Design and Architectures for Dependable Embedded Systems

Henkel, Jörg; Bauer, Lars; Becker, Joachim; Bringmann, Oliver; Brinkschulte, Uwe; Chakraborty, Samarjit; Engel, Michael; Ernst, Rolf; Härtig, Hermann; Hedrich, Lars; Herkersdorf, Andreas; Kapitza, Rüdiger; Lohmann, Daniel; Marwedel, Peter; Platzner, Marco; Rosenstiel, Wolfgang; Schlichtmann, Ulf; Spinczyk, Olaf; Tahoori, Mehdi; Teich, Jürgen; Wehn, Norbert; Wunderlich, Hans-Joachim

Proceedings of the 9th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS'11) Taipei, Taiwan, 9-14 October 2011

doi: http://dx.doi.org/10.1145/2039370.2039384

Abstract: The paper presents an overview of a major research project on dependable embedded systems that has started in Fall 2010 and is running for a projected duration of six years. Aim is a 'dependability co-design' that spans various levels of abstraction in the design process of embedded systems starting from gate level through operating system, applications software to system architecture. In addition, we present a new classification on faults, errors, and failures.

Preprint

General Copyright Notice

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

This is the author's "personal copy" of the final, accepted version of the paper published by ACM.[1]

[1] **ACM COPYRIGHT NOTICE**

©2011 ACM. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Design and Architectures for Dependable Embedded Systems

Jörg Henkel, Lars Bauer, Joachim Becker, Oliver Bringmann, Uwe Brinkschulte, Samarjit Chakraborty, Michael Engel, Rolf Ernst, Hermann Härtig, Lars Hedrich, Andreas Herkersdorf, Rüdiger Kapitza, Daniel Lohmann, Peter Marwedel, Marco Platzner, Wolfgang Rosenstiel, Ulf Schlichtmann, Olaf Spinczyk, Mehdi Tahoori, Jürgen Teich, Norbert Wehn, Hans-Joachim Wunderlich

http://spp1500.itec.kit.edu/

ABSTRACT

The paper presents an overview of a major research project on dependable embedded systems that has started in Fall 2010 and is running for a projected duration of six years. Aim is a 'dependability co-design' that spans various levels of abstraction in the design process of embedded systems starting from gate level through operating system, applications software to system architecture. In addition, we present a new classification on faults, errors, and failures.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; D.2.4g [**Software Engineering**]: Reliability;

General Terms

Reliability

Keywords

Resilience, Fault-Tolerance, Embedded Systems, MPSoCs, Dependability

1. INTRODUCTION

Moore's Law has been the pace maker for the semiconductor industry for more than four decades. It was accompanied by a) decreased per-transistor costs, b) increased performance through decreased signal delay, c) decreased energy consumption per switching activity, etc. However, as Gordon E. Moore stated in a talk at ISSCC 2003: *"No exponential is forever ... but we can delay 'forever' ..."* [37] he indicated that the exponential growth cannot be sustained forever but that it may be possible to delay the point when scalability finally comes to an end. The German national research program *"Design and Architectures for Dependable Embedded Systems"* (German Research Foundation, DFG SPP 1500) aims at exactly this point: to find new means for extending the applicability of Moore's Law beyond the currently predicted limits which are mainly related to dependability issues. In fact, it has been predicted by various sources that on-chip systems will become inherently undependable (see [47], for example) and that a paradigm shift needs to be applied by *"... building dependable systems with non-dependable components ..."* [6]. Interestingly, this paradigm is not new as John von Neumann many years before [51] already proclaimed this paradigm even though not in the context of Moore's Law reaching its limit. As for the current paradigm shift, also the ENIAC Strategic Research Agenda states: *"Emerging devices are expected to be more defective, less reliable and less controlled in both their position and physical properties. It is therefore important to go beyond simply developing fault-tolerant systems that monitor the device at run-time and react to error detection. It will be necessary to consider error as a specific design constraint and to develop methodologies for error resiliency, accepting that error is inevitable and trading off error rate against performance (e.g. speed, power consumption) in an application-dependent manner"* [18] which summarizes the purpose and the goals of our research program very well.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CODES+ISSS'11, October 9–14, 2011, Taipei, Taiwan.
Copyright 2011 ACM 978-1-4503-0715-4/11/10 ...$10.00.

1.1 Goals

Among others, we are addressing the following dependability concerns that arise when migrating to upcoming technology nodes:

a) Fabrication and Design-Time Effects

Yield and Process Variations: Yield defines the number of flaw-free circuits in relation to all fabricated circuits. A high yield is so far considered vital for an economic production line. Unfortunately, yield will dramatically decrease because feature sizes reach a point where the process of manufacturing underlies statistical variances. Future switching devices may be fabricated through 'growing' or 'self-assembly'. All known research suggests that these processes cannot be entirely controlled, leading to fabrication flaws, i.e. circuits with faulty switching devices.

Complexity: The steadily increasing integration complexity is efficiently exploited by the current trend towards many-core network-on-chip architectures. These architectures introduce hardware and software complexities, which were previously found on entire printed circuit boards and systems down to a single chip and provide significant performance and power advantages in comparison to single cores. The large number of processing and communication elements requires new programming and synchronization models. It leads to a paradigm shift away from the assumption of zero design errors.

b) Operation and Run-Time Effects

Aging Effects: Transistors become far more susceptible to environmental conditions like, for instance, heat. It causes

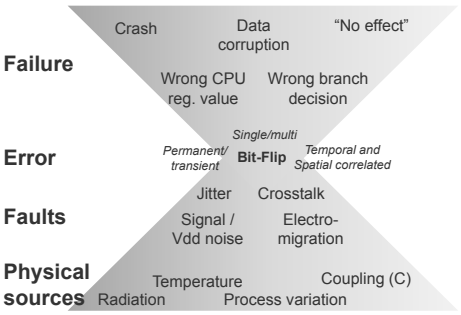

Figure 1: Cross-layer representation of faults, errors, and failures with bit error as resilience articulation point

an irreversible altering of the physical (and probably chemical) properties, which itself leads to malfunctions and performance variability over time. Though effects like electromigration in current CMOS circuits are well known, they typically did not pose a problem since the individual switching device's life time was far higher than the product life cycle. In future technologies, however, individual switching devices will fail (i.e. age) earlier than the life cycle of the system they are part of. Another emergent altering effect is the increasing susceptibility to performance variability resulting in changing critical paths over time. This, for instance, prevents a static determination of the chip performance during manufacturing tests.

