

CARUSO – an Approach Towards a Network of Low Power Autonomic Systems on Chips for Embedded Real-time Applications

Uwe Brinkschulte^{*}, Jürgen Becker[#], and Theo Ungerer⁺

^{*} *University of Karlsruhe, Department of Computer Science,
brinks@ira.uka.de*

[#] *University of Karlsruhe, Department of Electrical Engineering,
Becker@itv.uni-karlsruhe.de*

⁺ *University of Augsburg, Department of Applied Computer Science,
Theo.Ungerer@Informatik.Uni-Augsburg.de*

Abstract

This paper proposes CARUSO – a new SoC approach that emphasizes Connectivity, Autonomic computing principles, Real-time, and Ultra-low power requirements. The requirements shall be fulfilled by a multithreaded processor core within a reconfigurable SoC. A helper thread running with low priority in an own thread slot concurrent to the application implements an autonomic manager that monitors the application. A middleware decides if self-optimization, self-configuration, self-protection, or self-healing techniques must be triggered based on the autonomic manager's information and further application knowledge.

1. Introduction

State-of-the-art embedded computing systems with limitations in space and energy consumption are manufactured as so-called Systems on Chip (SoC), where all the electronic components are placed on a single chip. CARUSO (Connective Autonomic Real-time Ultra-low-power System on Chip) is a new SoC project aiming to integrate hardware, software and configware for high performance embedded computing with respect to other requirements:

- **Autonomic Computing** principles like self-optimization, self-configuration, self-protection, and self-healing to allow the development of robust, flexible and highly adaptive embedded systems. Future embedded systems

should work with a minimum of human interaction. Therefore, they must be able to adapt themselves automatically to changing conditions, failures, faulty operations or attacks.

- **Connectivity** to enable several SoCs to dynamically form ad-hoc networks. Future embedded systems will consist of multiple small computing components which cooperate to solve a common task. Examples are ubiquitous computing systems or swarms of small robots. Often, the networks must be established dynamically. The principles of autonomic computing can require a change of the network topology for an optimization, reconfiguration or a self-healing process.
- **Real-time** capabilities to keep the timing requirements introduced by many embedded applications. Depending on the application, hard, firm or soft real-time properties can be necessary.
- **Ultra-low-power** to increase the battery life time for mobile applications and to reduce the heat dissemination in temperature-critical environments. Especially if many small computation nodes are used to realize a larger embedded system, the energy consumption of a single node as well as the overall energy consumption is important.
- **Cost and space** reduction to fit the needs of upcoming embedded systems. The reduction of development and operational costs is always important for embedded systems. New applications introduce additional strong space limitations. Instead of one central computer,

networks of small computing nodes are used because they are more flexible and space-saving. New robot designs e.g. tend to use an individual computing node for every hinge, sensor or actor.

The main research challenge of this approach is not to treat each requirement isolated, but to explore the relationship between all requirements in a general view and to exploit synergy and side effects in an optimal manner. For example, optimizing the overall energy consumption of a distributed system including hardware, software and middleware increases the optimization space compared to the optimization of a single node. Additional knowledge on the current task distribution, real-time constraints and the execution history can be taken into account to dynamically reduce the energy consumption according to the self-optimization and self-reconfiguration principles of the autonomic computing paradigm.

The CARUSO approach meets the global challenge of a vertical system integration of application, interconnection by middleware, software, configware, and hardware. The application should have a unique and homogeneous view on different aspects like autonomy, communication, mobility, timeliness, energy consumption, fault tolerance and fault repair.

