Evaluation of the Learning Classifier System XCS for SoC run-time control *

Andreas Bernauer, Dirk Fritz, Wolfgang Rosenstiel
Wilhelm-Schickard-Institute of Computer Science,
Department of Computer Engineering
72076 Tübingen, Sand 13, Germany
bernauer@informatik.uni-tuebingen.de

Abstract: In this paper, we evaluate the feasibility of using the learning classifier XCS to control a System-on-Chip. Increasing number of transistors and process variation make it difficult for a chip designer to foresee all possible run-time conditions. Postponing some decisions from design time to run time alleviates the designer's life and allows shorter time-to-market. In this paper, we evaluate if XCS can take these run-time decisions on a processor with four cores. The evaluation shows that 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.

1 Introduction

The number of System-on-Chip (SoC) designs is expected to increase strongly according to the International Technology Roadmap for Semiconductors [EKRZ04]. 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. In particular, changed electrical circuit properties, the susceptibility to internal and external noises and accelerated aging pose great challenges [NX06, Bor05].

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) [LHR+05], which takes at run time the decisions that the designer formerly took at design time. The chip is getting aware of itself and gains organic properties such as the ability to react to

*This work is partially funded by Deutsche Forschungsgemeinschaft, Priority Program Organic Computing (SPP 1183) under the grants HE 4584/3-2 and RO 1030/14-2.

unforeseen situations. 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. To the best of our knowledge, no attempt has yet been made to use XCS to control a SoC.

This paper is organized as follows: Section 2 summarizes related work. Section 3 and 4 describe the models and the experimental setup, respectively. After the results, which Section 5 presents, Section 6 concludes this paper and shows direction of future work.

2 Related work

Holland et al. [Hol76] 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. [Wil95] presented a specialized version of an LCS, the XCS, for which studies showed [Kov97] that it learns accurate, complete, and minimal representations for boolean functions. Butz [But99] 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 [Dor95], 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 Models

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

For the performance, we use the frequency as a (rough) estimate. Later setups will include more sophisticated performance measures. We get the temperature estimates from Hotspot [SSH+03] based on our power estimates.

The power consumption consists of the static and dynamic power dissipation ($P_{\mathrm{total}} = P_s + P_d$). For the static power dissipation P_s, we use the model of Butts et al. [BS00] with $P_s = V_{\mathrm{DD}} N k_{\mathrm{design}} \hat{I}_{\mathrm{leak}}$, with supply voltage V_{DD}, N transistors, and design and technology dependent parameters k_{design} and $\hat{I}_{\mathrm{leak}}$ (given in [BS00]). For the dynamic power dissipation P_d, we add an activity factor α to the well-known model [WE93] as done by Intel to estimate power dissipation in the Pentium M [GS03] $P_d = \alpha C_L V_{\mathrm{DD}}^2 f_p$, where 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.

Defining an accurate model for timing errors is difficult, as the timing error depends on the

actual path the signal is taking. Therefore, we use a simple model where we assume a fixed set of inverters between the two pipeline stages modeling either the longest or the average path length. We model the average switching time of an inverter using the temperature dependent model from [GK03]:

$$t_{\mathrm{av}} = \frac{(t_{\mathrm{dr}} + t_{\mathrm{tempdelay}}(T)) + (t_{\mathrm{df}} + t_{\mathrm{tempdelay}}(T))}{2} \tag{1}$$

t_{dr} and t_{df} are the well-known raise and fall delays of a signal on an inverter [WE93] (which depend on voltage) and $t_{\mathrm{tempdelay}}$ models the influence of the temperature on the time delay as shown in [GK03].

We use the model of Zhu et al. [Zhu06] to model the effect of frequency and voltage scaling on fault rates with $\lambda_0 10^{\frac{(1-f)d}{1-f_{\min}}}$, where λ_0 is the average fault rate corresponding to $V_{\max}$ and $f_{\max}$ and d is a constant. 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 Experimental setup

This section describes 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 used for evaluation.

