Early ISS Integration into Network-on-Chip Designs

Andreas Wieferink[1], Malte Doerper[1], Tim Kogel[2], Rainer Leupers[1], Gerd Ascheid[1], and Heinrich Meyr[1]

[1] Institute for Integrated Signal Processing Systems, Aachen University of Technology, Germany
wieferink@iss.rwth-aachen.de
http://www.iss.rwth-aachen.de
and
[2] CoWare, Inc.
tim.kogel@coware.com
http://www.coware.com

Abstract. Future signal processing SoC designs will contain an increasing number of heterogeneous programmable units combined with a complex communication architecture to meet flexibility, performance and cost constraints. Designing such a heterogeneous MP-SoC architecture bears enormous potential for optimization, but requires a system-level design environment and methodology to evaluate architectural alternatives effectively.
Recently, efficient tool frameworks have been proposed to support the design space exploration for large scale embedded systems. The technique presented in this paper allows integrating retargeteable Instruction Set Simulators (ISS) into such an exploration framework very early in the design flow.
In a dual-processor JPEG decoding case study, we illustrate the effectiveness of this approach.

1 Introduction

The ever increasing System-on-Chip (SoC) complexity along with the ever shrinking time-to-market constraints pose enormous challenges to conceptualize, implement, verify and program today's SoCs. The traditional flow to directly implement the hardware on Register-Transfer-Level (RTL) after having a textual architecture specification available does not work for these complex embedded systems any more.

Thus, several flows have been suggested which close the gap between the un-timed architecture specification and the RTL implementation by inserting suitable design steps in between. These are based on architectural models having an abstraction level higher than RTL. Their modeling effort as well as simulation performance is multiple orders of magnitude superior. This allows real design space exploration since iteration cycles are very efficient, compared to the expensive redesign costs and turnaround cycles on RT-Level.

For both domains, the system modules themselves as well as the communication between these modules, efficient abstract modeling techniques have been introduced. In table Fig. 1, these abstraction levels are classified.

communication accuracy	data accuracy	timing accuracy	software accuracy
TLM	packets	un-timed	functional specification
		timed	performance model
	bytes/ words	transaction	instr. accurate ISS
		transfer	cycle accurate ISS
RTL	bit-vectors	cycle	HDL processor model

Fig. 1. Modeling Abstraction Levels

For the inter-module communication, all levels higher than RTL are covered by the new Transaction-Level-Modeling (TLM) paradigm [1]. They have in common that the pin wiggling on the interfaces is not modeled in detail any more, but is replaced by condensed Interface Method Calls (IMC). A distinction within this large TLM abstraction level can be done with respect to the accuracy of data and timing. *Bytes/words accurate* TLM can achieve a significant performance gain against RTL without sacrificing cycle accuracy when modeling at *transfer* timing level. *Packet level* TLM, in contrast, obtains further large gains in simulation efficiency especially by modeling a whole burst of data transfers as one single communication event.

Also the system modules can be modeled on a variety of abstraction levels. Due to rising flexibility constraints, silicon area is more and more filled by programmable SoC modules, especially by the relatively power efficient Application Specific Instruction set Processors (ASIPs). Also software running on a processor can be modeled more efficiently above RTL. This can either be done applying an Instruction Set Simulator (ISS), which actually can simulate execution of the final application software, or it can be modeled with an abstract module which just uses annotated execution delay estimations or does not contain timing at all.

It can be observed that the communication accuracy on the *timed packet* TLM abstraction level is very good [2]. But the *performance model* for the abstract processor modules normally applied on this level cannot deliver reliable information. CPU load, impact of the RTOS, SW response time and the on-chip communication traffic shape as caused by the instruction caches can hardly be analyzed without an ISS.

Thus, we present the integration of fast instruction accurate ISSs already into a *packet level* system simulation framework to combine its advantages of high simulation speed and efficient design exploration capabilities with the accuracy of an ISS.

The following section presents related work in system level design. In section 3, the system exploration framework this work is based on is summarized. Section 4 then more detailed motivates the need for the early ISS integration. The advantages of the ISS availability for meaningful design space exploration is then presented in section 5. Next, a dual-processor case study is presented in section 6. Finally, section 7 concludes this work.

