Current state of ASoC design methodology

Andreas Bernauer[1], Dirk Fritz[1], Björn Sander[2], Oliver Bringmann[2], and Wolfgang Rosenstiel[1,2]

[1] Wilhelm-Schickard-Institute of Computer Science,
Department of Computer Engineering
72076 Tübingen, Sand 13, Germany
bernauer@informatik.uni-tuebingen.de
[2] Forschungszentrum Informatik
76131 Karlsruhe, Haid-und-Neu-Str. 10-14, Germany
bsander@fzi.de

Abstract. This paper gives an overview of the current state of ASoC design methodology and presents preliminary results on evaluating the learning classifier system XCS for the control of a QuadCore. The ASoC design methodology can determine system reliability based on activity, power and temperature analysis, together with reliability block diagrams. The evaluation of the XCS shows that in the evaluated setup, XCS can find optimal operating points, even in changed environments or with changed reward functions. This even works, though limited, without the genetic algorithm the XCS uses internally. The results motivate us to continue the evaluation for more complex setups.

Keywords. Dagstuhl Seminar Proceedings, System-on-Chip, design methodology, system reliability, learning classifier system, XCS, ASoC

1 Introduction

The number of System-on-Chip (SoC) designs is expected to increase strongly according to the International Technology Roadmap for Semiconductors [1]. Lower power consumption, higher performance and simpler system integration are the major advantages of SoC design 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. In particular, changed electrical circuit properties, the susceptibility to internal and external noises and accelerated aging pose great challenges [2,3]. The accelerated aging is a consequence of higher average on-chip temperatures which result from higher power densities as well as of thermal cycles. Various failure mechanisms, like electromigration or time-dependent dielectric breakdown, occur earlier in the lifetime of a chip. Overall, this leads to a decrease in reliability. In the past, the reliability

Dagstuhl Seminar Proceedings 08141
Organic Computing - Controlled Self-organization
http://drops.dagstuhl.de/opus/volltexte/2008/1564

of semiconductor devices was mainly seen as a manufacturing problem. First approaches that tried to improve reliability by blind redundancy turned out to be too expensive—at least for most market segments. This and the fact that it is hardly possible to unfold the opportunities for increased reliability offered by the functionality at the chip level, the functionality of the system comes into focus with respect to reliability considerations and special attention is payed to higher design levels.

The decreased feature sizes also lead to an increased design complexity, as more and more transistors fit on an individual chip. This makes it difficult for a designer to foresee all possible operating conditions and failure modes of a chip. To alleviate the designer's life, we foresee some intelligence on an Autonomic System-on-Chip (ASoC), which takes at run time the decisions the designer formerly took at design time. In this paper, we present our first results on evaluating how the learning classifier system XCS can be a candidate for this intelligence and control the operating point of an ASoC.

This paper is organized as follows: Section 2 summarizes related work. In Section 3, the developed methodology for system reliability estimation at design time is presented in detail and experimental results are discussed. Section 4 shows the evaluation of XCS to control a SoC at run time. Finally, Section 5 concludes the paper.

2 Related work

Great efforts were made to be able to describe the effects of the aging semiconductor devices through mathematical models. An overview is given in [4].

There exists a lot of work on structural reliability analysis based on Reliability Block Diagrams (RBDs) and fault trees [5]. An RBD is a form of reliability analysis using a functional diagram to portray and analyze the reliability relationship of components in a system. In doing so, each component is represented by a block, which is generally interconnected with other blocks [6]. A system that is described by a RBD fulfills its requirements if and only if there is a path from the entrance of the RBD to the exit which exclusively comprises functioning blocks.

In order to estimate the temperatures of semiconductor devices during operation, the power consumptions of the system components have to be known.For this purpose, Power State Machines (PSMs) are frequently used as a model. PSMs are graphs with a finite set of states which represent potential component configurations. The performance and power consumption of the corresponding configuration is associated with each state. An approach based on the modeling of system components with PSMs was introduced in [2].

The effects of Dynamic Reliability Management (DRM) on the reliability of single microprocessors were investigated by Srinivasan et al. [7]. They showed that it is necessary to take the target application into consideration during reliability analysis, since otherwise the worst case behavior (of all possible applications) has to be chosen to assure that reliability guidelines are met. This can

lead to high costs, for example for chip cooling solutions. Srinivasan et al.used an instruction set simulator to determine the activity times of the microarchitectural units. Neither structural redundancies nor power management methods were taken into consideration.

The leakage contribution to the total power dissipation increases with each new technology node. An approach which enables an leakage-aware design space exploration at system level was presented in [8].

In this work, the impact of the power management, the placement and present structural redundancies on the reliability of MPSoCs is examined during an early design phase.

Our approach on estimating system reliability takes a middle position to the methods discussed above, as we determine the activity of the system components by executing a SystemC model, which is generated from an abstract representation of the system functionality. Methods based on instruction set simulators allow determining the activity of the system parts on a fine-grained level, but the simulation speed is potentially insufficient. In contrast, techniques that rely on a synthetic load ensure a fast simulation speed, but the effects of the target application cannot be taken into account.