We chose the AMD Opteron (Barcelona) processor as a multi-processor SoC (MPSoC) hardware, which is a four-core general purpose processor produced on a single die. The advantage of using the Opteron as an MPSoC is 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 1 shows the floor plan of the Opteron, as derived from [AMD08]. We adjust the activity factor of the cores so that we meet the thermal design power and average CPU power. Each core is controlled by one XCS which runs on dedicated hardware, so regular core operation is not interrupted.

We execute four algorithms on the hardware: LR-decomposition, video filtering, matrix multiplication and a dual-process application where one process has to wait for the other. We simulate the algorithms with traces, which only describe the memory access patterns, needed computation time and the activity factory without computing anything actually. This decreases simulation time while still allowing for good estimates. When all cores execute an algorithm, power consumption lies at 83 W and temperature at 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 [AMD08].

We use eleven different frequencies (500 MHz–3000 MHz, encoded in four bits) and five voltage levels (0.8 V–1.3 V, encoded in three bits). Temperature range is 50 °C–90 °C, encoded in five bits. The action consisted of setting frequency and voltage. For this eval-

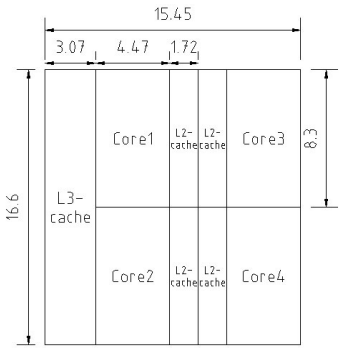

Figure 1: Floorplan of the Opteron. Measurements are given in mm.

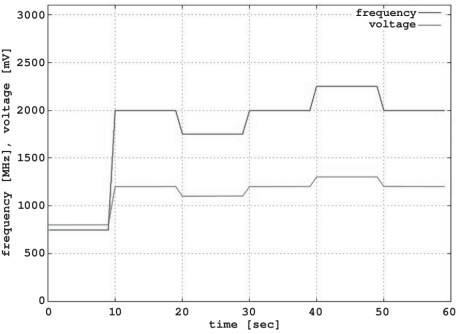

Figure 2: Control reaction of the XCS after setting a random frequency and voltage every 10 sec.

uation, we treated frequency and voltage separately, although a more realistic simulation would use only valid frequency-voltage pairs.

Given a trained XCS, we evaluated three scenarios: simple control, control under changed environment, and online learning without genetic algorithm. In the simple-control scenario, the XCS should find the optimal operating point under training conditions. In the changed-environment scenario, the XCS should find the optimal operation point although we change the environment and the reward function from their settings during training. In the online learning scenario we evaluate whether the XCS can learn a new reward function without its genetic algorithm, as on a SoC the genetic algorithm usually won't be available. All scenarios are modeled as single-step problems of the XCS, as for a multi-step problem the optimal operating point generally has to be known in advance (to signal the end of the problem to the XCS and distribute the reward) [BW01].

We train the XCS on a single core with an activity factor (see Section 3) of 0.05 without simulating cache access. The other cores are idling at 2000 MHz at 1.2 V. This setup allows a significantly smaller simulation time than running the algorithms. We did 50 000 repetitions, until all possible frequency-voltage pairs are tested sufficiently often. Between the runs, temperature is raised randomly by 5 K, 10 K, 20 K, or 30 K from default. For the online-learning scenario, we used $\beta = 1.0$ to reduce the time needed for learning.

