Design and Run-time Reliability at the Electronic System Level

Björn Sander,† Andreas Bernauer‡ and Wolfgang Rosenstiel†,‡

The ongoing scaling of CMOS technology facilitates the design of systems with continuously increasing functionality but also raises the susceptibility of these systems to reliability issues. These can for example be caused by high power densities and temperatures. At the moment it is still possible to cope with the posed challenges in an affordable manner. But in the future, a combination of design and run-time measures will become necessary in order to guarantee that reliability guidelines are met. Because of complexity reasons, the Electronic System Level (ESL) is gaining importance as starting point of design. Design alternatives are evaluated at ESL with respect to several design objectives, lately also including reliability. In this paper, the most important phenomena threatening the reliability are introduced and the current status of related research work and tools is presented. After that, a high level design space exploration considering performance, energy and reliability trade-offs in multi-core systems is introduced. Finally, it is shown how reliability can be further improved during run-time by the application of a machine learning system.

1. Introduction

Embedded processors are an integral part of System-on-Chip (SoC) designs, which offer lower power consumptions, higher performance and simpler system integration in comparison with other design styles. However, due to the continuing scaling of silicon technologies, it is becoming increasingly difficult for manufacturers to fulfill the expectations of their customers with respect to the reliability of the products. This is because decreased feature sizes not only lead to higher clock frequencies, lower supply voltages and smaller die sizes, but also cause some serious problems. Changing electrical properties of the circuits, the susceptibility to internal and external noises and an accelerated aging particularly pose great challenges. The accelerated aging is a consequence of higher on-chip temperatures which result from higher power densities as well as of thermal cycles. Various failure mechanisms, like e.g., electromigration or time-dependent dielectric breakdown, occur earlier after initial operation. Overall, this leads to a reliability decrease. Because a further intensification of the aforementioned problems has to be expected, the successful incorporation of reliability during design and operation will become one requirement for a promising market position.

Especially the automotive domain combines the embedded system challenges like stringent energy budgets, increasing performance demands, long operation times as well as unfriendly working environments. The first two points are a reason for the trend in automotive industry to reduce the number of control units by the use of multi-core systems. The last-mentioned points highlight the need of a fast reliability analysis allowing the derivation of substantiated statements about the behavior of the electronics contained in the cars.

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 – methods at higher levels of abstraction are no exception. To obtain component temperatures, their power consumptions have to be known which in turn are derived from activity information. In order to obtain the activity information in low level methods, worst case behavior of e.g., gates or transistors is often assumed, because actual utilization data is not available. For this reason, the assembled system might be overdesigned and costly at the end. Additionally, it is hardly possible to consider large systems because of complexity reasons or even to take the environment of the system into account. For the automotive domain the temperature of a motor compartment can serve as an example environment. At ESL, the characteristics of real applications are available which allows optimizing the design specifically for the intended use case. Environmental conditions as well as end-user behavior can also be incorporated. On the other hand, the comprehensive representation of physical effects, which is fundamental for ESL reliability analyses to become widely accepted, is still a field of research. Bridging the gap between target ap-
Applications on one side and the correct modeling of physical effects on the other will become one of the key tasks for the system design community in order to provide reliable products at reasonable cost.

The remainder of this paper is organized as follows: In Section 2, the phenomena threatening the reliability of semiconductor systems as well as the current status of related research work and tools are introduced. Section 3 presents a high level design space exploration considering performance, energy and reliability trade-offs in multi-core systems. An autonomic run-time control of an SoC under changing conditions is introduced in Section 4. Finally, Section 5 concludes the paper.

2. Threats for Reliability

2.1 Phenomena

The typical developing of failure rates of semiconductor devices over time can roughly be divided into three phases, forming the so called bathtub curve: Early life period, constant failure period and wear-out failure period (see Fig. 1). Faults that cause the breakdown of a circuit during early life period are almost always rooted in process variabilities during fabrication. Because faults of this class often occur in a short period of time after initial operation, the probability of a breakdown decreases per time interval. In other words, the failure rate decreases in the early life period steadily, until it can be considered as constant. Process variabilities are discussed in greater detail in Section 2.1.1.

Failures during the constant failure period are called random, because of the multiplicity of reasons for their occurrence. However, important failure sources are unwanted bit flips in storage devices and registers because of charge disruption in corresponding transistors caused by alpha particles, also called transient faults. These are discussed in Section 2.1.2.

In consequence of increasing material fatigue, the failure rate rises after a longer operation time – the wear-out period begins (see Section 2.1.3).

It has to be pointed out, that the bathtub curve is related to a statistical average of a large set of devices. The derivation of conclusions for a certain device is not possible.

2.1.1 Process Variabilities

Mainly because of random dopant fluctuations and sub-wavelength lithography, manufacturing semiconductor devices with feature sizes of 65 nm and below has already led to a serious increase of process variabilities. Prominent examples are variations of the transistor’s channel lengths, the insulator thickness between gate and transistor channel or the threshold voltages (7),8).

Among other things, the consequences of these variations are violations of deadlines because of unpredictable critical paths or leakage current increases (2).

To obey certain design constraints, often worst case methodologies are still applied. Because already today chips of the same wafer reveal great variations with respect to the above mentioned properties, it is essential to adapt today’s design flows to handle these phenomena (9),10).

State-of-the-art techniques are targeting specifically a zero-defect manufacturing. However, an ongoing application of this approach would forbid further scaling steps due to an unacceptable yield decrease. Therefore, the design for yield paradigm came up, moving away from functional perfection.

2.1.2 Transient Faults

Transient faults or so called soft errors are expected to gain significant importance in the upcoming process nodes. Because of lower voltages and smaller switching capacitances, the amount of charge representing one bit is decreasing with every new CMOS technology. This in turn increases the susceptibility of storage devices and flip-flops to soft errors (11),12).

Soft error detection/correction techniques are already well-known for memory...
Design and Run-time Reliability at the Electronic System Level

By parity checking and error correcting codes, it is relatively easy to detect and correct these errors in memories. But dealing with transient faults occurring in combinational logic and flip-flops is more difficult.

The error rate per logic state bit increases by about 8% with each new technology generation. Because the number of logic state bits on a chip doubles roughly each technology generation, soft-error failure rates of chips will rise steeply. For the 16 nm generation, the failure rate is expected to be almost 100 times that at 180 nm. Moreover, future technologies will become more susceptible to induced soft errors in logic and interconnect components.