2 Related Work

Heavy research on all abstraction levels between functional specification and RTL is currently done to cope with the ever increasing SoC complexity problem. The research on the packet level TLM abstraction level is addressing efficient and fast design space exploration for huge SoCs. Various system level design frameworks are proposed addressing design space exploration for partitioning and mapping [3, 4] as well as conceptualization and performance analysis [5]. This work is based on [5] because of its high flexibility and modularity. Basically, any of these system level design frameworks could be extended with retargeteable ISSs.

Traditionally, mixed HW/SW systems are not integrated and co-simulated before the RTL abstraction level is reached [6]. But the low simulation speed and the time consuming modeling on this detailed level prohibits architectural exploration and embedded software development.

Thus it is desirable to have accurate software models already on higher abstraction levels. Recently, efforts have been done to integrate Instruction Set Simulators already on bytes/words accurate TLM level [7–9]. In this paper, fast instruction accurate processor simulators have even been coupled into a packet level simulation framework to very early take advantage of the high ISS accuracy.

By integrating ISSs automatically generated by the retargeteable LISA processor design platform [10], this approach is very generic. It allows the system designer either to integrate processors purchased or reused as IP, alternatively he also could conceptualize an own optimally tailored ASIP for the respective task. In principle, another retargeteable processor exploration framework like ISDL [11], EXPRESSION [12], nML [13], MIMOLA [14] could also have been chosen. We selected the LISA [10] platform because of its powerful simulators as well as due to its support for C compiler generation [15] and for full RTL generation.

3 NoC exploration framework

An important design challenge in the exploration process is to find an optimal mapping of processing elements to a complex communication architecture. This disciplined exploration of different communication architectures has recently been subsumed under the Network-on-Chip (NoC) design paradigm.

The NoC exploration framework applied here [5] is implemented on top of the SystemC library [16]. The application's functionality is captured by system modules, which communicate to each other by exchanging data packets (Fig. 2). Master modules like processor cores actively initiate transactions, while slave modules, e.g. memories, can only react passively. All communication between these functional modules is encapsulated by the NoC channel module. The actual communication topology can be easily configured by selecting the respective network engine and parameterizing it accordingly (e.g., for AMBA AHB or STBus). These network engines determine the latency and throughput during simulation according to the selected network type (point-to-point, bus, crossbar, etc).

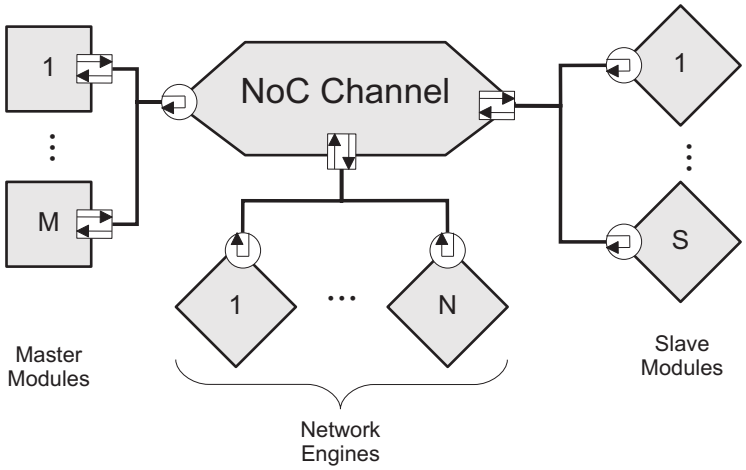

Fig. 2. NoC Channel Overview

4 Early ISS Integration

In the system level NoC exploration framework, data exchange is modeled on the high abstraction level of packet transfers. This temporal and data abstraction is one of the key techniques to enable the high system level simulation performance. As shown in several case studies [17, 5], it is beneficial to model a burst transfer as one single event, neglecting ever-recurring low-level implementation details. This is straight forward for the data communication as occurring in networking applications. Here, one data burst contains functionally associated data, e.g. the Route Lookup (RLU) information for an IP packet. In case of applications with higher data locality than networking, data caches can drastically reduce access latencies, and they introduce a packet oriented but totally different traffic characteristic on the NoC infrastructure. Thus, modeling the impact of the data caches already leads to a significant accuracy enhancement. For this, the abstract modules have to be equipped with data cache models; they do not need to be replaced completely yet for this issue.