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

$$R(f,p,t,v) = w_1 \frac{f}{f_{\max}} + w_2 \left(1 - \frac{p}{p_{\max}}\right) + w_3 \mathrm{rel}(t,v,f) \tag{2}$$

$\mathrm{rel}(t,v,f)$ models the reliability and is 0 in case of an error and 1 otherwise. We choose $w_1 = 200$, $w_2 = 35$, and $w_3 = 200$. Figure 3 shows the fault count depending on

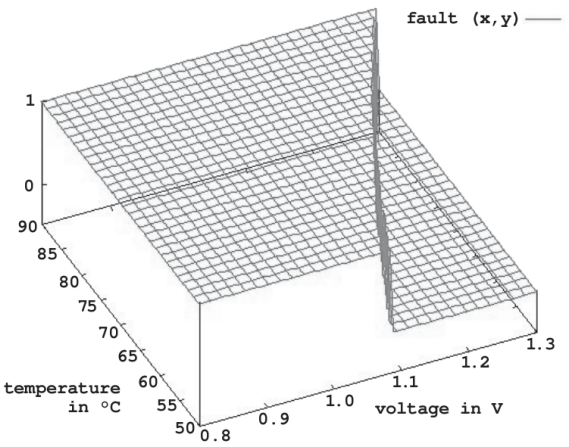

Figure 3: Timing errors dependency on temperature and voltage at 2000 MHz.

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).

In the scenario with changed environment, we raised temperature by 15 K. Also, we changed the $rel(t, v, f)$ function in (2) to

$$rel(t, v, f) = \begin{cases} 0 & \text{if timing error} \\ (\frac{70}{t})^2 & \text{if } t > 70 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

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.

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

$$R(f, p, t, v, w) = w_1 \text{time}(w) + w_2 \left(1 - \frac{p}{p_{\max}}\right) + w_3 rel(t, v, f) \tag{4}$$

Here, $rel(t, v, f)$ is the same as in (3) and

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

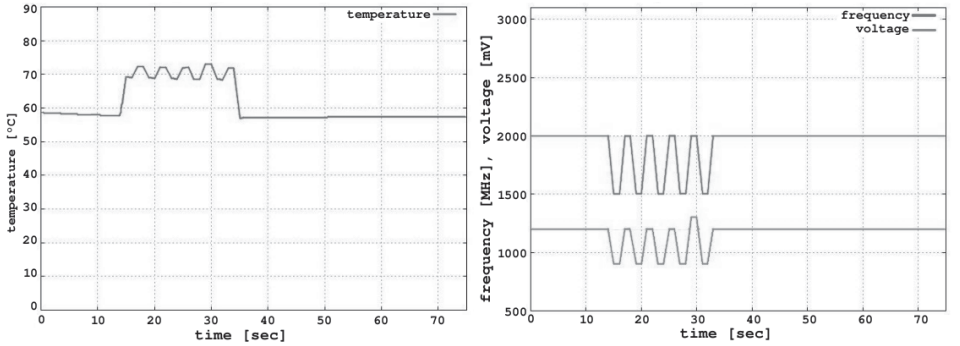

Figure 4: XCS's behavior to a temperature raise of 15 K at $t = 15\,\mathrm{s}$.

5 Results

After training, we let XCS control an MPSoC running an algorithm on each core. Each XCS contained about 600 rules, which would require about 8 kB (104 bit per rule including 84 bit solely for rule parameters such as fitness, etc.) if we implemented it unmodified on the SoC. We inhibit exploration and the genetic algorithm to avoid creating new classifiers and to simulate the situation of the XCS on a SoC. In the simple-control scenario, every 10 s of simulation time a random frequency and voltage was set. Figure 2 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 (see Figure 3) and is the optimal setting.

Figure 4 shows the result for the changed-environment scenario. 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.

For the online learning scenario, Figure 5 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). 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.