2.1.3 Thermal Effects/Aging

The point in time, at that an aging defect occurs in a microchip because of material fatigue, depends largely on the temperature it is exposed to during operation. The temperature in turn is a function of the power density that increases with every new technology node. This leads to an earlier appearance of aging defects.

Beside transistor sizes and clock frequencies, leakage current, which flows through a transistor even when it does not switch, plays a more and more dominant role with respect to the power density.

In Ref. 18 it is reported that the transition from the 180 nm to the 65 nm technology has increased the peak temperatures about 15°C. As a consequence, the average time until the first appearance of aging defects has decreased about 70%.

Important aging effects are:

- Electromigration: Electromigration in aluminum and copper interconnects is a mass transport of conductor metal atoms due to momentum transferred by the electron current causing opens and shorts.

- Stress Migration: This is a phenomenon where the metal atoms in the interconnect migrate due to mechanical stress. Stress migration is similar to electromigration.

- Time Dependent Dielectric Breakdown: Time dependent dielectric breakdown is an extremely important failure mechanism in semiconductor devices. With time, the gate dielectric wears down and fails, when a conductive path is formed.

- Thermal Cycling: Fatigue failures can occur in semiconductors due to temperature cycling and thermal shock. Permanent damage accumulates every time there is a cycle in temperature, eventually leading to failures.

- Hot Carrier Injection: Hot carrier injection describes the phenomena by which carriers gain sufficient energy to penetrate into the gate oxide, resulting in transistor parameter degradation, e.g., switching frequency reduction.

- Negative Bias Temperature Instability: Positive charges get stuck in the Si-oxide/Si-crystal lattice boundary because of negative bias stress, shifting the threshold voltage.

2.2 Research Work and Tools

2.2.1 Technology Abstraction

Abstraction of technology measures is essential for optimization processes in chip design flows. It is an ongoing task, since changes in technology have to be taken into account.

One prominent example is the power estimation at higher levels of abstraction. Being relatively mature (see e.g., Ref. 33), it has gained importance again, since the former assumption, that the power only depends on the switching activity of the corresponding device, was no longer acceptable.

For other system properties, like temperature, the adequate abstraction level still isn’t found and probably depends largely on the concrete scenario. FireBolt is one of the few commercial solutions for IC thermal analysis. The vendor quotes several reasons for the need of high resolution temperature estimation, e.g., the poor thermal conductance of insulator materials and sees the field of application for its tool in the front-end as well as in the back-end stages of semiconductor design flows. On the other hand, HotSpot, an academic tool developed for temperature analysis and optimization, is particularly intended for the microarchitectural level.

2.2.2 Dependable Architecture

Formulations of requirements that have to be accomplished by future hardware architectures to master the challenges posed by the nano age can be found in Refs. 38, 39.

In Ref. 40 it is emphasized that positive effects of multi-core platforms in sys-
tem design, like power reductions or performance gains, come at a cost. Because many on-chip components share resources, like caches or I/O interfaces, the isolation of faults is becoming a challenging task.

Efforts have already been made to mitigate the forecasted soft error increase. The proposed approaches can be distinguished with respect to the considered abstraction level and the used redundancy.

Nicolaidis uses so called shadow latches to protect combinational logic\cite{41,42}. A shadow latch is a redundant latch that is clocked out of phase. The outputs of the main latch and the redundant latch(es) are either compared or voted, depending on whether the goal is only to detect an error or to tolerate it. Razor is another method for detecting soft errors in combinational logic\cite{43,44}. The key idea of Razor is to tune the supply voltage by monitoring the error rate during circuit operation, thereby eliminating the need for voltage margins and exploiting the data dependence of circuit delay. A Razor flip-flop that double-samples pipeline stage values is introduced, once with a fast clock and again with a time-borrowing delayed clock. A robust comparator then validates latch values sampled with the fast clock. In the event of a timing error, a modified pipeline miss-speculation recovery mechanism restores the correct program state. A dynamic microarchitectural verification technique, called DIVA, was introduced in Ref.\cite{45}. The approach works by augmenting the commit phase of the processor pipeline with a functional checker unit. The functional checker verifies the correctness of the processor’s computation, only permitting correct results to commit. In Refs.\cite{46} and \cite{47} an approach that combines transient and timing error protection with a reliability run-time evaluation of the corresponding hardware is presented. In order to reduce the hardware overhead of error detection/correction methods relying on duplication and comparison, so called self-checking circuits were proposed\cite{48,49}. In such designs, outputs delivered by functional blocks belong to error detecting codes. Checker units, monitoring these codes, perform the concurrent error detection. Coding algorithms have emerged as a promising solution to achieve power and delay reduction in interconnects. In order to additionally provide error correction, several methods have been introduced recently\cite{50,51}.

2.2.3 Autonomic Computing

Keeping SoC design effort at a constant level is a difficult task, as integration density is increasing and time to market is shrinking. To manage the increasing design complexity, the ITRS estimates a requirement of a design reuse rate of 70\% until 2015 and 90\% until 2020\cite{52} [System Drivers Chapter]. Not only does increasing integration density pose a design challenge, but also increasing transistor variability\cite{53,54}, process variation\cite{55} and degradation effects\cite{56} add to the design complexity. These challenges make it more and more difficult for manufacturers to produce reliable chips\cite{5}.

Furthermore, traditional worst-case design becomes less favorable for nano-scale chips, as the worst case design requires margins that would use most of the benefits of the next technology node\cite{2}. More favorably are better-than-worst-case design\cite{57}, where the chip design is intended for the typical case and special circuits handle the presumably rare worst cases.

As a special case, special circuits that enable the chip to self-adapt individually to its actual variability, variation and degradation allow each individual chip to reach its maximum performance. In essence, self-adaptation moves former design-time decisions into the run time of a chip, considering that at design time some actual physical parameters are not known. For example, in Ref.\cite{58} the authors adjust the clocking of the pipeline stages after chip fabrication to compensate for process variability, and in Ref.\cite{59}, the supply voltage of an SRAM is adjusted at run time to balance the error rate and power consumption. The obvious advantages of run-time decisions at chip level are short reaction times, and adaptations that are transparent to the operating system. Furthermore, for many error mechanisms, run-time decisions at software level are inadequate, as the underlying hardware is erroneous.