But in the final SoC design, processor cores mostly execute the master module functionality to meet flexibility constraints. Especially in case of complex, MMU and cache equipped processors like the MIPS32 core, the program code also needs to be transferred over the NoC if (and only if) a program cache miss occurs. The cache behavior highly affects the performance of the processor, and it can have high influence on the network traffic and thus on the performance of the remaining modules. Even more, the impact of the processor architecture itself, e.g. if it is a VLIW processor or an optimally tailored ASIP, and the impact of the RTOS on the software execution time can hardly be estimated on the very high abstraction level. These issues make it nearly impossible to annotate accurate timing budgets to the master modules.

Thus, we propose very early ISS integration into the system level NoC platform. After the system architect has excluded lots of architectural alternatives, he needs a more

detailed model for further design decisions. For example, is it sufficient to integrate a Common-Of-The-Shelf (COTS) processor or is it necessary to optimally tailor an own ASIP. When replacing the abstract functional module with a processor simulator, one can exploit the accurate NoC framework capabilities efficiently especially for cached data transfers. A data packet then has the granularity of a cache line.

Recent developments in Design Automation more and more ease the task of migrating from an abstract master module model to software running on an ISS. When using a processor design environment like LISA [10], an initial customized instruction accurate processor model is available within a few man weeks. Applying the LISATek Processor Designer tools [18], a processor simulator, an assembler, a linker, and with some amount of additional manual specification even a C compiler can be generated automatically for the specified architecture [15]. This allows compiling (or recoding) the application code for the target architecture already now, which enables profiling and verifying the final software very early.

Alternatively, already existing LISA processor models could be integrated easily into the NoC framework to explore on this high abstraction level if the efficiency and computational power of these purchased or own IP blocks is suitable for the respective task.

So far, integrating LISA processors into their system context was earliest possible when a *bytes/words accurate* TLM model of the bus is available [9]. The technique presented in this paper allows coupling ISS directly into the abstract NoC simulation framework. All profiling, statistics and debugging capabilities provided by the NoC exploration framework is used for analyzing the cache traffic characteristics and optimizing the overall system architecture.

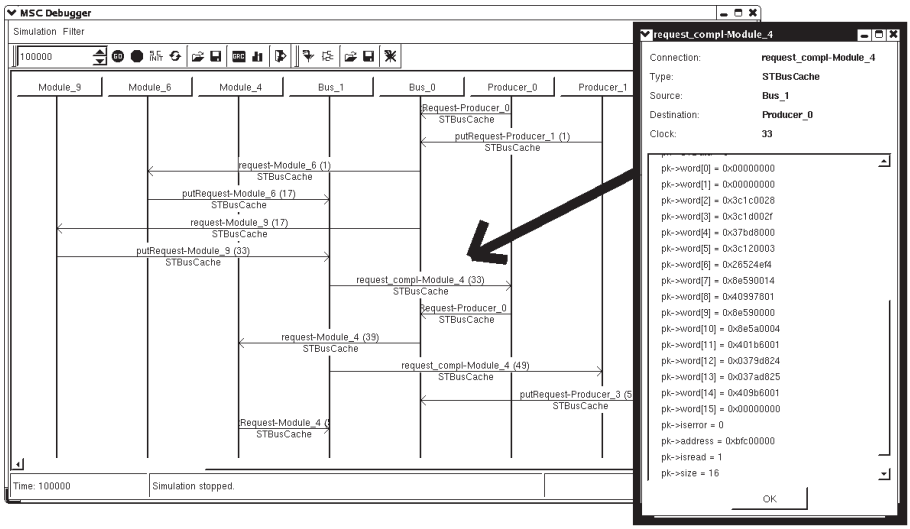

Fig. 3. NoC as observed by the MSC debugger

5 NoC Design Space Exploration

Having integrated the LISA processor models into the NoC simulation framework, its design exploration capabilities can again be used to also deeply analyze the processor related issues in system context. This section illustrates the debugging and profiling capabilities of the NoC simulation framework.