Thermal Effects: Thermal effects will have an increasing impact on the correct functionality of transistors in upcoming technology nodes. Various degradation effects are accelerated by thermal stress like very high temperature and thermal cycling. Aggressive power management can produce opposite effects, e.g. hot spot prevention at the cost of increased thermal cycling. Higher integration in the 3rd dimension (3-D circuits [22]) increases the thermal problem since the ratio of surface-area/energy significantly worsens. Devices will be exposed to higher temperatures and accelerated aging effects, etc. In addition, transient faults increase, too.

Soft Errors: The susceptibility of transistors against soft errors will increase significantly. Soft errors are caused by energetic radiation particles (neutrons) hitting silicon chips and imposing charge on the nodes that flips a memory cell or logic latches. These errors are transient and randomize the confluence of device and voltage scaling towards unacceptable levels for future applications.

The relationship between the various physical fault sources in technology and the final hardware/software level failure consequences at system level is depicted in the following hour-glass diagram in Fig. 1. Thereby, spatial and/or temporally correlated bit flips represent resilience articulation points between the domains of technology and hardware/ software systems. By using spatial bit flip correlation models, crosstalk faults as well as parametric drifts of sensors and analog components can be expressed whereas temporal correlation models are used to describe SEU and timing errors for instance. This also means that bit flips can be used by both digital and analog hardware as well as software dependability methods for modeling various fault origins and, thus, as means for error insertion to validate system behavior with respect to their resilience against these fault sources.

1.2 Means

The major distinction of our research program is that dependability concerns are addressed at various levels of design and architectural abstraction since we believe that a maximum of dependability can only be achieved if various abstraction levels work seamlessly together. This focus is manifested in the SPP 1500 pyramid shown in Fig. 2, which features the following means to enhance dependability:

Technology Abstraction: Upcoming technology nodes introduce new effects that negatively impact dependability. A clear interface between technology-related research topics and the higher abstraction levels to address undependability needs to be defined. Technology Abstraction's goal is to ensure that the architectures, methods and other means are as far as possible independent from technology since technology is not a focus of this research program.

Dependable Hardware Architectures at the lowest level may refer to logic-level architectures, i.e. logic components that exhibit inherent undependability. Moving up in abstraction level, dependable architectures also stand for register-transfer components and microarchitectures like processor pipelines and instruction set architectures. At an even higher level, system-on-chip architectures like multicore and many-core systems that are connected via on-chip networks are to be researched with respect to dependability. Methods that are playing a key role are fault-tolerance, reconfiguration capabilities, self-organization etc.

Dependable Embedded Software refers to the kind of software of an embedded system that interacts very closely with the hardware. Embedded software is therefore aware of the hardware components that are invoked and how they are invoked when the software is executed [41]. For that reason, software should be able to eliminate undependability effects that are propagated from various abstraction levels of the hardware, e.g. as presented by the application in [42].

Operation, Observation, and Adaptation: Both basic components, Dependable Hardware Architectures, and Dependable Embedded Software need to be operated in a dependable way. Undependability may be a result of the combined execution/interaction of hardware and software and as such may not be noticed by either one when viewed in an isolated way. The operation of embedded resources should be able to detect those scenarios. It therefore needs an infrastructure for observation and finally a means to adapt the embedded systems to ensure a dependable operation. Capabilities like failure-avoidance, failure-recovery etc. are among the research activities.

Design Methodologies: During the design phase of the embedded systems, the inherent undependability of the transistors needs to be taken into account. Above all, the design methods and concepts must not handle undependability as an exception. Instead, undependability is omni-present and dealing with it must be viewed as the major design goal whereas past major design goals like 'area' in the 1980s, 'speed/performance' in the 1990s and 'power consumption' in the 2000s now become secondary.

The rest of the paper outlines representative projects in the five topics of error detection and test, thermal and communication management, real-time OS, embedded software, and resilient architectures. Besides these projects, there are also the analog-related projects hexFPAA [54] (which investigates reconfigurable adaptive analog filters to bypass manufacturing and runtime errors) and MixedCoreSoC [55]

which provides runtime self-adapatation for both digital and analog cores to detect and bypass defective cores.

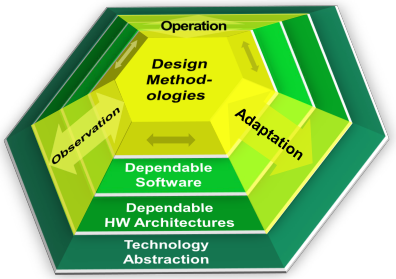

Figure 2: The pyramid for dependable embedded system

2. ERROR DETECTION, LOCALIZATION AND TEST

Fault-tolerance techniques are used to tolerate errors occurring during system operation. Fault-tolerance techniques include masking (e.g., Triple Module Redundancy), concurrent error detection, recovery (e.g., rollback and rollforward recovery) and self-repair [40].

Errors can be either permanent or temporary. Permanent errors are caused by manufacturing defects or components that break due to some wearout mechanisms. A *temporary error* is not present all the time for all operating conditions. Temporary errors can be either transient or intermittent. Some examples of causes of transient errors (faults) include externally induced signal perturbation usually due to electromagnetic interference, power-supply disturbances, and radiation due to cosmic rays. An *intermittent error* causes a part to produce incorrect outputs under certain operating conditions. Examples of causes of intermittent errors (faults) include weak components and cross-talk due to coupling between signal lines.

Temporary errors are generally detected using *concurrent error detection* (CED) techniques and retry-based techniques are used for recovery from temporary errors. A critical step in fault-tolerance is to identify the existence of errors during the execution of the system. Once the errors are detected, there are several options to recover from errors as long as the errors are detected early enough. Error recovery schemes include re-execution, rollback recovery, rollforward recovery, and checkpointing [48, 40].

While there has been a considerable amount of work on circuit level timing analysis, such analysis typically does not consider specific inputs to the circuit. For the adaption of software to technology characteristics of the chip the consideration of specific inputs in circuit level timing analysis can be beneficial. In [26] it is shown that the results of circuit level timing analysis may substantially vary for specific inputs derived from program level analysis. These results open up possibilities for compiler-level optimizations to mitigate the problems associated with process variability.

Thorough testing and precise high-resolution location of failing resources in a defective part are keys to successful implementations of defect and fault-tolerance. Thorough manufacturing testing is required to identify a defective manufactured part. During system operation, periodic testing is required to identify a defective system component with a permanent fault. Application-specific test and diagnosis techniques are useful for defect tolerance and also for detection, location, and repair of permanent faults during normal operation of the system. Test and error localization (diagnosis) techniques are also used after manufacturing, mainly for identifying defective parts and also for defect tolerance, in order to improve the manufacturing yield. Test and error localization during system operation are very complex tasks; however, they help detect permanent and transient errors and hence improve the overall system reliability.