6 Conclusion and future work

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

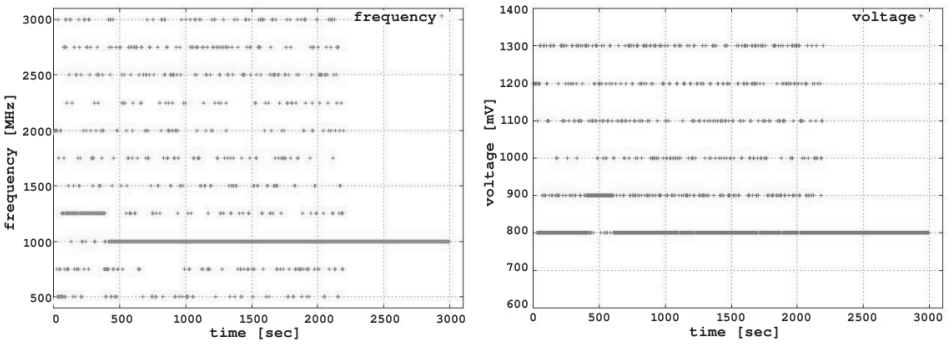

Figure 5: XCS learning a different reward function with disabled genetic algorithm and using initial classifiers. Each point represents a frequency or voltage setting the XCS tries out. After about 2200 sec. this exploration mode is turned off, forcing the XCS to always choose the action with the highest prediction.

functions without a functioning genetic algorithm, as it will be the case, once XCS is implemented on an ASoC. The results encourage us to further investigate the capabilities of XCS.

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 a fixed set of frequency-voltage pairs. Furthermore, we will investigate ways to reduce the necessary memory footprint to store the XCS on the SoC, for example by not storing all rule parameters and dropping low-accurate rules. We will also investigate how we can prevent the oscillating behavior of the XCS we observed when we raised the temperature of the environment and how further actions such as changing bus width or turning off processors affect the performance of the XCS.

Acknowledgments

We thank Johannes Zeppenfeld for his support in developing and coding large parts of `libasoc` (the ASoC simulation library written in SystemC), with which the simulation model has been described.

References

[AMD08] AMD. AMD Opteron Processor Family. `http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8825,00.html`, 2008.

[Bor05] Shekhar Borkar. Designing Reliable Systems From Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, Novem-

ber/December 2005.

[BS00] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 191 –201, New York, NY, USA, 2000. ACM.

[But99] Martin V. Butz. An Implementation of the XCS classifier system in C. Technical Report 99021, The Illinois Genetic Algorithms Laboratory, 1999.

[BW01] Martin Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, number 2321 in Lecture Notes in Artificial Intelligence, pages 253–272, London, UK, 2001. Springer-Verlag.

[Dor95] Marco Dorigo. ALECSYS and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19(3):209–240, 1995.

[EKRZ04] Don Edenfeld, Andrew B. Kahng, Mike Rodgers, and Yervant Zorian. 2003 Technology Roadmap for Semiconductors. *Computer*, 37(1):47–56, 2004.

[GK03] A. Golda and A. Kos. Temperature Influence on Power Consumption and Time Delay. In *Proc. Euromicro Symposium on Digital Systems Design*, page 378. IEEE Computer Society, 2003.

[GS03] Dani Genossar and Nachum Shamir. Intel® Pentium® M Processor Power Estimation, Budgeting, Optimization, and Validation. *Intel Technology Journal*, 7(2):44–49, May 2003.

[Hol76] John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology*, pages 263–293, New York, 1976. Academic Press.

[Kov97] Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, London, 1997.

[LHR+05] Gabriel Lipsa, Andreas Herkersdorf, Wolfgang Rosenstiel, Oliver Bringmann, and Walter Stechele. Towards a Framework and a Design Methodology for Autonomic SoC. In *2nd IEEE International Conference on Autonomic Computing*, Seattle, USA, June 13-16 2005.

[NX06] Vijaykrishnan Narayanan and Yuan Xie. Reliability Concerns in Embedded System Designs. *Computer*, 39(1):118–120, 2006.

[SSH+03] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. *SIGARCH Comput. Archit. News*, 31(2):2–13, 2003.

[WE93] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 2nd edition, 1993.

[Wil95] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[Zhu06] Dakai Zhu. Reliability-Aware Dynamic Energy Management in Dependable Embedded Real-Time Systems. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 397–407, Los Alamitos, CA, USA, 2006. IEEE Computer Society.