5.1 Debugging

In Fig. 3, a dual processor STBus system is observed in a Message Sequence Chart (MSC) diagram. Every system module is displayed as a vertical line, and every packet transfer appears as an horizontal arrow with the time stamp annotated in brackets. In the simulation shown here, the program caches of both processors (Producer_0 and Producer_1), of course, have a cache miss at system startup. The memory requests are put over a shared symmetric STBus model (Bus_0 for requests and Bus_1 for responses) to the respective memory modules (Module_x). The contents of every packet can be displayed by clicking onto the corresponding communication arrow. In Fig. 3, the first cache line returned to Producer_0 is shown.

Powerful filtering mechanisms in the MSC debugger together with the LISATek multiprocessor debugger front-end [19] assist the system architect in detecting bugs and deadlocks as soon as possible.

5.2 Profiling

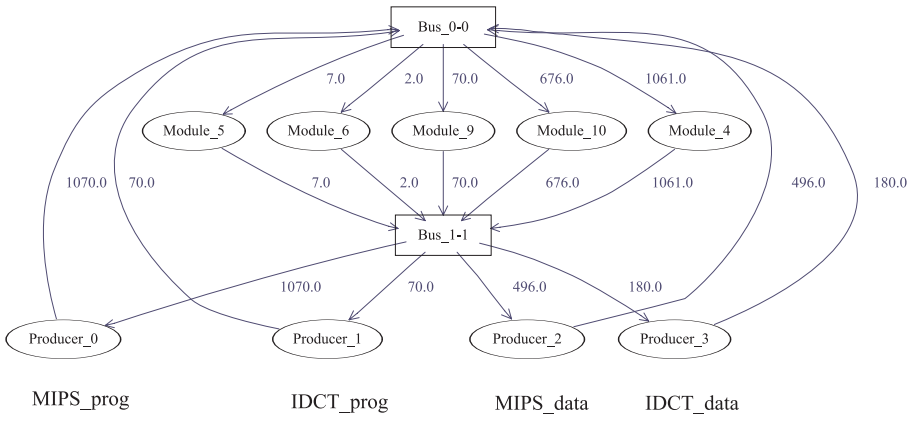

Fig. 4. Traffic graph generated by the NoC environment

As soon as the debugging has succeeded and the timed MP-SoC model is working correctly, the system designer now can move forward and profile the communication behavior of the current platform. In Figure 4, an automatically generated traffic graph

for the same dual processor system is shown. Accumulated over the full duration of the system simulation, the number of packet transfers between the system modules is displayed as weighted arrows.

There are several other visualizations available, for example buffer fill state histograms, to further assist the designer in detecting bottlenecks. Together with the processor profiling capabilities of the LISATek tools, the user gets valuable hints to improve the overall system performance early in the design flow.

6 Dual-Processor JPEG Case Study

The outlined early ISS integration has been applied on a dual-processor system optimized for decoding JPEG images. This section first describes the target application, as well as the development of the final SoC topology. Then, the simulation accuracy of the NoC framework is compared to the cycle accurate TLM reference.