The main focus of the project is to create and to realize a hardware and software architecture for CARUSO. This covers research topics from different fields like autonomic computing, middleware for real-time systems, multithreaded processor architectures, helper thread concepts, reconfigurable SoCs and energy-efficient hardware and software-design. Our solution proposes a multithreaded processor core that uses one or more threads for the (real-time) application and another thread slot for an autonomic manager to monitor the autonomic computing principles. A processor-integrated thread scheduling technique allows to guarantee each thread a specific percentage of the SoC's processing power on a fine-grained basis. Threads are isolated against each other. The currently needed processing power defined by the cumulative percentages of the active threads is used to manage energy consumption of the SoC (as proposed in [27]). In our understanding, a new class of SoCs is the most promising way to integrate all the necessary attributes.

2. State of the Art

Autonomic Computing [1, 2] has been introduced by IBM in the beginning of this millennium. The basic idea is to make computing systems behave more like organic entities, which adapt to new challenges, heal themselves after injuries and protect themselves against attacks. This new paradigm looks very promising for the management of complex computing systems like tomorrow's embedded systems, where hundreds of parameters have to be configured and optimized and where dependability is a main keyword. Current research approaches deal with single parts of the Autonomic Computing ideas, but there is no general system architecture. Furthermore, to our knowledge the paradigm has not been introduced to the fields of embedded real-time Systems on Chip yet.

To manage highly connective distributed systems, middleware is a well known and researched approach. State-of-the-art middleware is available for general distributed systems, embedded systems and real-time systems [3, 4, 5, 6, 7, 8, 9]. Autonomic Systems on Chip introduce new challenges for middleware: Future SoCs will contain more and more dynamically reconfigurable hardware [24, 25, 26]. This allows to decide at runtime, if a certain functionality is realized by a software or a hardware module. The middleware will have to handle such distributed software and hardware modules, which have to be reconfigured or even migrated in real-time between the computation nodes. In general, middleware will become a key component to realize Autonomic Computing principles like self-optimization, self-configuration and self-healing in a highly distributed system. Research on this topic has started, see e.g. [10]. Finally, the influence of middleware on the energy consumption of a distributed system has not been explored. There are two basic questions: firstly, how can middleware optimize the energy consumption of the entire system by choosing a smart distribution of hardware and software modules? Secondly, can the resource needs (memory, computing power) of a middleware be reduced beyond the current state-of-the-art to reduce the energy consumption of a single SoC?

Multithreaded processor architectures support the execution of multithreaded programs by special hardware measures like multiple register sets, multiple program counters and special pipeline design to allow the mixed pipelined execution of instructions from different threads [11]. Several

newly introduced high-end processors [12, 13], network processors [14], and signal processors [28] use hardware multithreading to reduce the latencies in program execution. The multithreaded Komodo microcontroller [15, 29] has been developed to explore the properties of hardware multithreading with hardware-integrated real-time scheduling for embedded real-time systems. The multithread application-specific extension (MT-ASE) of MIPS proposes to use multithreading on the ASIC level [30]. The integration of a dynamically reconfigurable SoC and a multithreaded processor core has not been explored yet even so several advantages can be expected: latencies caused by the dynamic reconfiguration could be masked by the execution of another thread, real-time scheduling could be done within the processor core, the energy consumption might be reduced (see below) and the helper thread concept could support Autonomic Computing.

Helper threads are threads, which are separated from the normal program flow of a multithreaded processor core to support system operations and management. This concept has been introduced on multithreaded processors to support branch prediction, trap handling or cache preloading [16, 17, 18]. In [19], the helper thread concept is proposed for real-time garbage collection and real-time debugging. Helper threads seem to be a very suitable concept to support the self-management necessary for Autonomic Computing.

Reconfigurable SoCs [24, 25, 26] contain a processor core and a reconfigurable part, usually an FPGA. The reconfigurable part is used to adapt the SoC to a specific task. Reconfiguration can be done statically or dynamically. In static reconfiguration, the SoC is preconfigured for the given task and the configuration never changes during runtime. In dynamic reconfiguration, the SoC changes parts or the complete configuration during runtime. This is usually done to optimize performance by replacing software by hardware execution. Reconfiguration can be done on a fine-grained (gates) or course-grained (function blocks) level. The latter stretches from reconfigurable data paths between existing ALUs to reconfigurable functional units which reconfigure according to the current needs to become e.g. an integer unit, a floating point unit, etc. Combining dynamically reconfigurable SoCs with a multithreaded processor core and the principles of Autonomic Computing is a new promising approach which will be followed in the CARUSO project.