2.1 Projects within this topic

Different projects in the SPP 1500 program address some of the research challenges related to error detection and localization of analog and digital circuits.

The goals of the project LIFT [56] are to close the gap between device and software level error resiliency methods and develop a holistic approach to error resilient system design. The main challenges involved in such a cross-layer approach include developing (i) abstraction techniques for propagating device characteristics upwards, (ii) compiler and program analysis techniques to exploit these characteristics, (iii) runtime monitoring techniques (e.g., to detect late glitches) to trigger software-level error correction (e.g., repeat particular instructions), and (iv) reliability-aware instruction set simulators.

In the project PERCEDES [64], techniques and methodologies to ensure robustness, reliability, availability, and recoverability of critical embedded systems at both the hardware and the software levels in a very cost-effective way are envisioned. This project aims to develop concurrent error detection and localization methods (hardware-level) to ensure data integrity. Also, recovery mechanisms will be designed which provide error localization from the design level, and carefully consider how these interact with both the microarchitectural and architectural levels (software-level).

The project OTERA [59] aims to increase dependability of runtime reconfigurable systems by a novel system-level strategy for online tests and online adaptation to an impaired state. This will be achieved by (a) scheduling such that tests for reconfigurable resources are executed with minimal performance impact, (b) resource management such that partially faulty resources are used for components which do not require the faulty elements, and (c) online monitoring and error checking. To ensure reliable runtime reconfiguration, each reconfiguration process is thoroughly tested by a novel and efficient combination of online structural and functional tests.

The project ARES [63] considers a SoC that includes a coarse-grained reconfigurable core (CGRC). Besides its purpose as a hardware accelerator the CGRC is used for the verification of other SoC components or to assume their functionality in case of defects. In this field of operations it is mandatory for the CGRC to be fault-tolerant. This is achieved by implementing logic for error detection and masking. If hardware defects are detected, the configured data paths are remapped in spatial [15] or temporal [16] domain to relinquish defective resources.

3. THERMAL MANAGEMENT AND COMMUNICATION VIRTUALIZATION

A major dependability concern of current and future MP-SoC (Multi-Processor System-on-Chip) are thermal problems resulting from the dynamic and static electrical energy dissipation densities in deep sub-micron CMOS technologies. Short term effects may lead to transient malfunction in sig-

nal timing or integrity, whereas long term effects include irreversible physical damages due to, for example, electromigration. The problem worsens with the inception of 3D integration as the per-surface dissipated thermal energy increases. Virtualization, i.e. the ability to separate application tasks, stored data and communication channels between tasks from the underlying physical resources have the potential to cope with thermal and other dependability exposures of MPSoC at architecture level.

Existing approaches to dynamic run-time thermal management at the MPSoC architecture level include dynamic voltage and frequency scaling [10, 24] and power gating of SoC IP cores and temperature-aware task scheduling [11, 38, 13] or remapping [53]. VMM (Virtual Machine Manager) and Hypervisors [5] are established software-based concepts for processor virtualization. Few hybrid hardware/software-based concepts (VMDq [8] and RiceNIC [52]) have been proposed for I/O virtualization with the primary objective to maximize throughput of the network interface card. The scalability of today's thermal management and virtualization techniques towards hundreds of cores in future 3D many-core processors is a major concern because of the dominantly centralized manner for gathering and evaluating system state information, the computational complexity of the specific heuristics and the associated communication overhead. Furthermore, centralized approaches represent a dependability exposure due to their single point of failure characteristic. In order to find optimized initial task mappings with respect to thermal stress, approaches for ESL power and temperature analysis can be applied at design time [44].

3.1 Projects within this topic

Within this research, we consider task and thread migration as key counter measures to the dependability exposure of a task running in a high temperature area of a 3D MPSoC at system and architecture abstraction levels. Unlike DVFS, where the task has to remain on a high temperature core unless heat conductivity or convection slowly lowers the temperature, the dislocation of a task onto a cooler core results in immediate cool-down and allows resuming reliable computation of the task in a more dependable environment. However, since tasks are typically parts of an application (e.g. represented by an interconnected task graph), task migration may not lead to a 'dangling lead' with respect to the specified connectivity between tasks.

Goal of the VirTherm-3D [58] project is to provision a scalable and robust thermal management concept for both 2D and 3D stacked many-core architectures. A single layer of such a many-core is shown in Fig. 3. The tiled many-core hardware and layered system and application software platforms depicted in Fig. 3 provide generic architecture templates used by several SPP 1500 projects beyond the ones described in this section. A distributed and hierarchical agent system in addition to the runtime environment initiates proactive task migrations onto cooler processing resources while a communication virtualization layer dynamically adapts and protects connectivity between (migrated) tasks and external I/Os [14] (See transition between left and right section of Fig. 4). Scalability of the thermal management is achieved by constraining the number of compute tiles managed by a single software agent. Agents are hierarchically structured to supervise and manage agents at the next lower layer. In order to achieve low latency and service class differentiation for Gb/s link interfaces, the communication virtualization aims at a multi-context hardware-based finite state machine approach with multi-priority buffering services in the I/O tile and network adapter functions of compute tiles. The SMASH [62] project targets proactive, temperature-driven thread migration and shadowing on heterogeneous multicore processor platforms by corresponding enhancements to the ReconOS framework [30]. SMASH extends the notion of threads to apply for hardware threads running on dedicated function or reconfigurable cores as well (see coexistence of RISC and HW accelerators in upper compute tile of Fig. 3). The runtime system maintains thermal sensitivity maps to balance the temperature gradients among multiple hybrid cores. To catch core failures, shadow threads are assigned to dependable cores with low thermal stress. Signature traces are evaluated between threads running on hot and cold cores and may trigger thread restart or migration.