However, current self-adaptation circuits such as\cite{58,59} are special-purpose. Although they help to realize better-than-worst-case designs, they also increase the design effort and prolong the design process, as they require trained engineers who must consider all possible situations and craft an individual solution to the self-adaptation problem. Furthermore, the individual solutions have a low design reuse rate, as they cannot be easily reused for a similar or different problem. The designer of nano-scale SoCs is thus forced to choose between poor chip performance based on presumably quick worst-case design, and improved chip performance based on presumably long better-than-worst-case design using
Design and Run-time Reliability at the Electronic System Level

Fig. 2 Generic self-adaptation system. The evaluator receives $k$-bit wide monitor input from $n_k$ monitors and outputs $l$-bit wide actuator input to $n_l$ actuators. The evaluator tries to keep the controlled unit at the operating point with the highest utility function (not depicted).

special-purpose self-adaptation circuits.

Our generic self-adaptation circuit and method offers a middle ground between a poor chip performance and a prolonged design process. A generic self-adaptation system offers a high design reuse rate for better-than-worst-case designs. In the following, we determine the requirements of a generically applicable self-adaptation system and present our self-adaptation method, which is based on the reinforcement algorithm XCS.

As any other self-adaptation system, a generic self-adaptation system will provide an input and output interface as depicted in Fig. 2: a monitor that provides the current state of the controlled unit, and an actuator that executes the self-adaptation action, which moves the controlled unit towards the desired state. For a reusable design, the bit vectors of the monitor and actuator must be model-free, that is, the generic self-adaptation system makes no assumption about the information that is carried by these signals. In practice, the input and output signal will be connected to $n_k$ and $n_l$ monitors and actuators, respectively, carrying $k$ and $l$ bits of actual data. The actual generic self-adaptation happens in the evaluator that reasons about the incoming state information and suggests a (at best optimal) action to move the controlled unit towards the desired state.

An evaluator that does not need individual manual instruction by a trained engineer has to autonomously learn the optimal actions by itself (which of course depend on the current situation). As learning does not come without making errors, this setup is only suitable for self-adaptation problems where errors are not fatal or where an additional guarding circuit prevents the error. Observing current design processes, we find that the designer can usually identify the desired system state, but does not exactly know how this state can be reached from a given system state (which is one of the reasons why special-purpose self-adaptation circuits prolong the design process considerably). Being a machine learning problem, we identify reinforcement algorithms to be the most suitable evaluators. The reinforcement algorithms can use the utility function of the designer (called reward function in machine learning terms) to favor desired system states. In particular, from the reinforcement algorithms, the temporal difference learners (TD) fulfill the aforementioned requirement that the evaluator makes no assumption on the input and output signals: temporal difference learners don’t require a model of the environment or the utility (reward) of the current state (they are model-free).

To choose from the various temporal difference learners, we note that if the designer happens to know the best action for a given situation, it is desirable that this information can be easily added to the evaluator. Not only does this save investments already taken to solve a particular design problem, but this also helps the learning of the evaluator. The learning classifier systems have the property to express the stored knowledge in form of human-understandable classifiers (or rules). The designer can thus add his expert knowledge to the learning evaluator and, furthermore, understand what the evaluator has learned so far. This is in contrast to other machine learning algorithms such as neural networks, where the acquired knowledge is opaque to the designer and not reusable.

From the learning classifier systems, the literature reports that XCS learns accurately and completely. Additionally, a hardware implementation of the XCS is available so that chip-level control is possible: in Ref. 66) the authors report a complete implementation of XCS on an FPGA, and in Ref. 67) report an area-optimized and fast (7.6 ns lookup period) hardware implementation that still retains the self-adaptation capabilities.

We therefore propose to use the XCS as the evaluator of a self-adaptation
method, that is reusable for various self-adaptation problems and avoids a prolonged design process. Our method covers both design and run time. At design time, the XCS learns based on a software simulation of the SoC. The resulting optimal set of classifiers is then loaded into the hardware implementation of the evaluator. The XCS will operate on its set of classifiers and self-adapt to the actual physical situation of the chip, leveling out differences between the software model and the actual chip. Our method combines the advantage of both software and hardware implementations: the software-based simulation has a lot of available resources and thus allows a fast and efficient learning of a large amount of classifiers, while the hardware-based system reacts quickly, transparently and independent of the running SoC-application.

We will shortly introduce the XCS in the following and refer to the literature 61), 64), 68) for details. Each classifier that the XCS learns consists of a condition, an action, a reward prediction and some further house keeping values. Condition and action are bit strings. The condition additionally uses a wild-card symbol that matches several cases at once. During each learning step, the XCS matches the conditions of all classifiers with the current monitor input signal. It notes the suggested actions and the associated reward predictions to choose the action which promises the highest reward. After the action has been applied to the SoC, the XCS receives a reward based on the utility function that assesses the new system state. The XCS uses this feedback to adjust its reward predictions and its classifier set. In the ideal case, the XCS learns a classifier set that accurately predicts the utility of any situation but, at the same time, is as small as possible 65). The XCS that is employed at design time uses a genetic algorithm to generate new classifiers. The genetic algorithm selects the parents of the new classifiers based on the reward prediction accuracy, a distinguishing feature of XCS and supposedly the reason for its generally good performance.

2.2.4 Design Methods and Tools

Several approaches have been proposed to address dependability in current design flows at different levels of abstraction. At the system level, most of the approaches use design decisions, like the configuration of scheduling or power management policies or the mapping of functionality onto components, to reduce the influence of aging effects. For this, the activities of system components are usually used to estimate their power consumptions. With these in turn the temperatures can be determined by applying models that utilize the correspondence between electrical and thermal units. The temperatures are then used to feed reliability models 69)–74). One methodology optimizing transient faults at the system level in the context of other design goals, like performance or power, can be found in Ref. 75). In Refs. 76) and 77) a reliability flow is introduced that considers hot carrier injection and negative bias temperature instability as well as electromigration effects at transistor level. Test chips are stressed to extract the data necessary for an adaptation of well known degradation models. In order to be able to use the models in digital design flows, a technology abstraction up to the gate level is performed. A gate sizing algorithm which takes NBTI-affected performance degradation into account to ensure the reliability of nano-scale circuits for a given period of time was introduced in Ref. 78). It was reported that by an area increase of about 10% one can achieve reliable performance of circuits for 10 years. The impact of various error-control schemes for interconnects on the trade-off between reliability, performance and energy is examined in Ref. 79) using analytical models and SPICE simulations. It could be shown that in low noise environments automatic repeat request schemes should be preferred. In these approaches, the sender encodes flits using an error detection code. The receiver includes a decoder which can detect errors. When an error is discovered, the sender is requested to retransmit the data. This process is repeated until the flit is error free.