Holland et al. [9] first proposed the learning classifier system (LCS). A learning classifier system consists of a set of condition-action pairs (the classifiers) which are learned and executed in an environment. The LCS is supposed to learn the necessary classifiers for a specific environment to reach some preset goal. Wilson et al. [10] presented a specialized version of an LCS, the XCS, for which studies showed [11] that it learns accurate, complete, and minimal representations for boolean functions. Butz [12] wrote an implementation of XCS in C, which we used as a reference implementation.

The only known application of learning classifier systems to control a machine is AutonoMouse [13], where an robot mouse learned to approach a light source under different noise and lesion conditions. To the best of our knowledge, XCS has not yet been applied to control the state of a SoC.

3 Reliability estimation at design time

The reliability of a design is calculable with two major pieces of information: how the individual components of the system depend on each other and how reliable the individual components are. For the first major piece of information, the dependence of the individual components, we use the standard approach of a reliability block diagram (RBD) and fault tree generation and analysis [5]. In general, we assume that the reliability block diagram is given for a specific design.

The reliability of the individual components of a design—the second major piece of information to calculate overall system reliability—is more difficult to obtain, as it depends on various run-time parameters, most notably temperature. Section 3.1 shows the failure models we use. Section 3.2 presents our system reliability estimation. Section 3.3 shows experimental results.

3.1 Component reliability

Our analysis currently covers temperature as a source of influence on the reliability of semiconductor components. Most fault models describing the aging of semiconductor components due to temperature are based on the Arrhenius relationship, whose general form is [4]:

$$\lambda(T) = c \cdot e^{-\frac{E_a}{kT}} \tag{1}$$

$\lambda(T)$ is the failure rate (in failures in time) with respect to the absolute temperature T, E_a is a material dependent activation energy for the corresponding failure mechanism (here: 1 eV which corresponds to the activation energy of aluminum for electromigration [4]), k is Boltzmann's constant ($8.62 \cdot 10^{-5}\,\mathrm{eV/K}$), and c a material specific constant.

To compare different design alternatives, we compute the acceleration factor $A(T_i)$ at temperature T_i, which is the ratio of the failure rate at temperature T_i and some base temperature (here we use 333.3 K (60 °C)):

$$A(T_i) = \frac{\lambda(T_i)}{\lambda(333.3K)} = e^{\frac{E_a}{k}\left(\frac{1}{333.3K} - \frac{1}{T_i}\right)} \tag{2}$$

If the failure rate is regarded as fixed, the reciprocal value of λ corresponds to the mean time to failure, MTTF. It would be desirable to determine the absolute reliability of a system in form of a MTTF value from the operating temperatures. For this, all material-dependent constants had to be known, which is seldom the case. Assuming $\lambda(333.3K) = 1$ allows us to compare designs anyway, although the actual failure rates are unknown.

Another fault model which describes semi-conductor aging due to temperature variation is the Coffin-Manson equation [4]:

$$N_f = C_0(\Delta T)^{-q} \tag{3}$$

N_f is the average number of temperature cycles of range ΔT a device can cope with until an error occurs. C_0 and the Coffin-Manson exponent q are material-dependent constants. From equation (3) follows that a single temperature cycle of range ΔT consumes

$$\frac{1}{N_f} \cdot 100 = \frac{\Delta T^q}{C_0} \cdot 100 \tag{4}$$

percent of the lifetime of the circuit. This means that ΔT^q is proportional to the lifetime consumed by a single cycle of range ΔT. Thus, by determining the amount and size of the temperature cycles a component goes through, we can calculate a value which is proportional to the consumed lifetime.

3.2 System reliability

The temperature of a system depends (besides the temperature of the environment) on the power consumption of the component, which in turn depends on its

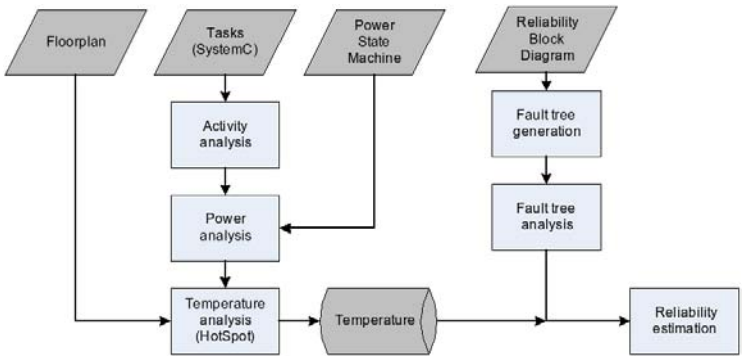

Fig. 1. System reliability estimation at design time.

activity, which in turn depends on its behavior at run time. Thus, the starting point of our reliability analysis is an executable system specification, written in SystemC [14].

Figure 1 gives an overview of our current system reliability analysis. In our analysis, the components of a system are processes which perform a certain task. We make the following simplifications to keep simulation time and complexity bearable. First, we base our analysis on a static analysis of the processes, that is we ignore the impact of input data. Second, we assume that each process runs on its own processor, that is, the system has no task scheduler. With this, the run-time behavior of a process depends only on its control flow and its communication with other processes. Determining the execution time and the activity behavior of a process still remains a hard problem, because the communication of the process with other processes usually leads to cyclic dependencies.