Energy efficient design is a very important current research focus. On the electrical level,

techniques like pipeline gating, frequency and voltage scaling to reduce the energy consumption caused by state switching are state of the art. On the microarchitectural and architectural level of a processor, concepts like increasing the code density, reducing external bus transfers and power management are well known. Power management is done on the pure hardware level or supported by software. On the hardware level, the processor pipeline turns off not needed processor components [20]. On the software level, the operating system tells the processor to turn off components or to reduce the clock frequency [21]. For multithreaded processors, only little research is done on energy reduction techniques. The main approach is to avoid the energy consumed by speculation misses [22, 23]. Instead of speculative execution, instructions from other threads are used to fill the instruction window of the processor. In the CARUSO project, a general system architecture for energy saving will be developed. The project aims to explore the benefits which can be reached by combining the known techniques with new energy saving approaches in hardware reconfiguration, multithreaded processor cores and middleware.

3. Research Objectives

A main research focus of the CARUSO project is not to have an isolated view on each requirement, but to explore and exploit the relationship between the requirements and attributes. The general view is expected to reveal synergy effects to improve the overall system. Here are the main questions to be answered by the project:

- *Autonomic Computing*
 - Which interrelationships exist between self-optimization, self-protecting, self-healing and self-reconfiguration?
 - How can this be supported by constructing dynamic networks of SoCs?
 - What is the influence of the Autonomic Computing principles on the real-time properties of the system?
 - What are the advantages of the dynamic hardware reconfiguration feature available in the SoC for Autonomic Computing?
 - What is the influence of the multithreaded processor core?

- *Energy consumption*
 - Does the overall optimization of the distributed system consisting of hardware,

software, middleware and configware lead to better results than the optimization of the single components?

- Can known time constraints in distributed systems be used as additional information source for resource usage and an energy-optimal distribution?
 - Is the possibility of software- and hardware-reconfiguration usable to reduce the energy consumption?
 - And again, what is the influence of the multithreaded processor core?
- *Connectivity*
 - What are the necessary features of the middleware in such a system?
 - How can middleware support energy saving and Autonomic Computing?
 - How the middleware is affected by the dynamic hardware reconfiguration feature of the processing nodes?
 - What kind of communication links are necessary for an optimal system performance?
 - *Real-time*
 - What is the impact of hard-real-time requirement on the architecture of the reconfigurable SoC and its multithreaded processor core?
 - How the real-time behavior is affected by dynamic hardware reconfiguration, Autonomic Computing and energy saving?
 - Can the real-time constraints deliver any additional information to support the requirements mentioned above?
 - How real-time scheduling is influenced?

The project aims to answer as many of these questions as possible. A prototypic implementation is planned to verify the results and to evaluate the system in a real-world application example.

4. CARUSO System Architecture

Figure 1 shows the proposed overall CARUSO system architecture. It is a distributed architecture consisting of hardware and software. The application is for demonstration and evaluation purpose and can be replaced by any other suitable application. Figure 2 shows the structure of a single CARUSO SoC in more detail. It consists of a multithreaded processor core, memory to store programs, data, and configuration, a power

management module and several dynamically reconfigurable modules.

The integration of a multithreaded processor core in a SoC is one of the key ideas of this project and supports the project goals in several ways:

- The main principles of autonomic computing can be realized using the helper thread concept. An autonomic manager helper thread can monitor the system and make decisions regarding optimization, failures or possible attacks.
- The energy consumption can be monitored as well by a helper thread, which controls the power management and decides (in cooperation with and based on information delivered by the operating system and the middleware) about deactivation of parts of the SoC or reduction of clock frequency and voltage.
- The real-time constraints can be monitored as well by a helper thread affecting the system configuration and the power management.
- Latencies caused by reconfiguration or other events can be bridged by switching to another thread.
- The fast context switch of a multithreaded processor core supports real-time processing, as shown in the Komodo project in particular if combined with scheduling policies like Guaranteed Percentage scheduling [15], which allows the time isolation of threads and supports power management.