2.2.5 Beyond CMOS

Intense research is done with respect to new materials and technologies that allow the replacement of the CMOS based manufacturing process of semiconductor devices.

Currently, the most promising working fields deal with carbon nanotubes (CNTs) 80)–83). CNTs are long macromolecules of pure carbon. For instance, they can be found as nanoscopic threads lying in smear of soot. In principle, in electronic circuits nanotubes can play the same role as silicon, but at a molecular scale, where silicon and other semiconductors cease to work. Furthermore, made in the right way, CNTs conduct current like metals. Because of this, CNTs can also be used as nanowires.
However, before complex nanotube-based circuits can be realized, a huge amount of obstacles and challenges have to be mastered. For example, the controlled creation of CNTs with specific properties is still a difficult problem until today\(^{84}\).

Other approaches show that recent results from synthetic biology can be used in principle to build up application-specific systems that solve certain problem instances in an outstanding parallel manner\(^{85,86}\).

Single electron transistors (SETs) are switching devices that use controlled electron tunneling to amplify current. A SET is made from two tunnel junctions that share a common electrode. A tunnel junction consists of two pieces of metal separated by a very thin (≈1 nm) insulator. The only way for electrons in one of the metal electrodes to travel to the other electrode is to tunnel through the insulator. Since tunneling is a discrete process, the electric charge that flows through the tunnel junction flows in multiples of e, the charge of a single electron\(^{87,88}\).

3. Design Space Exploration for Embedded Multi-Core Systems Considering Performance, Energy and Reliability

3.1 Related Work

In this section, related work dealing with simulation-based analyses of performance, energy, power, temperature and reliability of processor systems is described. In order to clarify the contributions of the proposed methodology, it is compared to the other approaches in Table 1.

A simulation method for the determination of the power consumption of single core processors was introduced in Ref. 89). Here, event signatures describing the complexity of an application, are used as input for microarchitectural power models. The potential of a Dynamic Reliability Management was examined by Srinivasan, et al.\(^{18}\). It could be shown that the incorporation of the target application is necessary for reliability analyses since otherwise a conservative design approach has to be taken to guarantee specified reliability constraints. In doing so, the impact of a Dynamic Voltage and Frequency Scaling (DVFS) was considered. Based on a SystemC implementation, a DVFS mechanism was analyzed in Ref.90) as well. Applications were modeled using a set of abstract parameters and the power consumptions of the examined AMBA AHB Master with Power

<table>
<thead>
<tr>
<th>Table 1</th>
<th>State of the art.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi Core</td>
<td>Multi Tasking</td>
</tr>
<tr>
<td>van Stralen(^{80})</td>
<td>X</td>
</tr>
<tr>
<td>Srinivasan (^{18})</td>
<td>X</td>
</tr>
<tr>
<td>Conti (^{90})</td>
<td>✓</td>
</tr>
<tr>
<td>Bergamaschi (^{91})</td>
<td>✓</td>
</tr>
<tr>
<td>Lebreton (^{92})</td>
<td>✓</td>
</tr>
<tr>
<td>Paci (^{73})</td>
<td>✓</td>
</tr>
<tr>
<td>Coskun (^{70})</td>
<td>✓</td>
</tr>
<tr>
<td>Coskun (^{95})</td>
<td>✓</td>
</tr>
<tr>
<td>Proposed Methodology</td>
<td>✓</td>
</tr>
</tbody>
</table>

1. Design of three-dimensional architectures
2. Design Space Exploration

State Machines. In Ref.91), a simulation model for multi core systems with an integrated power management was introduced. From a given upper power consumption bound of the system as well as from application traces, a strategy for maximizing the system performance by applying voltage and frequency adaptations can be derived. A similar approach was used in Ref.92) for network on chip architectures. Paci, et al.\(^{73}\) developed a method for temperature estimations of low power Multiprocessor Systems-on-Chip (MPSoCs). To determine the activity of memory and computing elements, a simulation platform based on the MPARM-environment\(^{93}\) was used. For temperature calculation, a model was applied that – similar to the HotSpot-Tool of Skadron, et al.\(^{37}\) – employs the well-known thermal-electrical duality\(^{94}\). A simulation methodology to analyze the impact of low cost packages and the placement on the reliability of MPSoCs was introduced by Coskun, et al.\(^{70}\). As workload, synthetic generated sets of independent tasks were used. These tasks were distributed to the processor cores according to a global earliest deadline first strategy. It was shown that the placement has a significant effect on the reliability of MPSoCs. This is especially true when advanced cooling concepts are not applicable for the system under consideration. In Ref.95), the approach was extended. Low power methods like chip
3.2 Proposed Methodology

The proposed methodology for the analysis of embedded multi-core processors at a high level of abstraction is depicted in Fig. 3. It can be used for accurate application-specific temperature and reliability considerations, combining application characteristics only available at system level and abstracted gate level power information. In doing so, the impact of scheduling policies as well as low power methods can be considered. An automated design space exploration can be used for the derivation of an optimized platform.

At first, a platform description format is defined. Using this format, designated applications, the allocation of platform components (in our case processors), the mapping of the applications onto the architecture, the used operating system as well as the applied low power mechanisms are specified.

A platform description is used as input for a subsequent model generation process leading to a system model in SystemC, which combines a functionality representation and component characterizations. An execution of such a system model allows the determination of design parameter values, e.g., for performance, power, energy as well as for temperature and reliability, respectively. An adaptation of platform parameters and a following regeneration of the system model facilitate a design space exploration. When a design point fulfills the designers’ needs, a refinement of the design towards the implementation is carried out.

Power and power distribution models, included into the system model, are the result of a component characterization process, which is described in the next section. The structure of the generated system models and the implementation of the considered low power mechanisms are explained in Section 3.2.2.