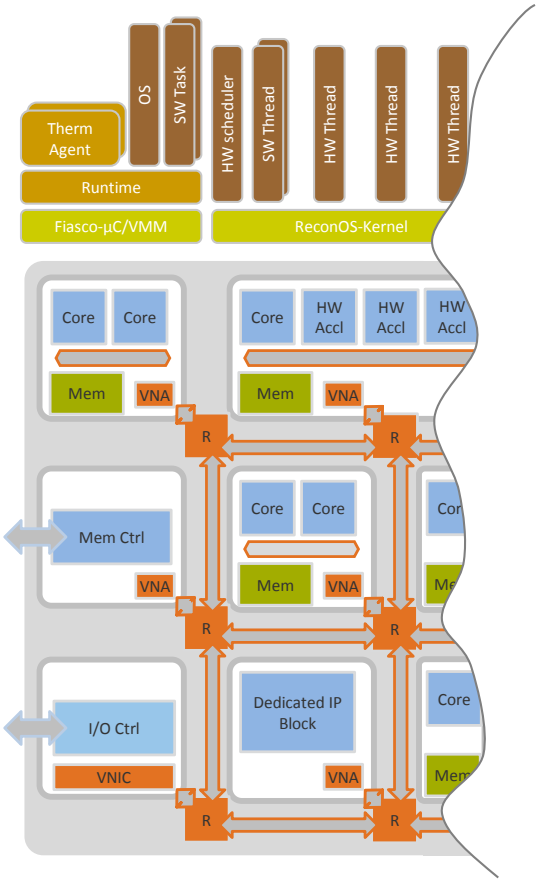

Figure 3: Tiled many-core hardware and software platforms

Open research challenges of these distributed and scalable approaches to improve dependability of heterogeneous, 2D and 3D MPSoC are: Meeting soft and hard real-time requirements during task migration and communication virtualization in embedded applications; Interplay between the virtual communication layer and service guarantees provided by the underlying physical Network-on-Chip (NoC) interconnect structure; Characterizing thermal sensitivity of cores during runtime using learning approaches; Identification of suitable techniques for thread shadowing considering OS interactions and memory accesses. As a prerequisite it is necessary to examine the basic thermal properties of our system

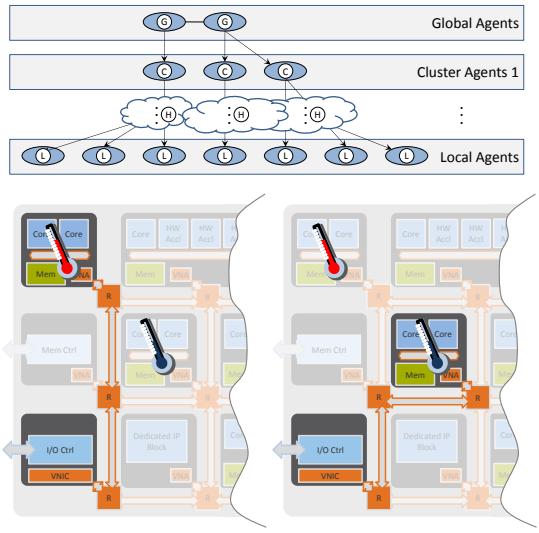

Figure 4: VirTherm-3D: Agent-based thermal management with hardware-assisted I/O virtualization

and to calibrate on-chip thermal sensors. This is done using an infrared thermal camera, with our setup shown in Fig. 5.

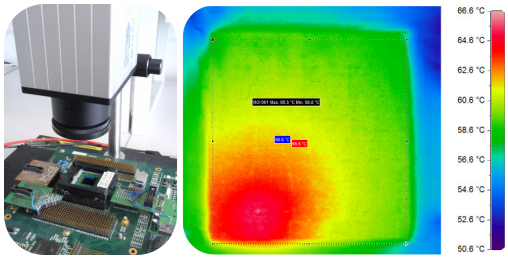

Figure 5: Infrared camera setup and thermal image of Xilinx Virtex-5 FPGA with computation in lower left corner

4. RESILIENT REAL-TIME OS

The operating system (OS) plays a key role in any complex computing system, especially in real-time systems. A current OS supporting software integration with memory management and virtualization contains several core functions that depend on error-free hardware. Errors in these functions quickly and irreversibly propagate through the system making it virtually impossible to recover from a function failure. Other OS functions can recover from failures with appropriate mechanisms. Such functions inherit the dependability requirements of the applications using it. Therefore, it is necessary to integrate operating system and hardware mechanisms that utilize the hardware and communication resources of a many-core system to efficiently provide the required dependability. The requirements to the design of a resilient OS are: to (1) identify the critical core functionality, (2) to minimize the hardware and software resources needed for the core, (3) to establish interfaces and signalling between hardware, operating system, and applications so as to provide system integrity which shall be guaranteed by corresponding formal safety analysis, and (4) to extend the underlying hardware architecture to provide the necessary fault handling mechanisms. These are the major concerns of the ASTEROID [57] project, which targets at an efficient trade-off between hardware and software measures to tackle unreliable hardware.

4.1 Error Detection and Recovery

Two basic mechanisms are required to harden an operating system and thus the mapped applications against errors of the underlying hardware: There must exist a framework to detect error conditions and in case the error already propagated in the software, there must be a mechanism to recover from that error.

The OS can for instance be executed on multiple cores which execute in a lock-step fashion. Here, both cores execute exactly the same instructions in parallel and a voting mechanism decides on correctness. This will protect all software running on the joint lock-step cores. However, for noncritical software this duplication imposes an unacceptable overhead. It is desirable to execute only those services redundantly which are critical, which is not possible using a plain lock-step mechanism. Instead a technique called fingerprinting, initially presented by Smolens et al. [49] and LaFrieda et al. [27] can serve this purpose. A small hardware fingerprint circuit hashes the execution stream in the processor-pipeline. Then, at any later time, the code can be executed on the same or another core and the voting is based on the fingerprint result.

To recover from errors, checkpointing is usually used, in which the system-state is saved on a volatile medium (checkpointed) on a regular basis. When recovery is necessary, a recent checkpoint state can be restored. This appealing concept of fault-tolerance in time has high pervasiveness in fault-tolerant systems because it can be implemented in software, hardware, or a combination thereof.

4.2 The role of Safety Standards

Deployment of embedded systems in safety-critical domains enforces a strongly safety oriented product life-cycle to qualify a product for deployment in safety-critical missions. This is not only crucial in order to minimize the risk of casualties in case of system failure but also to handle liability issues. To unify safety requirements, safety standards such as the industrial-oriented IEC-61508 [23] specify a safety certification process.

Of special interest in this scope is how to integrate applications with both safety and timing requirements. It is especially challenging because techniques from real-time and dependability domains need to be combined.

