Siemens AG
Tieto Deutschland GmbH
Universität Bremen
Universität Paderborn

Unterauftragnehmer
Universität Tübingen
OFFIS e.V. – Institut für Informatik
TU München

Förderkennzeichen des Vorhabens
01M3088

Laufzeit
01.10.2009 – 30.09.2012