3.2.1 Component Characterization

Power Models

To gain power models for the different processor components, data obtained by applying the commercial tool chain depicted on the top right in Fig. 3 was employed. A set of eight small examples consisting of data flow-oriented (ellip, fir, subband, spotdpnm, sintst) and control flow-oriented (gcd, sieve, fibonacci) programs was simulated on the processor gate level netlist, in each case for 2,000 cycles. In doing so, cycle-accurate power values were calculated with the Power Compiler for over 50 components of the synthesized processor. Using this data, instruction-dependent power models of the components were determined by dividing the sum of the power consumptions for instructions of a certain type by the number of times, instructions of this type occur.

The approach for instruction-dependent power models taken here is somehow similar to the one presented by Tiwari, et al. But because the models are not used in the same way, the abstraction levels widely differ. As the work by Tiwari et al. targeted a reduction of energy, a power model for the entire processor based on current measurements was sufficient. In our case, fine grained power models based on gate level simulation results are needed, because the power values obtained by executing the system model are intended to feed corresponding temperature and reliability models. In Fig. 4(a), an excerpt of the instruction-
dependent power model for the component unit_pipe_EX, which mainly generates the control signals for the EX stage of the pipeline, is compiled. The power consumption for Store Multiple (STM) instructions is significantly smaller than for the other types, because STM instructions last usually several clock cycles – without changing much of the control signals.

**Power Distribution Models**

It is essential for temperature and reliability considerations at ESL to uncover the high power densities existing in today’s semiconductors. For this reason, low complexity models applicable at ESL, allowing a more realistic distribution of the absolute power values determined by the execution of system models, have to be developed.

To address the issue whether or not it is possible to identify especially “critical” gates in individual system components and to abstract this information up to ESL, we first analyzed the averaged gate level power consumptions for different components of the designed processor\(^*\). As assumed, it cannot be expected that the components are used by the applications in similar fashion, leading to same power consumptions. As a result, one can come to the conclusion that it is very difficult to derive power distribution models considering single gates, because data-dependencies can hardly be comprised in high level ESL models. But can something be done, if we think more in bigger components instead of gates and are ready to sacrifice some of the information available at gate level?

With this in mind, we checked for the components whether power consumption fractions can be related to area fractions in an application-independent fashion. In doing so, we sorted the gates of the components by their power density in descending order. After that, for each gate the percentage of power and the percentage of area with regard to the component were calculated. Then, for each gate the percentage of power was added to the sum of all power percentages of the gates with a higher power density. The same was done for the area percentages. The determined values were used to plot the percentages of the power consumption against the percentages of the area for the different applications. In Fig. 4(b) this is depicted for the component unit_pipe_EX. It was observed that the curves are similar for the considered applications, even if the power consumptions of the gates and the absolute power values for the component are different. We noticed that same is true for all components of the considered processor.

To achieve manageable models which specify how component power consumptions determined by system model execution are distributed over the component

\[^*\] One might ask the question, why no cycle accurate but averaged power values are considered. The answer is, that this time granularity is not required, because one cycle is not sufficiently long to heat up a corresponding piece of silicon, even if it is assumed that a high power consumption is present and no energy is removed (see rough calculation below).

Specific heat capacity silicon: 700 J/kg·K; density silicon: 2.33 g/cm³
Mass assumed chip (1 cm²·0.01 cm): 2.3·10⁻⁵ kg
Heat capacity: 700 J/kg·K · 2.3·10⁻⁵ kg = 0.016 J/K
1,000 W for 1 ns: ΔT = 1,000 J/s · 10⁻⁹ J/s · 1/0.016 J/K = 62.5·10⁻⁶ K
area (consequently allowing the determination of power densities), we calculated the average curves from the application-dependent curves. After that, the average curves were piecewise linearized, to obtain area fractions for which the power density can be regarded as equal, leading to our power distribution models. In Fig. 4 (c), the procedure is pointed out for the component unit_pipe_EX. An overview of the power distributions for different processor components is given in Fig. 4 (d). It is interesting to notice that the power distributions differ widely in part.

The power consumptions and the power densities of single gates are no longer available. Nevertheless, using our model it is possible to carry out more accurate temperature and reliability considerations at ESL than before. This is because the duration of the time slice for a task is at least in the order of milliseconds when several tasks are alternately mapped to an embedded processor. It can be assumed, that these time intervals are long enough so that single gates virtually reach their steady state temperatures, which are often used for reliability calculations.

3.2.2 System Model

Structure

In Fig. 5, the class diagram of an example system model, which is also discussed in Section 3.3, is depicted. It consists of three main parts: A first part describing the applications, which are executed by the processors, a part modeling the processors themselves as well as a part conducting a scheduling of the applications. In order to keep the effort for model generation as low as possible, parts of the functionality, which are used by all possible system models or which can be easily configured by corresponding constructor arguments, are programmed “by hand” and combined in a static library (see fixed in Fig. 5). Among other things, implementations related to the assumed RTOS are contained. However, application-specific as well as processor-specific parts of the system model are generated (see generated).

On the left side, a piece of the run method of the class sc_rtos_module_turbo_decoder is shown. It is generated for a turbo decoder application. The method is processed during the simulated execution of the application. The superclass method increase_cpu_instruction_count is called in order to forward the number of instructions for the different instruction types to the corresponding processor object for load and power calculations. The superclass owns a pointer to the processor (sc_rtos_context), which is used for the invocation of increase_cpu_instruction_count (see arrows).

Power Consumption Estimation and Low Power

The system model shown in Fig. 5 considers two processor cores. Core 1 has a Past DVFS mechanism available. An excerpt of its implementation is depicted on the upper right. During simulation, an infinite loop is executed. At the beginning, the execution is interrupted for the duration of a power management period – which is defined in the platform description. Then the processor load is calculated for the previous period\(^1\). When the load exceeds the value of 0.7 and the processor is not working in the operation point with the highest performance, frequency and voltage of the processor are increased. When the load is below 0.3

\[\frac{\text{Number of active cycles}}{\text{Total cycle number}}\]

---

\(^1\) Number of active cycles

© 2010 Information Processing Society of Japan
and there is still an operation point with lower performance available, frequency and voltage are reduced.