The first step to obtain a processor's temperature behavior is a static timing and activity analysis of the process running on a selected processor. This results in minimal and maximal execution times for each basic block of the process. Based on these execution times, we use SysXplorer [15] to generate a communication dependency graph (CDG) of the whole system. A vertex in the CDG corresponds to a communication point within a SystemC process at which the SystemC process either receives or sends data to other processes. An edge either represents the communication between two SystemC processes or, if it is within one SystemC process, all possible control flows between the two connected communication points. In the latter case, the edge is annotated with the minimal and maximal execution time of the shortest and longest control flow path between the connected communication points, respectively, which result from applying the algorithm presented in [16].

Based on the activity analysis, the SysXplorer tool generates SystemC simulation code which represents the activity of the analyzed system. This simulation behaves like the analyzed system in terms of average waiting and computation times, without computing anything actually. The simulation records the time periods the process is waiting or computing, showing when the process is active.

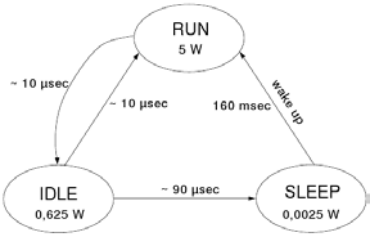

Fig. 2. Power state machine adopted for the PowerPC 750.

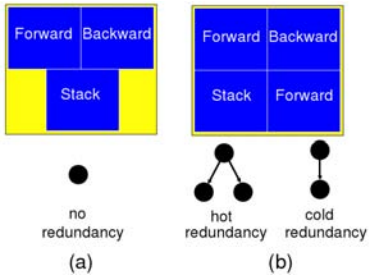

Fig. 3. Assumed placements and fault trees

Using the power state machine of the processor, we can determine when the processor is taking which power saving state. This gives us an estimate of the processor's power consumption over time [17].

Given the power consumption over time and the floor plan of the final design, which indicates the relative position of the processors, we can calculate the temperature behavior of the system using the Hotspot tool [18]. Together with the fault tree analysis mentioned at the beginning of this section and the reliability equations presented in 3.1 this results in the overall system reliability.

3.3 Results of system reliability calculation

We applied the presented system reliability analysis to two multi-processor system-on-chips (MPSoC) that both realize a 1/3,(14,12,3)-Viterbi decoder but that differ in their power management: one uses a full three-state power machine while the other uses a two-state power state machine without a SLEEP state.

The *forward*, *backward* and *stack* processes of the Viterbi algorithm each run on a dedicated PowerPC 750 processor. As the power management strategy of the PowerPC 750 was unknown, we adopted the strategy of the StrongARM SA-1100 [19] to the power values of the PowerPC 750 (Figure 2). Figure 3 shows the assumed placement for both MPSoCs, with the size of the PowerPC taken from [20].

Figure 4 shows the temperature behavior of both MPSoCs for the time interval from 0.85 s to 1.5 s, after the temperature reached a stable state. The temperature behavior for the MPSoC with three-state power management (labeled "with power management") shows some temperature variations, especially for the *forward* processor. The temperature behavior for the MPSoC with two-state power management (labeled "without power management") does not exhibit these large variations. The large temperature variations result from the large transition time from the power saving state SLEEP to RUN: while the MPSoC with three-state power management needs 160 msec to wake up, the MPSoC with two-state power management can quickly change between RUN and IDLE mode in about 10 μsec.

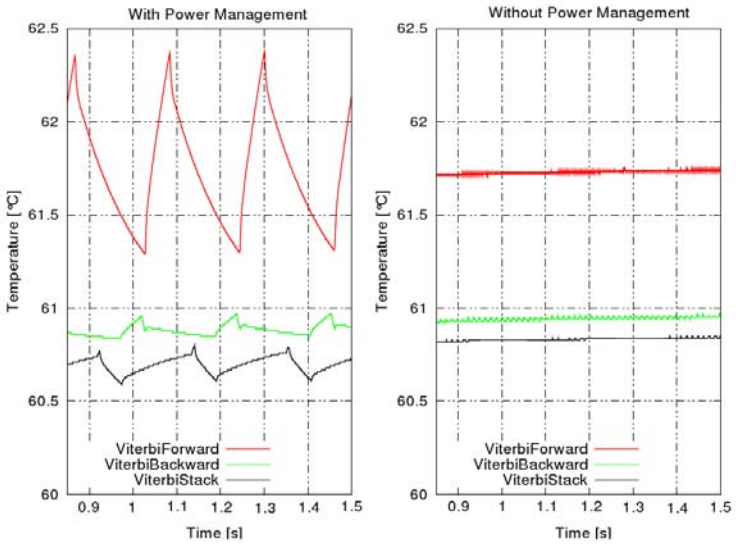

Fig. 4. Viterbi decoder temperature curves.