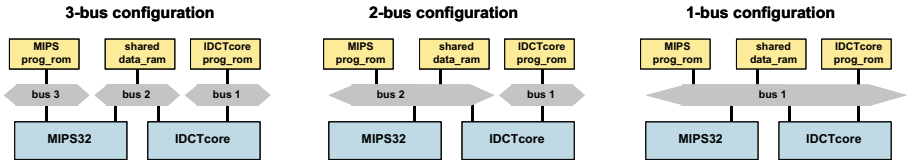

Fig. 5. SoC bus topologies

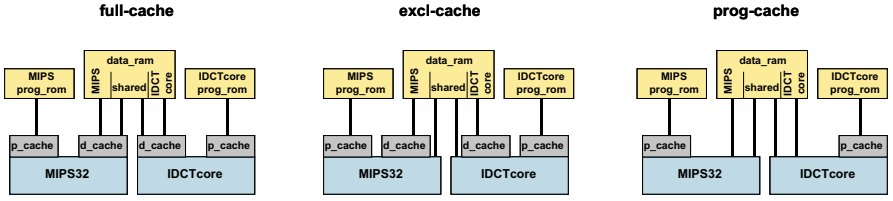

Fig. 6. SoC cache topologies

6.1 JPEG decryption system

The considered application is a JPEG-2000 decompression algorithm decoding a 150x100 sized bitmap. Both the C original source code of the algorithm and the sample bitmap are freely available from the JPEG group's web page [20]. We modified the code to run it jointly on two processor cores: First, a specially designed IDCTcore coprocessor for calculating the inverse discrete cosines transformation (idct), which is the algorithmic

core of JPEG decryption. Second, a MIPS32 processor core performs the remaining, i.e., the control and file I/O tasks. Both processors share a common data memory, and each of them accesses a personal program memory.

The bus topologies and cache topologies that have been explored are depicted in Fig. 5 and Fig. 6, respectively.

6.2 Simulation Accuracy

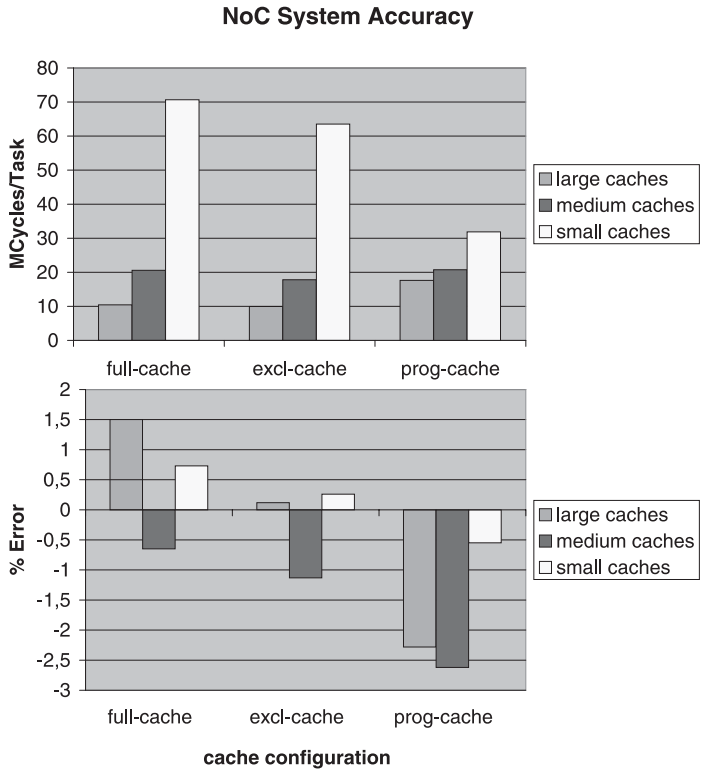

Fig. 7. Simulation Accuracy (1-bus topology)

In Fig. 7, for different cache configurations the number of cycles consumed to decode the given bitmap is shown. In the *full-cache* system configuration, cache consistency is ensured easily by flushing and invalidating the data cache after a processor's task is performed. The *excl-cache* configuration avoids this problem by only caching memory regions exclusive to one processor. The synchronization of the two processors is interrupt-based.

The *small caches* contain just one line (16 words), thus the cache miss probability is very high, causing high traffic on the bus. Even in this case, it turned out that the

overhead of a 2-bus and 3-bus topology does not obtain a significant performance gain compared to the 1-bus configuration. This is obviously the case because the processor caches are located in front of the system bus(es).

As depicted in Fig. 7 for the 1-bus topology, the cache size in contrast has a high impact on the overall system performance. The difference between *small caches* and *large caches* is up to a factor of 7, the program cache size impact alone still leads to a factor of two. These factors would even be higher for slower memories behind the AMBA bus. The program cache size is an example for a property which is nearly impossible to annotate to an abstract processor module.