How the power consumption of the processor components is estimated and hardware measures for reducing the power consumption like operand isolation and clock gating can be considered, is indicated by the code snippet for the component \texttt{unit.pipe\_EX\_DW01\_add\_32\_2}, which is located in the execute stage of the examined processor. The averaged power consumption \( P_i \) of a component \( i \) for a time interval \( t_{\text{power\_interval}} \) is calculated with

\[
P_i = P_{i,\text{dyn},V_{\text{ref}}} \cdot \left( \frac{V_{\text{cur}}}{V_{\text{ref}}} \right)^2 + P_{i,\text{stat},V_{\text{ref}}}
\]

as a sum of dynamic and static power consumption. \( P_{i,\text{dyn},V_{\text{ref}}} \) and \( P_{i,\text{stat},V_{\text{ref}}} \) are the dynamic and static power consumption for a component \( i \) at a voltage \( V_{\text{ref}} \), respectively. \( V_{\text{ref}} \) is the voltage assumed during building the power models for the components (in this case 0.9 V). To estimate the power consumption \( P_i \) for the current supply voltage \( V_{\text{cur}} \), \( P_{i,\text{dyn},V_{\text{ref}}} \) is multiplied by the factor \( \left( \frac{V_{\text{cur}}}{V_{\text{ref}}} \right)^2 \). In doing so, the quadratic dependency of the dynamic power consumption on the supply voltage is taken into account.

While \( P_{i,\text{stat},V_{\text{ref}}} \) was assumed to be fixed\(^1\), \( P_{i,\text{dyn},V_{\text{ref}}} \) is calculated as

\[
\sum_{IT} E_{IT,V_{\text{ref}}} \cdot \#\text{Inst}_{IT} / t_{\text{power\_interval}}
\]

\( IT \) is the set of all instruction types (addition, subtraction, multiplication, \ldots), for which the considered component is active and \( E_{IT,V_{\text{ref}}} \) is the energy needed for the execution of one instruction of type \( IT_i \) from \( IT \) at a voltage \( V_{\text{ref}} \), which can be calculated from the power model. The number of instructions of type \( IT_i \) executed in the interval \( t_{\text{power\_interval}} \) is denoted as \#\text{Inst}_{IT_i}.

When operand isolation or clock gating is available for a component, it is controlled by an enable signal. In order to reduce the dynamic power of the component, it is prevented from switching, when the result is not needed. On the other hand, the static power consumption remains virtually unchanged. In order to express this fact within the power estimation, it is sufficient to adapt the set \( IT \) for the considered component. This can be seen for the component \texttt{unit.pipe\_EX\_DW01\_add\_32\_2} as well. Because of operand isolation, it just works when instructions of the types \texttt{MUL} (Multiply) and \texttt{MLA} (Multiply-Accumulate) are executed.

### 3.3 Results

In order to demonstrate the applicability of the proposed methodology, it was used for the analysis of an example scenario, which is depicted in Fig. 6(a). Security-critical image and audio data is to be transmitted over an insecure channel. For this purpose, the data is compressed by the sender at first (JPEG- and ADPCM-Encoder). After that, a checksum is calculated (CRC32) and an encryption is performed (Blowfish). Finally, a turbo decoder algorithm is applied. At the receiver, the data is decoded.

As design problem, the implementation of the receiver by a dual core processor was considered. For each of the two ARM-like processors (see Ref. 99)), four

\(^1\) Average value of the static power consumptions observed during derivation of the power models with ModelSim and Power Compiler, see Ref. 99.)
possible operation points\textsuperscript{1} were assumed. The design space, consisting of several billion platforms, is defined by the parameters that can be altered. These are compiled in Fig. 6 (b).

For an exploration of the design space, 10,000 different platforms were examined and each one of them was simulated for two seconds. The results with respect to performance, energy and a self-defined reliability measure\textsuperscript{2} are depicted in Fig. 7 (a). The position of the Pareto points is highlighted using plotted lines. The results show that a performance increase generally goes along with higher energies and a reduction of reliability. In this regard, the results match the expectations.

For the selection of a design point meeting the requirements, especially the

\begin{equation}
R_{Pl_i} = 1 - \max \left( \frac{\Theta_{Temp_{Pl_i}}}{\Theta_{Temp_{Pl_a}}} , \frac{\Theta_{Temp.Cycles_{Pl_i}}}{\Theta_{Temp.Cycles_{Pl_a}}} \right)
\end{equation}

\text{Eval. average temperature of } Pl_i \quad \text{Eval. temperature cycles of } Pl_i

The negative impact of high average temperatures as well as of temperature cycles on the reliability shall be considered together.

numbered Pareto points would come into consideration. This is because the energy and the reliability of the non-numbered Pareto points are almost equal to the values of point 1 but the performance values are significantly lower. For Pareto points 1 to 10, performance, energy and reliability are plotted in Fig. 7 (b) with respect to point 1. It becomes apparent that for an optimal relation between performance and energy, the platform belonging to Pareto point 6 should be chosen. Furthermore, the reliability of this point takes an average value compared to the other Pareto points.

4. Learning Classifier System XCS for SoC Run-Time Control

In this section we evaluate whether XCS can control the operating point of a SoC at run-time under changing conditions\textsuperscript{100}. For this evaluation, we define the operating point as the operating frequency and voltage of the SoC; other setups may include more complex parameters such as bus width or which processor is running.

4.1 Models

We identify the optimal operating point of a SoC as the combination of performance, temperature, power consumption and (soft) error rate, which are all run-time values. In the following we describe the models from the literature that we use to estimate these values.

We use the frequency as a (rough) estimate of the performance of the system. Later, this can be replaced by more sophisticated measures. Until then we note that, as the evaluator is model-free, it does not directly depend on the sophistication of the performance measure. For temperature simulation, we use the HotSpot tool presented in Ref. 101, based on our following power estimates. In Ref. 102, the authors have shown the high temperature simulation accuracy of the HotSpot tool.

Total power consumption $P_{total}$ can be divided into static and dynamic power consumption:

\begin{equation}
P_{total} = P_s + P_d
\end{equation}

We estimate the static power consumption $P_d$ with the model of Butts, et al.\textsuperscript{103}:
\[ P_s = V_{DD} N_{\text{design}} \hat{I}_{\text{leak}} \]  \hspace{1cm} (2)

with \( V_{DD} \) as the supply voltage, \( N \) transistors, and design and technology dependent parameters \( k_{\text{design}} \) and \( \hat{I}_{\text{leak}} \) given by\(^{103} \).