Table 1 shows the mean time to failure (MTTF) and consumed lifetime values for both MPSoCs, standardized to the case with two-state power management. From the table we conclude, that the power management does not lead to noteworthy reliability gains, but the time to first failure due to temperature cycling is decreased by a factor of about 40. Changing the placement of the processors did not affect the reliability values (data not shown).

We applied our system reliability analysis also on two version of the Viterbi decoder with hot and cold redundancy, shown in Figure 3. Table 2 shows the resulting mean times to failure. We observe that the reliability increases, more for cold redundancy than for hot redundancy as expected (in cold redundancy the redundant component only ages when the other component fails).

4 Evaluation of XCS

The goal of this evaluation is to determine whether XCS can control the operating point of a SoC at run-time. For this evaluation, we define the operating point as the operating frequency and voltage of the SoC; later setups may include more complex parameters such as bus width or which processor is running.

The optimal operating point of a SoC is mainly influenced by the run-time properties performance, temperature, power consumption, and (soft) error rate. Section 4.1 describes the models from the literature we use to estimate these properties.

	PSM	
	3-state	2-state
Temperature *forward* [°C]	61.56	61.61
Temperature *backward* [°C]	60.79	60.87
Temperature *stack* [°C]	60.57	60.77
Mean Time To Failure	1.01	1.00
Consumed Lifetime	38.42	1.00

Table 1. Average temperatures, MTTF and CLT values

Variation	MTTF
no redundancy	1.00
hot redundancy	1.22
cold redundancy	1.36

Table 2. Results of the MTTF analysis

4.1 Parameter models

This section describes the models we use to estimate the power consumption and the soft error rate of the system. For the performance, we use the frequency as a (rough) estimate. Later setups will include more sophisticated performance measures. For the temperature, we feed the Hotspot tool [18] with the estimated power values. See Section 3.2 for details on the Hotspot tool.

Power consumption The power consumption consists of the static and dynamic power dissipation.

$$P_{\text{total}} = P_s + P_d \tag{5}$$

For the static power dissipation, we use the model of Butts et al. [21]

$$P_s = V_{\text{DD}} \times N \times k_{\text{design}} \times \hat{I}_{\text{leak}} \tag{6}$$

where V_{DD} is the supply voltage, N is the number of transistors, k_{design} is a design dependent parameter, and $\hat{I}_{\text{leak}}$ is a technology dependent parameter. k_{design} and I_{leak} are given in [21] (we use $k_{\text{design}} = 9$ for the cores, $k_{\text{design}} = 1.2$ for the caches, and $\hat{I}_{\text{leak}} = 10^{-8}$.

For the dynamic power dissipation, we use a modified version of the well-known model [22] as used by Intel to estimate power dissipation in the Pentium M [23]:

$$P_d = \alpha C_L V_{\text{DD}}^2 f_p \tag{7}$$

where α ($0 < \alpha < 1$) is the activity factor, C_L is the lump capacitance, and f_p is the clock frequency. The activity factor gives an estimate on the average number of zero-to-one transitions during a clock cycle and can be gained through logic simulation.

Timing errors Defining an accurate model for timing errors is difficult, as the timing error depends on the actual path the signal is taking between the pipeline stages. For this, we use a simple model where we assume a fixed set of inverters between the two pipeline stages. The amount of inverters could be either the longest or the average path between the pipeline stages. We model the average switching time for an inverter as

$$t_{\mathrm{av}} = \frac{(t_{\mathrm{dr}} + t_{\mathrm{tempdelay}}(T)) + (t_{\mathrm{df}} + t_{\mathrm{tempdelay}}(T))}{2} \tag{8}$$

where t_{dr} and t_{df} are the well-known raise and fall delays of a signal on an inverter [22] and $t_{\mathrm{tempdelay}}$ models the influence of the temperature on the time delay [24] as:

$$t_{\mathrm{tempdelay}}(T) = \frac{T - 218\,\mathrm{K}}{418\,\mathrm{K} - 218\,\mathrm{K}} \cdot t_g \tag{9}$$

where t_g is the whole additional time delay due to raising the temperature from 218 K to 418 K.

Soft errors We use the model of Zhu et al. [25] to model the effect of frequency and voltage scaling on fault rates:

$$\lambda(V, F) = \lambda(f) = \lambda_0 10^{\frac{(1-f)d}{1-f_{\mathrm{min}}}} \tag{10}$$

where λ_0 is the average fault rate corresponding to V_{max} and f_{max} and d is a constant. Zhu normalizes the range of minimum and maximum voltage V and frequency, that is $V_{\mathrm{max}} = f_{\mathrm{max}} = 1.0$ (therefore, and because delay scales linearly with $\frac{1}{V}$ [26], here $V = f$). The fault rate is used as a parameter in a Poisson distribution describing the occurrences of faults.

As the mean time to failure due to hard errors is usually in the scale of several years, we do not model the effect of hard errors.