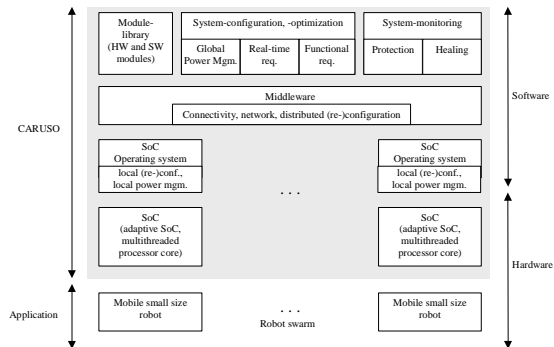


Fig. 1: CARUSO architecture

The power management is responsible for voltage and frequency scaling, a fine-grained deactivation of components and data paths and other energy saving techniques. It delivers status information to the multithreaded processor core (helper thread) and gets instructions from it. The

reconfigurable modules can be divided in two classes: The function units are responsible to perform tasks and services in hardware rather than in software and thereby gain performance or save energy. The interface & communication units have to form a flexible interface which can be adapted to the current needs. The interface allows the connection to sensors, actors and other SoCs to build a dynamic network of SoCs. The communication links can be wireless or wire-based.

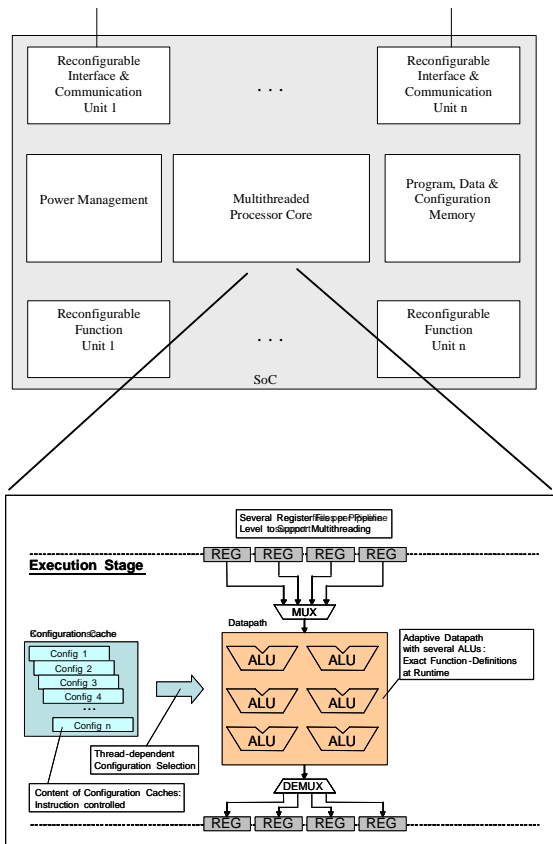


Fig. 2: The CARUSO SoC incl. adaptive multithreaded Data path

On top of each SoC, a special SoC operating system supports the application to use the SoC. This operating system runs on the multithreaded processor core and uses the reconfigurable units. Its main tasks are

- the local configuration and reconfiguration management. This means the administration of the reconfigurable resources and the administration of which modules and services

are currently executed in hardware or software on the device.

- the local power management. This is done in cooperation with the power management module on the chip and the global power management of the middleware layer.
- the support of the real-time scheduling on the multithreaded processor core.
- to provide an API to the higher layers of the system.