The classical IEC 61508 is to identify safety-functions first. These are technical functions which are intended to achieve or maintain a safe state of the system. Secondly, a risk assessment identifies the safety integrity level (SIL) of each safety function. This SIL is a general starting point for system design and defines guidelines for fault-tolerance, testing and validation effort. As the underlying framework, the core of the operating system itself is obviously the most critical piece. Thus, it is advised to keep the core software lean (e.g. by using a microkernel) in order to decrease certification effort and reduce the error-proneness. Then, additional software services (e.g. noncritical, best-effort applications) can be added if the core enforces sufficient temporal and functional isolation.

From the real-time perspective, fault-tolerance concepts must be handled with care because the scheduling of replicas, creation of checkpoints, and the process of recovery may lead to deadline misses, because these mechanisms are

potentially unbounded in time. Thus it is not only sufficient to have a resilient operating system. But the additional computation load must be considered. Moreover, if an application depends on an OS service, the error detection and recovery procedure inherits the application deadline requirements. These timing requirements strongly impacts the overall system scheduling threatening system efficiency. Currently, we could not even prove correctness, as existing real-time analyses such as [21] are not applicable because they only represent the error-free case, so real-time analysis must be combined with probabilistic error modeling, as in [4].

4.3 Dependability across the Software Stack

As argued above, the operating system is a primary candidate to use and provide *software* measures to compensate for unreliable hardware. However, dependability in this respect is a nonfunctional concern that affects and depends on all parts of the system. Tackling it in a problem-oriented way by the operating system is an open challenge: (1) It is still unclear, which combination of software measures is most beneficial to compensate certain hardware failures – ideally these measures should be understood as a matter of configuration and adaptation. (2) To achieve overall dependability, the implementation of these measures, even though provided by the operating system, cannot be scoped just to the operating system layer – it inherently crosscuts the whole software stack. (3) To achieve cost-efficiency with respect to hardware and energy, the measures have, furthermore, to be tailored with respect to the actual hardware properties and reliability requirements of the application: The goal of the DanceOS project [60] is to provide dependability of software running on unreliable hardware by the *fine-grained* and *tailorable* application of (software-based) fault-tolerance techniques. [45]

This calls for the application of advanced software technology, in particular program analysis and aspect-oriented programming [50], to separate the *what* and *where* of software-based dependability in a reusable way, that is, to separate the implementation of the fault-tolerance concern from the functional parts of the software. Thereby, fault-tolerance measures can flexibly be applied to, for instance, the whole software system, only its most critical parts, or only those parts that are actually affected by malfunction of some concrete hardware instance. The application of these measures shall be possible in this realm across the complete software stack, including the application itself and the (tailored) embedded operating system [29].

5. RELIABLE EMBEDDED SOFTWARE

Software for embedded systems is constrained by a lack of resources like time, energy, or memory. One common design principle that increases the predictability of embedded software is to reduce the number of run-time decisions and replace these with static behavior. In fault-tolerant systems, this implies that *every error is corrected in the same way.* Since the occurrence of errors is in general unpredictable, this may result in missed deadlines and inefficiently utilized resources.

To overcome this problem, one can make use of the fact that not all errors have fatal consequences that lead to a crash or an otherwise unusable system. Some errors show no effect at all, while other errors only affect the *quality of service.*

This observation is the basis for constructing reliable, real-time and resource-aware embedded software for which several conflicting objectives exist. One of the most obvious conflicts is adhering to real-time constraints vs. the achieved quality of service. If strict adherence to real-time constraints is required, correcting non-critical errors is only feasible during the currently available idle time. If real-time requirements could be softened, e.g., for a video player, this would allow a system to also correct less critical errors even if this would introduce a certain amount of jitter in the display. Additional constraints, like the amount of energy available to a mobile device, may also influence the decision taken.

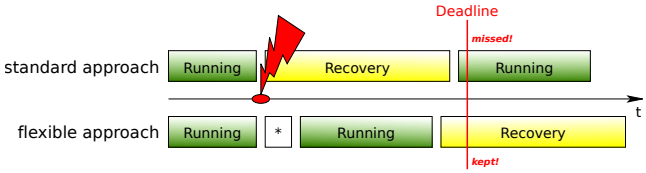

Figure 6: Separate Error Detection and Correction

5.1 Flexible Error Handling

In order to achieve flexibility, a system has to *classify* errors according to their impact and possess a detailed knowledge of the current timing and resource conditions of the system and the possible correction methods. Essential for flexible error handling is the separation of *error detection* and *error correction*, as shown in Fig. 6. When an error is signaled, error detection (∗) only has a short amount of time to decide *if*, *how*, and *when* to handle that error based on its impact and the current timing and resource conditions of a system. Regular system operation can then commence to keep deadlines until the error correction takes place.

```
1 // When affected by a fault,
2 // no correction required.
3 unreliable int u;
4
5 _Pragma( "Correction method=rollback" )
6 int critical_function( reliable int );
```

Figure 7: Type Qualifiers

In general, it is hard to assess the impact of an error on a system. Error detection usually only provides information about the affected memory address or hardware component. However, no information on the semantics of the error, e.g., which data is affected, is provided. Here, a specific property of embedded systems helps to obtain the required semantics. In most embedded systems, the set of applications is known in advance. This allows to perform a part of the error impact analysis at compile-time, based on application annotations. An example is shown in Fig. 7. There, "reliable"/"unreliable" qualifiers indicate if errors affecting to the annotated variable have to be corrected or an unexpected change of the value will not have a severe impact.

5.2 Reliability Annotations

Manually annotating every variable of a program is tedious. Compiler-based static methods extract *application knowledge* by analyzing the control and data flow to determine where values marked reliable or unreliable may show up during execution. In the example of Fig. 8, the value of the *reliable* variable `r` is copied to one of two different vari-

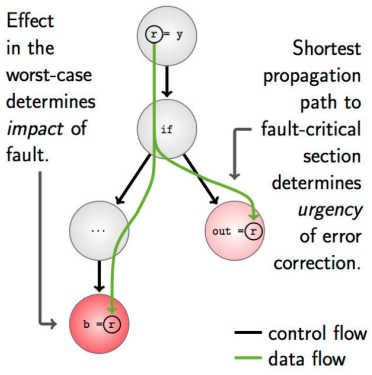

Figure 8: Tracking Reliability in the CDFG

ables, depending on the control flow. From the graph, the *impact* and *urgency* of error correction can be derived.

Reliability annotations provide an additional benefit when mapping the components of an application onto a complex architecture. Future architectures will contain components that exhibit varying dependability characteristics, like processor cores with different arithmetic precision and memories with different reliability guarantees. Using reliability annotations, the compiler can create a mapping that ensures the assignment of reliability-critical code and data objects to the most reliable components of a system.