4.2 Experimental setup

We used the AMD Opteron (Barcelona) Processor as the hardware, which is a four-core general purpose processor with a three level cache hierarchy produced on a single die in 65 nm technology. The advantage of using the Opteron as an MPSoC was that most of the parameters which are necessary for simulation are publicly available, and that the processor can adjust the frequency of each core individually. Figure 5 shows the floor plan of the Opteron, as derived from [27]. Every core in the Opteron has its own L1-cache (not depicted) and L2-cache. All cores share a common L3-cache. The caches are 2-, 16-, and 32-way associative, respectively, that is each cache block is 32 KB in size and each cache line contains 64 bytes [27]. We adjust the activity factor of the cores so that we meet the thermal design power and average CPU power.

We modeled the Opteron with `libasoc`, a simulation library written with SystemC to model ASoC designs. We executed four algorithms: LR-decomposition,

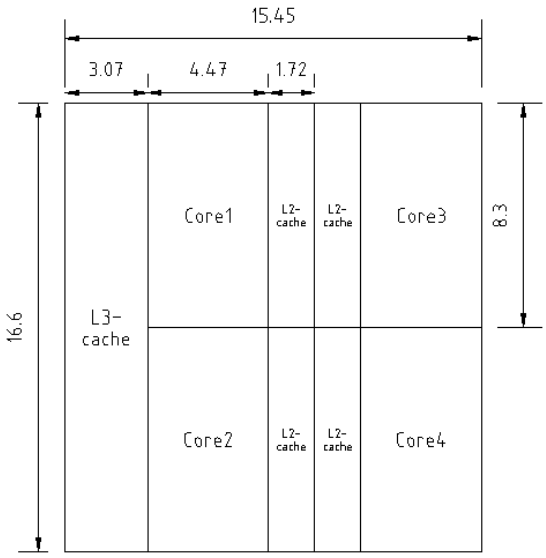

Fig. 5. Floorplan of the Opteron. Measurements are given in mm.

video filtering, matrix multiplication and a dual-core application. Figure 6 shows the traces of the algorithms. `LR[B]` *`mult rows act`* describes the LR computation, where *`mult`* is the number of needed multiplications or divisions, *`rows`* is the number of rows of the equality system, and *`act`* is the activity factor. `[LOAD|STORE] CACHE` *`mem act lines`* loads or stores *`lines`* lines of data to/from *`mem`*. `GR` *`size pixels act`* describes the *`size`*x*`size`* filter of a video containing *`pixels`* pixels per image. `ACT` *`cycles act`* runs a core for *`cycles`* cycles. `READY` and `WAIT_ON` synchronize two cores.

We simulated the trace files for one second of simulated time, where the cores repeatedly executed the algorithms. A single core could solve 7 417 linear equality systems (LR-decomposition), filter 231 images per second, or calculate 101 698 matrix multiplications. When Core 1 and Core 2 do LR-decomposition, Core 3 does matrix multiplication and Core 4 does video filtering, only 7 311 LR-decompositions, 223 images and 91 917 matrix multiplications could be computed, indicating that access to main memory becomes the bottleneck. In the latter case, power consumption lies at 83 W and temperature raises to 55 °C. When all cores are idling, power consumption lies at 26 W. These simulated values are comparable to the actual values of the Opteron [27].

For the environment of the XCS, the following parameters were available (compare to Section 4.1): temperature, power dissipation, timing error (only logic), frequency, voltage, and amount of soft-errors (only memory). We allowed eleven different frequencies ranging from 500 MHz to 3000 MHz and five voltage levels ranging from 0.8 V to 1.3 V. We encoded frequency with four bits, voltage

```
LR:
  LOAD CACHE MEM 0.02 1
  LR 3 63 0.06
  LRB 3 63 0.06
  STORE CACHE MEM 0.02 8
```

```
VIDEO:
  LOAD CACHE MEM 0.02 1
  GR 3 11000 0.06
  STORE CACHE MEM 0.02 512
LOOP VIDEO 28
```

```
MATRIX:
  LOAD CACHE MEM 0.02 1
  LRB 3 63 0.06
  STORE CACHE MEM 0.02 8
```

```
PARA1:
  LOAD CACHE MEM 0.02 1
  LOAD CACHE MEM 0.02 0
  LOAD CACHE MEM 0.02 0
  ACT 4000000000 0.5
  READY core2
  LOAD CACHE L3 0.02 1
  ACT 1000000000 0.3
  STORE CACHE MEM 0.02 32
LOOP PARA 2000000
```

```
PARA2:
  LOAD CACHE MEM 0.02 1
  ACT 2000000000 0.5
  STORE CACHE L3 0.02 512
  ACT 0.0 0.0
  WAIT_ON core1
LOOP PARA 2000000
```

Fig. 6. Traces of the algorithms. `LR` LR-decomposition, `VIDEO` video streaming, `MATRIX` matrix multiplication, `PARA1` and `PARA2` dual-core application.

with three bits, temperature, ranging from 50 °C to 90 °C, with five bits, and used one bit to indicate an error (either timing or soft error). The action consisted of setting frequency and voltage. For this evaluation, we treated frequency and voltage separately, although a more realistic simulation would use only valid frequency-voltage pairs.