The middleware layer handles the global view of the distributed system. It dynamically establishes the communication links between the SoCs. According to the principles of Autonomic Computing, the overall system is monitored for optimization, protection and healing. As result of failures, attacks, power, performance or real-time requirements, a reconfiguration may be triggered. A module library provides modules and services in hardware and software for the allocation or reallocation on a specific SoC within the system. The middleware is therefore responsible for the global system management and optimization.

The application has been chosen as a typical example for the project aims: a swarm of small robots is collaborating to perform a given task (e.g. cleaning the floor). Each robot has limited energy resources, real-time requirements and must cooperate with the other robots in the swarm. The network is dynamic because robots might disappear (e.g. leaving the range of the wireless communication) or reappear. The swarm works autonomous, has to face failures or other problems of single robots while the overall performance has to be optimized. The application is therefore dedicated to demonstrate and to evaluate the CARUSO system.

5. Conclusions

We propose a new SoC approach called CARUSO that emphasizes connectivity, autonomic computing principles, real-time, and ultra-low power requirements. The requirements shall be fulfilled by a multithreaded processor core within a reconfigurable SoC. The autonomic manager is implemented as a helper thread running with low priority in an own thread slot concurrent to the application. The middleware decides if self-optimization, self-configuration, self-protection, or self-healing techniques must be triggered based on the autonomic manager's information and further application knowledge.

The CARUSO project is in its starting phase. Some of the questions might become obsolete and new ones might arise during the project duration. Theoretical and practical work has to be done to achieve the project goals. A prototypic implementation and a real-world application example are necessary to validate the expected results and to make them applicable for subsequent projects and products. This way, CARUSO aims to contribute to future embedded, ubiquitous and autonomous computing systems.

6. References

- [1] P. Horn: "Autonomic Computing: IBM's Perspective on the State of Information Technology", *IBM Manifesto*, Oktober 2001, <http://researchweb.watson.ibm.com/autonomic/manifesto>
- [2] J.O. Kephart, D.M. Chess: "The Vision of Autonomic Computing", *IEEE Computer*, Januar 2003, 41-50.
- [3] Object Management Group: "The Common Object Request Broker: Architecture and Specification", <http://www.omg.org>
- [4] Microsoft Corporation: ".NET framework", <http://www.microsoft.com/net/>
- [5] T.B. Downing: "JavaRMI: Remote Method Invocation", *IDG Books Worldwide*, February, 1998
- [6] Object Management Group: "MinimumCORBA 1.0", http://www.omg.org/technology/documents/specialized_corba.htm.
- [7] Ch. Gill, V. Subramonian, J. Parsons, H.-M. Hunag, S. Torri: "ORB Middleware Evolution for Networked Embedded Systems", *8th Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003)*, Guadalajara, Mexico, IEEE, 2003.
- [8] A.~D. McKinnon, D.~Bakken, J.~Shovic: "Micro-QoS-CORBA: A Reflective, QoS-Enabled, Configurable MicroCORBA With CASE Support", *Second Workshop on Real-time and Embedded Distributed Object Computing*, OMG, 2001.
- [9] U. Brinkschulte, A. Bechina, F. Picioroaga, E. Schneider: "Distributed Real-Time Computing for Microcontrollers - the OSA+ Approach", *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)*, Washington D.C., April 29 - May 1, 2002.
- [10] G. S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, R. Moreira, N. Parlavantzas: "Reflection, Self-awareness and Self-healing in OpenORB", *Workshop on Self-healing systems (WOSS)*, Charleston, South Carolina, 2002
- [11] T. Ungerer, B. Robic, J. Silc: "A Survey of Processors with Explicit Multithreading", *ACM Computing Surveys*, 35, 1, March 2003, 29-63.
- [12] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, S. R. Kunkel: "A Multithreaded PowerPC Processor for Commercial Servers", *IBM Journal Research and Development*, Volume 44, 2000, 885-898
- [13] D.T. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, M. Upton: "Hyper-Threading Technology Architecture and Microarchitecture: A Hypertext History", *Intel Technology Journal*, 2002, Volume 6
- [14] P.N. Glaskowsky: "Network Processors Mature in 2001", *Microprocessor Report*, February 2002
- [15] J. Kreuzinger, A. Schulz, M. Pfeffer, T. Ungerer, U. Brinkschulte, C. Krakowski: "Real-time Scheduling on Multithreaded Processors", *Real-Time Computing Systems and Applications (RTCSA)*, Cheju Island, South Korea, December 2000, 155-159.
- [16] R.S. Chappell et al.: "Simultaneous Subordinate Microthreading (SSMT)", *Proc. 26th ISCA*, Atlanta, GA, 2.-4.5.1999, 186-195.
- [17] S.W. Keckler, A. Chang, W.S. Lee, W.J. Dally: "Concurrent Event-handling Through Multithreading", *IEEE Trans. Comput.*, 48, 903-916.
- [18] C.B. Zilles, G.S. Sohi: "Execution-based Prediction Using Speculative Slices", *28th ISCA*, Göteborg, Schweden, 30.6.-4.7.2001, 2-13.
- [19] M. Pfeffer, S. Uhrig, Th. Ungerer, U. Brinkschulte: "A Real-time Java System on a Multithreaded Java Microcontroller", *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)*, Washington D.C., April 29 - May 1, 2002, 34-41.
- [20] R.G. Gonzales: "Micro-RISC Architecture for the Wireless Market", *IEEE Micro Magazine*, July/August 1999
- [21] A. Weissel, F. Bellosa: "Process Cruise Control: Event-driven Clock Scaling for Dynamic Power Management", *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, Grenoble, France, 2002, 238-246
- [22] J.S. Seng, D.M. Tullsen, G.Z.N. Cai: "Power-Sensitive Multithreaded Architecture", *Proc. 2000 IEEE*