We estimate the dynamic power consumption \( P_d \) with the model of Weste, et al.\(^{104} \), to which we add an activity factor \( \alpha \) as done by Intel to estimate power consumption in the Pentium M\(^{105} \):

\[ P_d = \alpha C_L V_{DD}^2 f_p \]  \hspace{1cm} (3)

with \( C_L \) being the lump capacitor, and \( f_p \) the clock frequency. The activity factor \( \alpha \) is the probability that a single transistor switches from zero to one (and thus consumes power). It can be gained through logic simulation.

An accurate model for timing errors is difficult to describe, as the timing error depends on the actual path a signal takes through the circuit. We therefore assume a fixed set of inverters that represent either the longest or the average path length. We get the average temperature-dependent switching time of an inverter from Ref. 106) as:

\[ t_{\text{av}} = \left( \frac{t_{\text{dr}} + t_{\text{t delay}}(T)}{2} \right) + \left( \frac{t_{\text{df}} + t_{\text{t delay}}(T)}{2} \right) \]  \hspace{1cm} (4)

\( t_{\text{dr}} \) and \( t_{\text{df}} \) are the voltage-dependent rise and fall delays of a signal on an inverter\(^{104} \). \( t_{\text{t delay}} \) represents the temperature dependency of the time delay as shown in Ref. 106). The model is sufficient for our simulation to determine temperature-dependent timing errors. More detailed models would improve the simulation accuracy concerning the learned classifier table of the XCS, but do not affect the actual learning capabilities of the evaluator.

Our model of the effect of frequency and voltage scaling on fault rates is the one of Zhu, et al.\(^{107} \). The model is based on a Poisson distribution with parameter

\[ \lambda(f) = \lambda_0 10^{\frac{(f-f_{\text{min}})}{d}} \]  \hspace{1cm} (5)

with \( \lambda_0 \) as the average fault rate corresponding to \( V_{\text{max}} \) and \( f_{\text{max}} \) and \( d \) as a technology-dependent constant. Voltage and frequency are scaled to \([0, 1]\) with \( V_{\text{max}} = f_{\text{max}} = 1 \) and assumed to be approximately linearly dependent.

We do not model hard errors because usually their mean time to failure is in the scale of years and thus beyond the scope of our evaluation.

### 4.2 Experimental Setup

In the following we describe the hardware and software we are using for the evaluation of the XCS, how we encode the environment and action of the XCS, and which tests we use for evaluation.

We use the AMD Opteron (Barcelona) as the basis for a multi-processor SoC (MPSoC). The AMD Opteron has four general purpose processor cores on a single die. We choose the AMD Opteron because most of the parameters necessary for simulation are publicly available and because it can adjust the frequency of each core individually. The floor plan of the AMD Opteron is depicted in Fig. 8, as derived from\(^{108} \). We choose the activity factor for the cores so that we meet the average CPU power and the thermal design power. For each core, there is one XCS controlling it. The XCS runs on dedicated hardware to not interrupt regular core operation.

Concerning the software, we use the following four algorithms to run on each core: LR-decomposition, video filtering, matrix multiplication, and a dual-process application where one process has to wait for the other. The algorithms are simulated by traces that contain the memory access patterns, the computation time needed for calculation and the activity factor. This information suffices for a realistic simulation with good estimates concerning the self-adaptation behavior of a SoC as controlled by an XCS. The cores do not actually calculate anything to keep simulation time manageable. When all cores are busy, total
power consumption lies at 83 W and temperature at 55°C. When all cores are idle, total power consumption lies at 26 W. These simulated values are comparable to the actual values of the Opteron108.

We distinguish eleven different frequencies (500 MHz–3,000 MHz, encoded in four bits) and five voltage levels (0.8 V–1.3 V, encoded in three bits). The temperature range is 50°C–90°C, and encoded in five bits. The input signal of the evaluator contains information on frequency, voltage, current temperature, and whether a timing error occurred. The output signal of the evaluator (the action) consists of setting the frequency and the voltage. For this evaluation, we treated frequency and voltage separately, although a more realistic simulation would use a fixed set of frequency-voltage pairs that also contains operating points with higher voltage as actually needed to compensate future aging effects.

The XCS knows two different modes of operation: multi-step and single-step. As in the multi-step problem the optimal operating point has to be known in advance (to signal that the problem has been solved and the reward can be distributed to the classifiers that made the solution possible48), we use the single-step mode of operation for the XCS.

4.3 Learning Setup

For the initial learning, we use the fact that the four cores are identical and reduce total simulation time by simulating only a single core while the other cores idle at 2,000 MHz at 1.2 V. The single core has an activity factor (see Section 4.1) of 0.05. We do not simulate cache access during initial learning, but in the later scenarios. We ensure that all possible operating points are tested sufficiently often by doing 50,000 repetitions. Between different runs, we randomly raise the temperature by 5 K, 10 K, 20 K, or 30 K from the default temperature.

After the XCS has been trained, we evaluate the following three scenarios:

• **simple control:**
  In this scenario, the XCS is supposed to find the optimal operating point under the trained conditions. We expect the XCS to actually find the optimal operating point.

• **control under changed environment:**
  In this scenario, the XCS is supposed to find the optimal operating point although the actual environment differs from the environment during training. This simulates the situation where the design-time model of the SoC differs from the actual run-time situation of the SoC. We expect the XCS to self-adapt to the changed environment.

• **online learning without genetic algorithm:** In this scenario, the XCS is supposed to learn a new utility function without using its genetic algorithm. This mimics a possible hardware implementation of the XCS which lacks the genetic algorithm due to area limitations. We expect the XCS to be able to learn the new utility function (by adjusting the already learned classifiers), albeit not as well as when the genetic algorithm is available. In this scenario, we set the learning parameter \( \beta = 1.0 \) to reduce the time needed for learning.

4.4 Utility Functions

Different optimization goals require different utility functions. In the scenario of simple-control, the XCS is supposed to maximize performance with minimal power consumption and low error rate. We weighted performance and power consumption equally and penalize timing errors:

\[
R(f, p, t, v) = w_1 \frac{f}{f_{\text{max}}} + w_2 \left(1 - \frac{p}{p_{\text{max}}} \right) + w_3 \text{rel}(t, v, f)
\]

with \( \text{rel}(t, v, f) \) modeling the reliability with respect to timing errors as

\[
\text{rel}(t, v, f) = \begin{cases} 
0 & \text{if timing error} \\
1 & \text{otherwise}
\end{cases}
\]

\( f_{\text{max}} \) and \( p_{\text{max}} \) are the maximum frequency and maximum power consumption, respectively. The maximum power consumption needs not to be given exactly; a rough estimate that is larger than the actual value suffices. Concerning the weights, we choose \( w_1 = 200, w_2 = 35, \) and \( w_3 = 200 \) in our simulations.