5.3 Challenges

The temporal behavior of many applications is hard to predict and in many cases it is data-driven. For example, in an H.264 video decoder [19, 20], the time required for decoding a specific video frame is hard to determine. Thus, the idle time available for error correction is not known beforehand. Here, heuristic models [43] may help to obtain useful estimates. These heuristics, however, do not guarantee an upper bound for the timing, so in rare cases, using this information might introduce jitter into the system [17]. This fits well to the flexible trade-offs considered in the project.

Classification of error impact is application-dependent; no single classification can cover all possible applications. Only the extreme cases, program crashes and no effect at all, are common to all applications. In addition, the system designer must consider use cases. In a video display, e.g., a discolored pixel may be acceptable for consumer electronics, but not when using the same code for safety-critical applications.

An additional challenge is that the amount of compiler-generated data required for the run-time decisions may potentially be huge. Thus, efficient consolidation and compact representations of that data to enable fast accesses are required so that the run-time overhead is minimized.

Another important challenge is the tailoring of reliability-increasing techniques on the software-level to the system's architecture. This requires application-driven reliability analysis techniques capable of considering both hardware and software aspects cross-layer. Within this SPP, existing application-driven reliability analysis techniques like [31], that are restricted to hardware aspects, are extended accordingly.

5.4 Projects within this topic

The FEHLER [61] and CRAU [65] projects view the resource and real-time requirements from different perspectives. While FEHLER treats real-time conditions of a system as being of primary importance, the CRAU project takes a resource-centric approach to assess the error correction capabilities of a system composed of multiple components and applications.

Both projects meet in the middle. The compile-time processing in FEHLER generates reliability information for code and data objects of the given application. This information can be used at run-time to implement flexible error handling and at design-time to create a mapping of objects with different reliability *requirements* to architecture components with different reliability *characteristics*.

Overall, flexible error handling that employs reliability annotations in the FEHLER project together with the compositional view of component and system reliability provided by the CRAU project will enable future embedded system designers to explore the design space of a system under given constraints for reliability and additional resources.

6. RESILIENT ARCHITECTURES

Exploring the knowledge of multiple abstraction layers from circuit and micro-architecture up to algorithm and application layer is key to minimize dependability cost in terms of area, energy and performance. Until today such a *cross-layer design approach* has mostly been addressed at the lower design levels. At higher levels, systems were designed under the premise of fault free underlying hardware. Thus, a large body of related work focuses on low level techniques of presenting a practically error-free platform built from potentially unreliable elements to higher abstraction and design levels[35].

The majority of robust systems assume the co-existence of high-reliability components, based on conservative design with corresponding implementation overhead, and low-reliability components. This *differential* or *asymetric* reliability is exploited in various ways to protect individual system components like buses, network-on-chip, memories, datapaths etc. against transient or permanent errors. This is performed by employing built-in redundancy or built-in self-recovery techniques. All these techniques basically comprise information and/or execution redundancy. Examples are checker processors, ECC and residue checking, Razor FFs and BISER Latches just to name a few [34]. An error resilient hardware architecture which can be seen as practically error free can then be composed of protected components. Applications on these platforms can be implemented in a traditional way, still assuming fault free operation of the underlying hardware.

In addition to this *horizontal integration*, recent research is also evaluating the additional potential of a *vertical integration* of error resilience on the application level with platforms with a reduced reliability. True cross-layer optimization approaches do not only exploit the fact that some important classes of algorithms have inherent error resilience, but also adapt the applications and architectures jointly to achieve the best trade-offs.

A large number of important and relevant applications have some type of inherent error resilience. They can be attributed to one of the two following categories:

- *Algorithmic resilience* is given when a certain amount of errors can be tolerated by the algorithm itself. This is the case for probabilistic applications and applications which can tolerate statistical behavior. Examples are Recognition, Mining and Synthesis (RMS) applications [12] and wireless systems. Further applications in this context are fixed point DSP and numerical algorithms.

- *Cognitive resilience* stems from the interaction of an application with a human being like in audio and video processing. Here, errors are tolerable as long as the user cannot discern quality differences, or accepts them as trade-offs for a longer battery life-time, for example.

Taking into account this application resilience at the architectural level opens the door for further strong reduction of the architectural overhead for resilience. The potential for such approaches has been recently shown by various researchers. Examples are algorithmic noise tolerance (ANT) [46], significance driven approach (SDA) [36], probabilistic CMOS [39], error-resilient system architecture (ERSA) for RMS applications [28, 9], architectures for multimedia [7] and wireless communications [25, 33, 32, 3]. It is important to mention that the exploitation of application resilience is not limited to hardware implementations but can also be applied to software implementations.

6.1 Project within this topic

We investigate the aforementioned cross-layer-reliability approach using a wireless baseband transmission system as application [66].

A multiple-input, multiple-output (MIMO) detector is taken as demonstrator vehicle. Such a MIMO detector could be considered as an IP block in the tiled many-core Architectures shown in Fig. 3. Special focus is put on *adaptive reliability* tuned by the application at run-time according to the required system performance and oberved disturbances and operating conditions. Wireless communication systems are an excellent application to investigate such an approach [2].

While run-time adaptivity is common in todays communications standards, these techniques have never been applied to increase system error resilience with respect to hardware failures. The Quality-of-Service (QoS) in wireless systems is typically defined as the bit error rate with respect to a given signal-to-noise (SNR) ratio. In current standards, like HSPA or LTE, these service parameters and the desired system throughput are dynamically adjusted at runtime, e.g., higher throughput rates are specified only for higher SNR. This is due to the fact that the computational requirement on the different algorithms is decreasing with higher SNR. But in future, the negotiated QoS may also depend on the reliability of the receiver hardware under given operation conditions.

This leads to an entirely new paradigm – adaptive QoS with respect to communications reliability *and* implementation reliability. To illustrate this, instead of providing higher throughput at high SNR, relaxed reliability requirements on the underlying hardware are possible. In other words, at high SNR the error-resilience requirements can be relaxed for the same QoS. In this way, QoS, hardware reliability, and implementation efficiency can be traded-off against each other.

7. CONCLUSION

We would like to conclude with a view from Dr. Sani Nassif, IBM (July 2011): *"It is clear that without significant innovation, the reliability of deeply scaled integrated circuits will worsen to the point where it might no longer make sense to continue scaling. Trends that have been present for several generations in SRAM, which uses the smallest and most dense devices on a typical integrated circuit, will likely broaden to include normally robust circuits such as latches and possibly even logic gates. A lot of innovations at the technology, circuit, micro-architecture and system levels were introduced to handle the unreliability of SRAM, like redundancy, error correction, 8-transistor cell structures and the like. New innovations at an even wider scale will be needed to create reliable systems in the future, and the challenges are not solvable at any one level alone, but will require deep cooperation between hardware, software and system designers"* [1].