We modeled the problem of controlling the operating point of the MPSoC as a single-step problem of the XCS. We could not model the problem as a multi-step problem, because the reinforcement program could not signal the end of the problem, that is when the optimal operating point is reached.

4.3 Evaluation of XCS control

The XCS generates its classifiers during an initial training phase. In the training phase, a random frequency and voltage was set, the system ran for 2 seconds simulated time, the current state of the environment was reported to the XCS, which then decided on an action (a frequency-voltage setting), and received a reward for this decision after an additional two seconds of simulated time. We repeated this training phase until all possible frequency-voltage pairs were tested sufficiently often, in this case 50 000 times. Also, between the runs, temperature was raised randomly by 5 K, 10 K, 20 K, or 30 K. The waiting period of two seconds was needed to let temperature reach a stable state. During the training phase, the XCS runs only on one core with an activity factor of 0.05 without cache access. The other cores were idling at 2000 MHz at 1.2 V. This setup allowed a significantly smaller simulation time than running the traces. During training, we set [28] the don't-care probability $P_{\#} = 1.0$, the exploration

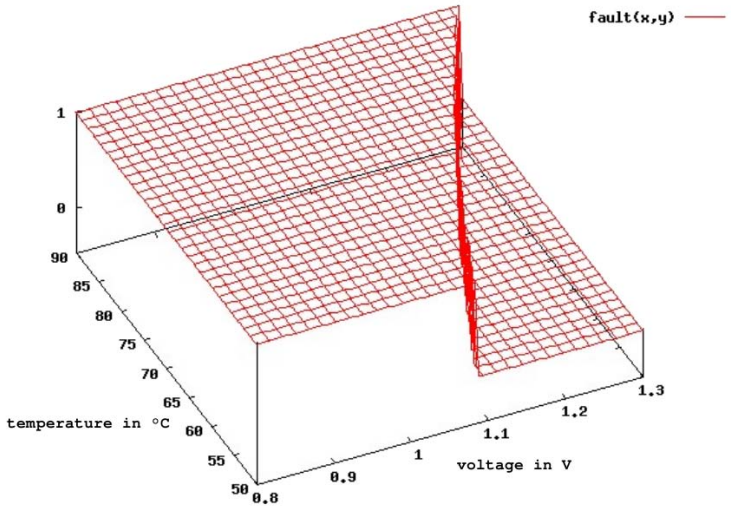

Fig. 7. Timing errors dependency on temperature and voltage at 2000 MHz.

probability $P_{\text{explr}} = 1.0$, the genetic algorithm threshold $\theta_{\text{GA}} = 25$, the deletion threshold $\theta_{\text{Del}} = 20$, and the merging threshold $\theta_{\text{Sub}} = 25$.

The reward function for the training phase should reflect that the XCS should minimize power consumption, maximize performance, while keeping the error rate low. This resulted in the following reward function:

$$R(f, p, t, v) = w_1 \frac{f}{f_{\max}} + w_2 \left(1 - \frac{p}{p_{\max}}\right) + w_3 \text{rel}(t, v, f) \tag{11}$$

$\text{rel}(t, v, f)$ models the reliability and is 0 in case of an error and 1 otherwise. We chose $w_1 = 200$, $w_2 = 35$, and $w_3 = 200$ to indicate that the system should reach high frequency values while keeping power consumption low. Figure 7 shows the fault count depending on temperature and voltage at 2000 MHz. We can see, that [2000 MHz, 1.2 V] is a safe setting in terms of faults at a usual temperature range (up to 70 °C).

Simple control After training, we let XCS control an MPSoC running each algorithm on a core. We set the explore probability $P_{\text{explr}} = 0$ to inhibit exploration and let the XCS always choose the classifier promising the highest reward. Setting the thresholds $\theta_{\text{GA}} = 0$, $\theta_{\text{Del}} = 0$, and $\theta_{\text{Sub}} = 0$ inhibits the genetic algorithm and avoids deleting or merging of classifiers. The only possibility for new classifiers was through covering. Every 10 s a random frequency and voltage was set. Figure 8 shows the resulting frequency and voltage settings. We observe that the XCS resets the frequency and voltage to 2000 MHz and 1.2 V which leads to no errors in the actual temperature range and is the optimal setting.

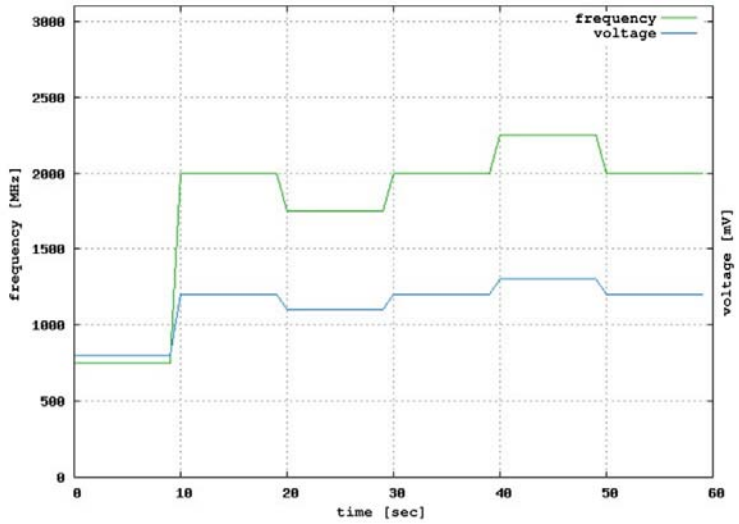

Fig. 8. Control reaction of the XCS after setting random frequency and voltage every 10 sec.