Figure 9 shows the timing errors depending on temperature and voltage at 2,000 MHz. From the figure we note that the operating point [2,000 MHz, 1.2 V] is safe in terms of faults due to timing error (at the usual temperature range up to 70°C).
In the scenario with changed environment, we raised temperature by 15 K. Also, we changed the \( \text{rel}(t, v, f) \) function in (6) to

\[
\text{rel}(t, v, f) = \begin{cases} 
0 & \text{if timing error} \\
\left(\frac{20}{70}\right)^2 & \text{if } t > 70 \\
1 & \text{otherwise} 
\end{cases} 
\]  

(8)

We further changed the weights to \( w_1 = 200, w_2 = 35, \) and \( w_3 = 200 \) if temperature was below \( 70^\circ \text{C} \), and to \( w_1 = 100, w_2 = 100, \) and \( w_3 = 200 \) if temperature was above \( 70^\circ \text{C} \). This represents an emergency behavior which allows the XCS to use a less performing setting and shows another way how the designer’s prior knowledge may enter the XCS control mechanism.

In the scenario with the disabled genetic algorithm, we change the goal of the reward function to also minimize the waiting time between two processes:

\[
R(f, p, t, v, w) = w_1 \text{time}(w) \\
+ w_2 \left(1 - \frac{p}{p_{\text{max}}}\right) \\
+ w_3 \text{rel}(t, v, f)
\]  

(9)

Here, \( \text{rel}(t, v, f) \) is the same as in (8) and

\[
\text{time}(w) = \begin{cases} 
1 - \frac{w}{w_{\text{max}}} & \text{if wt(\text{Core 1})=0} \\
0 & \text{otherwise} 
\end{cases}
\]

where \( \text{wt(\text{Core 1})} \) is the waiting time of Core 1 and \( w_{\text{max}} \) is the maximum possible waiting time. Again, a rough estimate for the maximum possible waiting time suffices.

4.5 Results

After the initial learning, we use the XCS to control an MPSoC that runs an algorithm on each core. Each XCS contains about 600 classifiers, which require about 8 kB of memory to be stored on the chip (104 bit per classifiers including 84 bit solely for classifiers parameters such as fitness, etc.), if we implemented the XCS unmodified on the SoC. The XCS implementation of Ref. 67 promises to need less memory, however we have not analyzed this further. We inhibit exploration steps of the XCS as well as the genetic algorithm to avoid creating new classifiers and to simulate the situation of the XCS on a SoC. In the scenario of simple-control, we change to a random frequency and voltage every 10 s of simulation time. Figure 10 shows the resulting frequency and voltage settings. We observe that the XCS resets the frequency and voltage to 2,000 MHz and 1.2 V which leads to no errors in the actual temperature range (see Fig. 9) and is the optimal setting.
16 Design and Run-time Reliability at the Electronic System Level

Figure 11 shows the result for the scenario of a changed-environment. Once the temperature raises above 70°C, XCS changes the frequency and voltage such that temperature falls again and timing errors stay low. However, we also observe an oscillating behavior, as the XCS “forgets” that the previously chosen setting makes the system temperature raise above limits. This will be a point of future research.

For the scenario of online learning, Fig. 12 shows the frequency-voltage pairs the XCS tries out to learn the new reward function, which aims to minimize the waiting time \( w \) of Core 2 for Core 1 (and thus keeps total run time low). We observe that, despite the high learning rate, the XCS needs a long time to learn the new reward function. However, and most importantly, we also observe the XCS is able to self-adapt to the new reward function.

The results show that XCS can control the operating point of a SoC, even under changed environmental conditions. We also showed that the XCS can learn new reward functions without a functioning genetic algorithm, as it will be the case if the XCS is implemented on a SoC. Future work includes evaluating real applications instead of algorithms, reducing the necessary memory footprint, and how further actions (such as changing bus width or turning off processors) affect the performance of the XCS.

5. Conclusions

In this paper, phenomena threatening the reliability of semiconductor systems as well as the current status of related research work and tools were surveyed. It was highlighted that the gap between the target applications and the correct modeling of physical effects has to be bridged at the electronic system level. As a first step, a design time and a run-time method targeting multi-core systems were introduced. While the design time method can be used to explore the trade-off between performance, energy and reliability in an early design phase, an autonomic learning technique based on XCS was applied to extend the reliability of a multi-core system during run-time.

Acknowledgments This work is partially funded by Deutsche Forschungsgemeinschaft, Priority Program Organic Computing (SPP 1183) under the grant RO 1030/14-2 and the BMBF project ROBUST under grant 01M3087.

References

2) Borkar, S.: Designing reliable systems from unreliable components: The chal-
http://www.edn.com/article/CA6528579.html

(Received January 28, 2010)
Björn Sander is research scientist at the FZI Research Center for Information Technology, Karlsruhe, Germany. He received a diploma in Computer Science from the University of Karlsruhe in 2006 and is currently working towards his Ph.D. at the Computer Engineering department at the University of Tübingen, Germany. His research interests include electronic system level methodologies and low power design.

Andreas Bernauer was born in 1978. He received his M.S. from University of Connecticut, USA, in 2004 and graduated in Computer Science at the University of Tübingen, Germany, in 2005. Since 2005, he has been working at the Computer Engineering department at the University of Tübingen. He is currently pursuing his Ph.D. and is a member of the Autonomic System-on-Chip project. His research interests include organic computing and learning classifier systems.

Professor Wolfgang Rosenstiel received his Ph.D. in 1984 from the Karlsruhe University. Since 1990 he is a professor (Chair for Computer Engineering) at the Wilhelm-Schickard-Institute for Informatics (WSI), Tübingen University, as well as the director of FZI department “System Design in Microelectronics”. He is a member of ITRS-Committee, the chairman of German edacen-trum, the editor-in-chief of journal “Design Automation for Embedded Systems”, a committee member of DFG senate for Collaborative Research Centers. Professor Rosenstiel received Shared University Research Grant 2007 from IBM and the DATE fellow award for the long term commitment in EDA and the support for the DATE conference. His research topics are hardware/software design and design tools for embedded systems, computer architecture, parallel computing and neural nets.