Int. Conf. on Computer Design: VLSI in Computers and Processors, Austin, TX, 2000, 199-206

[23] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, P.W. Cook: "Power-aware Microarchitecture: Designing and Modeling Challenges for Next-generation Microprocessors", *IEEE Micro Magazine*, 2000, Volume 6, 26-44

[24] K. Compton, S. Hauck: "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, 34, 2, June 2002, 171-210.

[25] J. Becker, A. Thomas, M. Vorbach, V. Baumgarte: "An Industrial/Academic Configurable System-on-Chip Project (CSoC): Coarse-grain XPP-/Leon-based Architecture Integration", *Design, Automation and Test in Europe Conference (DATE 2003)*, Munich, March 2003

[26] J. Becker (Invited Tutorial): "Configurable Systems-on-Chip (CSoC)"; in: *Proc. of 9th Proc. of XV Brazilian Symposium on Integrated Circuit Design (SBCCI 2002)*, Porto Alegre, Brazil, September 5-9, 2002

[27] S. Uhrig, Th. Ungerer: "Fine-Grained Power Management for Real-Time Embedded Processors", to appear at: *RTS Embedded Systems*, Paris, March 30 – April 1, 2004

[28] E. Norden: "A Multithreading Extension for Low-Power, Low-Cost Applications", *Embedded Processor Forum*, 2003, http://www.infineon.com/cmc_upload/documents/082/795/TC2_MT_EPF03.pdf

[29] S. Uhrig, C. Liemke, M. Pfeffer, J. Becker, U. Brinkschulte, Th. Ungerer: "Implementing Real-time Scheduling Within a Multithreaded Java Microcontroller", *6th Workshop on Multithreaded Execution, Architecture, and Compilation MTEAC-6*, Istanbul, in conjunction with *35th International Symposium on Microarchitecture MICRO-35*, Workshop Proceedings, Nov. 2002, 57-64.

[30] R. Wilson: *MPF*: "MIPS Adds Multithreading to CPU for Net Apps", *iApplianceWeb* 10/14/2003.