Control under changed environment We also evaluated XCS' behavior to changed environment settings. For this, we changed the $rel(t, v, f)$ function in (11) to

$$rel(t, v, f) = \begin{cases} 0 & \text{if timing error} \\ (\frac{70}{t})^2 & \text{if } t > 70 \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

and changed the weights to $w_1 = 200$, $w_2 = 35$, and $w_3 = 200$ if temperature was below 70 °C and $w_1 = 100$, $w_2 = 100$, and $w_3 = 200$ if temperature was above 70 °C. This represents an emergency behavior which allows the XCS to use a less performing setting and shows how the designer's prior knowledge may enter the XCS control mechanism. Figure 9 shows the result. We observe that 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.

Learning without genetic algorithm As the final ASoC probably won't have a genetic algorithm implemented (because of the large population size and thus large memory requirement the genetic algorithm needs to operate), we evaluated XCS's capabilities to learn a different reward function while its genetic algorithm is disabled. For this, we used the classifiers learned previously with (11) and altered the training phase slightly to reduce the time needed for learning,

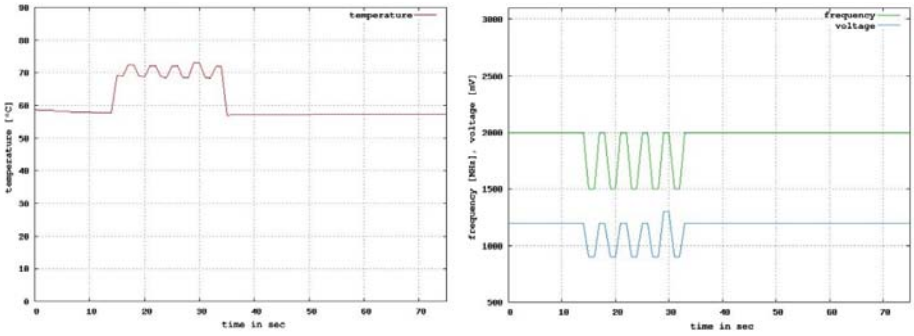

Fig. 9. XCS's behavior to a temperature raise of 15 K at $t = 15$ s.

by keeping the current frequency-voltage pair until the XCS finds a higher rewarded one (instead of randomly setting frequency-voltage pairs). This restricts the problem space to the more interesting frequency-voltage pairs. Furthermore, we increased the learning rate β from 0.2 to 1.0 to help forgetting the original reward function.

Figure 10 shows the frequency-voltage pairs the XCS tries out to learn the following new reward function, which aims to minimize the waiting time w of Core 2 for Core 1 (and thus keep total run time low):

$$R(f,p,t,v,w) = w_1 \text{time}(w) + w_2 \left(1 - \frac{p}{p_{\max}}\right) + w_3 rel(t,v,f) \qquad (13)$$

Here, $rel(t,v,f)$ is the same as in (12) and

$$\text{time}(w) = \begin{cases} 1 - \frac{w}{w_{\max}} & \text{if the waiting time of Core 1=0} \\ 0 & \text{otherwise} \end{cases}$$

We observe that, despite the high learning rate, the XCS needs a long time to learn the new reward function, but nonetheless succeeds.

5 Conclusion and future work

The paper showed the current state of ASoC design methodology. We can estimate the reliability of a system at design time, allowing us to incorporate reliability as an optimization criterion for designs. Applying our reliability analysis on an example showed the impact of power management on system reliability.

Furthermore, this paper showed our first evaluation of the learning classifier system XCS to control a SoC. 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, once XCS is implemented on an ASoC. The results encourage to further investigate the capabilities of XCS.

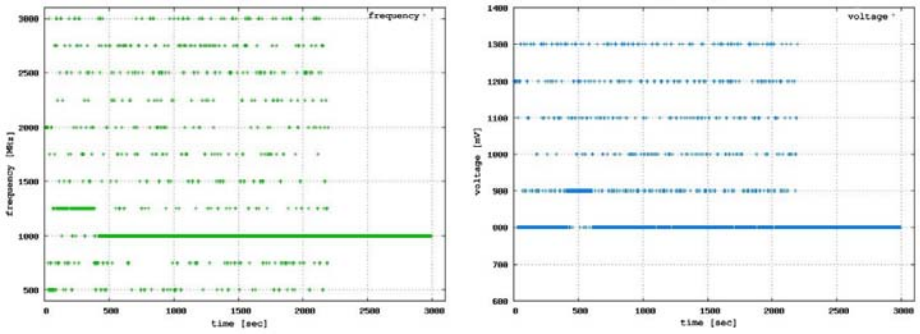

Fig. 10. XCS learning different reward function with disabled genetic algorithm and using initial classifiers.