We consider it of paramount importance to involve several layers of abstraction when designing dependable embedded hardware/software systems. Experts from hardware and system-on-chip-design, system software and OS, and application specialists, among others, need to cooperate in order to ultimately achieve the required adaptability in order to build truly reliable future embedded systems.

This SPP 1500 priority program brings together researchers from these diverse areas to assess the behavior of future devices and invent new dependability solutions for future generations of embedded devices. The combined experience and expertise unique to this research program is aimed to enable embedded systems researchers to break with traditional approaches of embedded system design and architectures in what we call a 'dependability co-design.' This will enable us to comprehensively tackle dependability problems of future embedded systems and to develop innovative solutions to improve the dependability of embedded systems while incorporating the traditional optimization objectives common to embedded systems. The solutions developed in the context of this program take a holistic view on dependability problems and consider them as combined hardware/software challenges.

8. ACKNOWLEDGEMENT

This research program is supported by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu). We would like to thank Dr. Sani Nassif (IBM) for in-depth discussions during the SPP 1500 meeting in July 2011. Furthermore, we would like to thank Philip Axer (TU Braunschweig), Thomas Ebi (KIT), and Holm Rauchfuss (TUM) for support in preparing this paper.

9. REFERENCES

[1] Sani Nassif during the SPP 1500 meeting in Stuttgart, Germay, July 2011.

[2] Designing Chips without Guarantees. *Design & Test of Computers, IEEE*, 27(5):60–67, 2010.

[3] R. A. Abdallah and N. R. Shanbhag. Error-Resilient Low-Power Viterbi Decoder Architectures. *Signal Processing, IEEE Transactions on*, 57(12):4906–4917, 2009.

[4] Philip Axer, Maurice Sebastian, and Rolf Ernst. Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints. In *Proc. of Int. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM symposium on Operating systems principles SOSP '03*, pages 164–177, 2003.

[6] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, 2005.

[7] M. A. Breuer. Multi-media applications and imprecise computation. In *Proc. 8th Euromicro Conference on Digital System Design*, pages 2–7, 2005.

[8] S. Chinni and R. Hiremane. Virtual machine device queues. 2007.

[9] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, pages 555–560, 2010.

[10] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton. Dynamic voltage scaling for commercial FPGAs. In *ICFPT, 2005*, pages 173–180, 2005.

[11] Ayse Kivilcim Coskun, Tajana Šimunic Rosing, Keith A. Whisnant, and Kenny C. Gross. Static and dynamic temperature-aware scheduling for multiprocessor SoCs. *IEEE Trans. Very Large Scale Integr. Syst.*, 16:1127–1140, 2008.

[12] P. Dubey. Recognition, Mining and Synthesis Moves Computers to the Era of Tera. *Technology@Intel Magazine*, pages 1–8, 2005.

[13] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. Economic learning for thermal-aware power budgeting in many-core architectures. In *Proc. 9th Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[14] Thomas Ebi, Holm Rauchfuss, Andreas Herkersdorf, and Jörg Henkel. Agent-based thermal management using real-time I/O communication relocation for 3D many-cores. In *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2011.

[15] S. Eisenhardt, A. Küster, T. Schweizer, T. Kuhn, and W. Rosenstiel. Runtime datapath remapping for fault-tolerant coarse-grained reconfigurable architectures. In *International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2011.

[16] S. Eisenhardt, A. Küster, T. Schweizer, T. Kuhn, and W. Rosenstiel. Spatial and temporal data path remapping for fault-tolerant coarse-grained reconfigurable architectures. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2011. accepted to be published.

[17] Michael Engel, Florian Schmoll, Andreas Heinig, and Peter Marwedel. Temporal Properties of Error Handling for Multimedia Applications. In *Proceedings of the 14th ITG Conference on Electronic Media Technology*, 2011.

[18] European Nanoelectronics Initiative Advisory Council. Eniac strategic research agenda - european technology platform nanoelectronics. *Second Edition*, 2007.

[19] Andreas Heinig, Michael Engel, Florian Schmoll, and Peter Marwedel. Improving Transient Memory Fault Resilience of an H.264 Decoder. In *Proceedings of the Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia)*, 2010.

[20] Andreas Heinig, Michael Engel, Florian Schmoll, and Peter Marwedel. Using Application Knowledge to Improve Embedded Systems Dependability. In *Proceedings of the Workshop on Hot Topics in System Dependability (HotDep)*, 2010.

[21] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the SymTA/S approach. *IEE Proceedings Computers and Digital Techniques*, 2005.

[22] W.-L. Hung, G. M. Link, Yuan Xie, N. Vijaykrishnan, and M. J. Irwin. Interconnect and thermal-aware floorplanning for 3d microprocessors. In *Proceedings of the 7th International Symposium on Quality Electronic Design (ISQED)*, pages 98–104, 2006.

[23] International Electrotechnical Commission (IEC). Functional safety of electrical / electronic / programmable electronic safety-related systems, 1998.

[24] Phillip H. Jones, Young H. Cho, and John W. Lockwood. Dynamically optimizing FPGA applications by monitoring temperature and workloads. In *VLSI Design. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, pages 391–400, 2007.

[25] A. Khajeh, Minyoung Kim, N. Dutt, A. M. Eltawil, and F. J. Kurdahi. Cross-layer co-exploration of exploiting error resilience for video over wireless applications. In *Proc. IEEE/ACM/IFIP Workshop Embedded Systems for Real-Time Multimedia ESTImedia*, pages 13–18, 2008.

[26] Veit B. Kleeberger, Sebastian Kiesel, Ulf Schlichtmann, and Samarjit Chakraborty. Program-Aware Circuit Level Timing Analysis. In *International Symposium on Integrated Circuits (ISIC)*, 2011. To appear.

[27] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *Proc. of Int. Conf. Dependable Systems and Networks*, pages 317–326, 2007.

[28] L. Leem, Hyungmin Cho, J. Bau, Q. A. Jacobson, and S. Mitra. ERSA: Error Resilient System Architecture for probabilistic applications. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1560–1565, 2010.