We compared the cycle counts against a reference system with the same ISSs, but connected to CoWare TLM AMBA AHB bus models [21]. In any of the system configurations, the estimation error introduced by the packet level NoC bus models is lower than 3 %. This is due to the fact that the currently deployed generic bus model of the NoC simulation framework does not yet take all AMBA AHB pipeline effects into account.

7 Conclusion

Integrating LISA processors into abstract NoC exploration frameworks combines the high accuracy of ISSs for the processor modules with the flexibility and simulation speed of the NoC simulation framework for the communication modeling. A seamless design flow is obtained by smoothly stepping forward from full abstract simulation to the cycle true TLM world, initially reusing the NoC communication models.

References

1. T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
2. Ariyamparambath, M. and Bussagila, D. and Reinkemeier, B. et al. A Highly Efficient Modeling Style for Heterogeneous Bus Architectures. In *International Symposium on System-on-Chip*, Tampere (Finland), Nov 2003.
3. Vladimir D. Zivkovic, Ed Deprettere, Erwin de Kock, Pieter v. d. Wolf. Fast and Accurate Multiprocessor Architecture Exploration with Symbolic Programs. In *Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2003.
4. D. Gajski, J. Zhu, R. Dömer, A.Gerstlauer, S. Zhao. SpecC: Specification Language and Methodology. In *Kluwer Academic Publishers*, 2000.
5. T. Kogel, M. Doerper, A. Wieferink et al. A Modular Simulation Framework for Architectural Exploration of On-Chip Interconnection Networks. In *The First IEEE/ACM/IFIP International Conference on HW/SW Codesign and System Synthesis*, Newport Beach (California USA), Oct 2003.
6. J. Notbauer, T. Albrecht, G. Niedrist,S. Rohringer. Verification and management of a multimillion-gate embedded core design. In *DAC*, 1999.
7. P. Paulin P. Magarshack. System-on-chip beyond the nanometer wall. In *DAC*, 2003.
8. W. Cesario, A. Baghdadi, L. Gauthier et al. Component-Based Design Approach for Multicore SoCs. In *DAC*, 2002.

9. A. Wieferink, T. Kogel, G. Braun et al. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, Paris, France, Feb 2004.
10. A. Hoffmann, T. Kogel, A. Nohl et al. A Novel Methodology for the Design of Application Specific Instruction-Set Processor Using a Machine Description Language. In *IEEE Transactions on Computer-Aided Design of Integrated Cicuits and Systems (TCAD) vol. 20 no. 11*, pages 1338–1354. IEEE, Nov 2001.
11. G. Hadjiyiannis, S. Devadas. Techniques for Accurate Performance Evaluation in Architecture Exploration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2003.
12. P. Mishra, P. Grun, N. Dutt, and A. Nicolau. Processor-memory co-exploration driven by an architectural description language. In *Intl. Conf. on VLSI Design*, 2001.
13. A. Fauth and J. Van Praet and M. Freericks. Describing Instruction Set Processors Using nML. In *Proceedings of the European Design & Test Conference*, Mar. 1995.
14. R. Leupers. HDL-based Modeling of Embedded Processor Behavior for Retargetable Compilation. In *Proc. Int. Symp. on System Synthesis*, Sep. 1998.
15. M. Hohenauer, H. Scharwaechter, K. Karuri et al. A Methodology and Tool Suite for C Compiler Generation from ADL Processor Models. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, Paris, France, Feb 2004.
16. SystemC initiative. *http://www.systemc.org*.
17. T. Kogel, A. Wieferink, R. Leupers et al. Virtual Architecture Mapping: A SystemC based Methodology for Architectural Exploration of System-on-Chip Designs. In *Int.Workshop on Systems, Architecturs, Modeling and Simulation (SAMOS)*, Samos (Greece), July 2003.
18. LISATek Product Line. *CoWare, http://www.coware.com*.
19. Wieferink, A. and Kogel, T. and Nohl, A. et al. A Generic Toolset for SoC Multiprocessor Debugging and Synchronisation. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, The Hague (Netherlands), June 2003.
20. Official JPEG homepage. *http://www.jpeg.org*.
21. ConvergenSC. *CoWare, http://www.coware.com*.