Future work on evaluating the XCS will include the evaluation of realistic applications, for example communicating applications or shared memory usage, a better measurement for performance, and choosing only from fixed set of frequency-voltage pairs. We will also investigate how we can prevent the oscillating behavior of the XCS we observed when we raised the temperature of the environment.

6 Acknowledgments

This work is part of the Organic Computing initiative carried out by the Deutsche Forschungs Gemeinschaft. We thank the Deutsche Forschungs Gemeinschaft for their grant under which this work has been carried out. We also thank Johannes Zeppenfeld for developing and coding large parts of `libasoc` (the ASoC simulation library), and the team of Prof. Herkersdorf at the TU München for fruitful discussions.

References

1. Edenfeld, D., Kahng, A.B., Rodgers, M., Zorian, Y.: 2003 Technology Roadmap for Semiconductors. Computer **37** (2004) 47–56
2. Narayanan, V., Xie, Y.: Reliability Concerns in Embedded System Designs. Computer **39** (2006) 118–120
3. Borkar, S.: Designing Reliable Systems From Unreliable Components: The Challenges of Transistor Variability and Degradation. IEEE Micro **25** (2005) 10–16
4. JEDEC: Failure mechanisms and models for semiconductor devices. Technical Report JEP122C, JEDEC Solid state technology association (2006)
5. Shooman, M.L.: Probabilistic Reliability: An Engeneering Approach. 2 edn. Robert E. Krieger Publishing Componay, Malabar, Florida (1990)
6. Neubeck, K.: Practical Reliability Analysis. Prentice Hall (2003)

7. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: Lifetime Reliability: Toward an Architectural Solution. IEEE Micro **25** (2005) 70–80
8. Gupta, A., Dutt, N., Kurdahi, F., Khouri, K., Abadir, M.: Floorplan driven leakage power aware IP-based SoC design space exploration. In: CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, ACM (2006) 118–123
9. Holland, J.H.: Adaptation. In Rosen, R., Snell, F.M., eds.: Progress in theoretical biology, New York, Academic Press (1976) 263–293
10. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation **3** (1995) 149–175
11. Kovacs, T.: XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, Pant, eds.: Soft Computing in Engineering Design and Manufacturing, Springer-Verlag, London (1997) 59–68
12. Butz, M.V.: An Implementation of the XCS classifier system in C. Technical Report 99021, The Illinois Genetic Algorithms Laboratory (1999)
13. Dorigo, M.: ALECSYS and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. Machine Learning **19** (1995) 209–240
14. Open SystemC Initiative (OSCI): SystemC. http://www.systemc.org (1999)
15. Forschungszentrum Informatik – Forschungsbereich Systementwurf in der Mikroelektronik: SysXplorer. http://www.fzi.de/sim/sysxplorer.html (2008)
16. Li, Y.T., Malik, S., Wolfe, A.: Efficient microarchitecture modeling and path analysis for real-time software. In: Proceedings of the Real-Time Systems Symposium (RTSS), Los Alamitos, CA, USA, IEEE Computer Society (1995) 298
17. Bergamaschi, R.A., Jiang, Y.W.: State-based power analysis for systems-on-chip. In: DAC '03: Proceedings of the 40th conference on Design automation, ACM (2003) 638–641
18. Skadron, K., Stan, M.R., Huang, W., Velusamy, S., Sankaranarayanan, K., Tarjan, D.: Temperature-aware microarchitecture. SIGARCH Comput. Archit. News **31** (2003) 2–13
19. Benini, L., Bogliolo, A., Micheli, G.D.: A survey of design techniques for system-level dynamic power management. IEEE Trans. Very Large Scale Integr. Syst. **8** (2000) 299–316
20. IBM: PowerPC 750 Microprocessors. (2008)
21. Butts, J.A., Sohi, G.S.: A static power model for architects. In: MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture, New York, NY, USA, ACM (2000) 191 –201
22. Weste, N., Eshraghian, K.: Principles of CMOS VLSI Design: A Systems Perspective. 2nd edn. Addison-Wesley (1993)
23. Genossar, D., Shamir, N.: Intel® Pentium® M Processor Power Estimation, Budgeting, Optimization, and Validation. Intel Technology Journal **7** (2003) 44–49
24. Golda, A., Kos, A.: Temperature Influence on Power Consumption and Time Delay. In: Proc. Euromicro Symposium on Digital Systems Design, IEEE Computer Society (2003) 378
25. Zhu, D.: Reliability-Aware Dynamic Energy Management in Dependable Embedded Real-Time Systems. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), Los Alamitos, CA, USA, IEEE Computer Society (2006) 397–407
26. Burd, T., Brodersen, R.: Energy efficient CMOS microprocessor design. Hawaii International Conference on System Sciences (1995) 288

27. AMD: AMD Opteron Processor Family. http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8825,00.html (2008)
28. Butz, M., Wilson, S.W.: An Algorithmic Description of XCS. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems. Number 2321 in Lecture Notes in Artificial Intelligence, London, UK, Springer-Verlag (2001) 253–272