[29] Daniel Lohmann, Wanja Hofer, Wolfgang Schröder-Preikschat, Jochen Streicher, and Olaf Spinczyk. CiAO: An aspect-oriented operating-system family for resource-constrained embedded systems. In *Proceedings of the USENIX Annual Technical Conference*, pages 215–228, 2009.

[30] Enno Lübbers and Marco Platzner. ReconOS: Multithreaded programming for reconfigurable computers. *ACM Trans. Embed. Comput. Syst.*, 9:8:1–8:33, 2009.

[31] M. Glaß, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich. Symbolic system level reliability analysis. In *Proceedings of the 2010 International Conference on Computer-Aided Design (ICCAD)*, pages 185–189.

[32] M. May, M. Alles, and N. Wehn. A Case Study in Reliability-Aware Design: A Resilient LDPC Code Decoder. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 456–461, 2008.

[33] M. May, N. Wehn, A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, D. Ziener, and J. Teich. A Rapid Prototyping System for Error-Resilient Multi-Processor Systems-on-Chip. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 375–380, 2010.

[34] S. Mitra, K. Brelsford, Young Moon Kim, Hsiao-Heng Kelin Lee, and Yanjing Li. Robust System

Design to Overcome CMOS Reliability Challenges. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(1):30–41, 2011.

[35] S. Mitra, K. Brelsford, and P. N. Sanda. Cross-layer resilience challenges: Metrics and optimization. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, pages 1029–1034, 2010.

[36] Debabrata Mohapatra, Georgios Karakonstantis, and Kaushik Roy. Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, ISLPED, pages 195–200, 2009.

[37] Gordon E. Moore. No exponential is forever: but "forever" can be delayed! [semiconductor industry]. In *Solid-State Circuits Conference. Digest of Technical Papers (ISSCC)*, pages 20–23, vol.1, 2003.

[38] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(12):1870 –1882, 2009.

[39] Krishna V. Palem. Energy aware algorithm design via probabilistic computing: from algorithms and models to moore's law and novel (semiconductor) devices. In *Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems*, CASES, pages 113–116, 2003.

[40] D.K. Pradhan. *Fault-tolerant computer system design*. Prentice-Hall, Inc., 1996.

[41] Semeen Rehman, Muhammad Shafique, Florian Kriebel, and Jörg Henkel. Reliable software for unreliable hardware: Embedded code generation aiming at reliability. In *Proc. 9th Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[42] Semeen Rehman, Muhammad Shafique, Florian Kriebel, and Jörg Henkel. ReVC: Computationally reliable video coding on unreliable hardware platforms: A case study on error-tolerant H.264/AVC CAVLC entropy coding. In *Proc. 18th International Conference on Image Processing (ICIP)*, 2011.

[43] Michael Roitzsch and Martin Pohlack. Video quality and system resources: Scheduling two opponents. *J. Vis. Commun. Image Represent.*, 19:473–488, 2008.

[44] B. Sander, J. Schnerr, and O. Bringmann. ESL power analysis of embedded processors for temperature and reliability estimations. In *Proc. 7th Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 239–248, 2009.

[45] Horst Schirmeier, Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk. DanceOS: Towards dependability aspects in configurable embedded operating systems. In *Proceedings of the 3rd HiPEAC Workshop on Design for Reliability (DFR)*, pages 21–26, 2011.

[46] Naresh R. Shanbhag, Rami A. Abdallah, Rakesh Kumar, and Douglas L. Jones. Stochastic computation. In *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, pages 859–864, 2010.

[47] S.K. Shukla and R.I. Bahar. *Nano, quantum and molecular computing: implications to high level design and validation*. Solid Mechanics and Its Applications Series. Kluwer Academic Publishers, 2004.

[48] D.P. Siewiorek and R.S. Swarz. *Reliable computer systems: design and evaluation*, volume 2. Digital Press, 1992.

[49] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatryk. Fingerprinting: bounding soft-error-detection latency and bandwidth. 24(6):22–29, 2004.

[50] Olaf Spinczyk and Daniel Lohmann. The design and implementation of AspectC++. *Knowledge-Based Systems, Special Issue on Techniques to Produce Intelligent Secure Software*, 20(7):636–651, 2007.

[51] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98, 1956.

[52] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A.L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 306–317. Citeseer, 2007.

[53] Xiuyi Zhou, Jun Yang, Yi Xu, Youtao Zhang, and Jianhua Zhao. Thermal-aware task scheduling for 3d multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, 21:60–71, 2010.

— SPP1500 - http://spp1500.itec.kit.edu/ —

[54] Joachim Becker. Runtime Reconfigurable Analog Circuits and Adaptive Filter Synthesis for Compensation of Unreliable Hardware Constraints (hexFPAA).

[55] Uwe Brinkschulte and Lars Hedrich. MixedCoreSoC – A Highly Dependable Self-Adaptive Mixed-Signal Multi-Core System-on-Chip (MixedCoreSoC).

[56] Samarjit Chakraborty and Ulf Schlichtmann. Lifting Device-Level Characteristics for Error Resilient System Level Design: A Crosslayer Approach (LIFT).

[57] Rolf Ernst and Hermann Härtig. ASTEROID - An Analyzable, Resilient, Embedded Real-Time Operating System Design (ASTEROID).

[58] Jörg Henkel and Andreas Herkersdorf. VirTherm-3D Communication Virtualization Enabling Thermal Management for Dependable 3D Many-Cores (VirTherm-3D).

[59] Jörg Henkel and Hans-Joachim Wunderlich. OTERA: Online Test Strategies for Reliable Reconfigurable Architectures (OTERA).

[60] Rüdiger Kapitza, Daniel Lohmann, and Olaf Spinczyk. Dependability Aspects in Configurable Embedded Operating Systems (DanceOS).

[61] Peter Marwedel and Michael Engel. Software-Based Error Handling Using Cooperation Between Compilers and Operating Systems (FEHLER).

[62] Marco Platzner. Temperature-driven Thread Mapping and Shadowing in Hybrid Multi-Cores (SMASH).

[63] Wolfgang Rosenstiel. Self-Adaptive Coarse-Grained Reconfigurable Architectures as Reliability Enhancers in Embedded Systems (ARES).

[64] Mehdi Tahoori. Providing Efficient Reliability in Critical Embedded Systems (PERCEDES).

[65] Jürgen Teich. Compositional System Level Reliability Analysis in the Presence of Uncertainties (CRAU).

[66] Norbert Wehn. Design of Efficient, Dependable VLSI Architectures Based on a Cross-Layer-Reliability Approach Using Wireless Communication as Application (MIMODeS).