A Prototyping and Evaluation Framework for Research on Timing-analysable Memory Hierarchies for Embedded Multicore SoCs

Florian Haas, Sebastian Altmeyer

University of Augsburg, Germany

Abstract

Research on memory hierarchies regarding the non-functional requirements in embedded multicore systems demands for a framework to support the prototyping and evaluation of new methods. In current multicore processors, accesses on shared resources by arbitrary tasks lead to interferences, which can result in timing violations of high-priority tasks. However, incorporating all potential interferences in the schedulability analysis leads to an enormous overestimation of the task execution times, and requires a full analysis of all tasks running on the system. Enhancements in the memory hierarchy can provide isolation to restrict potential interferences, thus improving the worst-case performance. To research on modifications in the memory hierarchy of a multicore processor, a prototyping and evaluation framework is required. This paper describes the design of such a framework, and outlines the individual parts and their interconnections.

Keywords

parallel real-time system, memory hierarchy, FPGA prototyping framework

1. Introduction

The performance of multicore processors is strongly desired in various domains of embedded systems to satisfy the increasing demand for computational power. Complex algorithms and software systems, e. g. in autonomous driving, benefit from high-performance general-purpose shared-memory multicores. However, these processors do not meet the typical requirements on real-time and safety, and thus cannot be used without performance-degrading and laborious software mechanisms. Elaborate methods in such systems have been developed to further improve the average-case performance of the processor, for example the increasing depth of the memory hierarchy. These and the shared resources, like last-level caches, buses, and main memory, result in the ultimate challenge of calculating tight WCET bounds for the tasks in a time-critical system.

The crucial problem is the missing guaranteed freedom of interferences between tasks that run on separate cores. Thus, an arbitrary low-priority task is able to influence the timing behaviour

CERCIRAS WS01: 1st Workshop on Connecting Education and Research Communities for an Innovative Resource Aware Society

haas@es-augsburg.de (F. Haas); altmeyer@es-augsburg.de (S. Altmeyer)

https://es-augsburg.de/haas (F. Haas); https://es-augsburg.de/altmeyer (S. Altmeyer)

0000-0002-4714-2562 (F. Haas); 0000-0002-2487-7144 (S. Altmeyer)

of another, potentially high-priority task on a different core. This can happen through accesses on shared resources, for example shared caches or the main memory [1]. As a consequence, a schedulability analysis of the overall system with only minimal overestimation becomes nearly impossible for more than a few cores and deeper memory hierarchies.

The general objective of research on this topic is to facilitate predictable performance, with minimal over-estimation of timing bounds, by reducing the sources of potential interferences on shared resources. Existing software-based approaches, e. g. performance counter monitors [2], or program modification during compilation, are limited, as they can either only detect excessing interferences, or are required to be applied to all tasks of the system. Thus, hardware mechanisms promise a better lever to control the behaviour of any task on the system. However, to research hardware-implemented methods, a proper evaluation platform is required. For example, a hardware implementation of a memory bandwidth reservation mechanism like MemGuard [3] could be evaluated and compared with other approaches. To research potential improvements on shared resource accesses under timing constraints, a realistic model of a typical memory hierarchy is needed in the first place. Microarchitecture simulators with multicore configurations exist, but their processor-centric design does not support for a prototype implementation and a realistic evaluation. Further, the evaluation system needs to be capable of executing realistic benchmarks, for prototyping different ideas, as well as for a thorough evaluation of their impact on the performance.

Previous work focused mostly on fault tolerance of parallel systems, but the research always involved shared-memory systems. Different systems have been used to evaluate the proposed methods, from software-only approaches on typical desktop and server hardware, over the Gem5 simulator, to FPGA prototypes. As a side effect of the conducted implementations and evaluations, some experience with diverse platforms has been collected. The work on a software-only fault tolerance mechanism [4, 5] showed the numerous restrictions of an unmodifiable hardware implementation. To overcome these limitations, later research was undertaken on the Gem5 microarchitecture simulator, where a customized hardware transactional memory was built into the memory hierarchy [6, 7]. However, since the simulator focuses on the detailed simulation of the processor cores itself, it provides only a rather functional memory hierarchy with limited timing accuracy. Switching to an FPGA prototype with multiple MicroBlaze softcores [8] showed the difficulties of integrating hardware and software parts with non-open processor cores. Overall, these experiences affirm the demand for an open system to prototype and evaluate memory hierarchies for future research ideas.

This paper describes a prototyping and evaluation framework for embedded multicore systems, and outlines the assembly of the individual parts into a synthesizable design for both simulation and prototyping on an FPGA. The framework is based on ChipYard, which supports design and evaluation of full-system hardware, using the Rocket Chip generator and its in-order RISC-V CPUs. The main benefit of ChipYard is the configurability and customizability of the involved modules. The interconnects could also be replaced with a NoC to research on manycore systems, or a combination of both with shared-memory clusters connected through a NoC. Based on the proposed framework, the research on elements of the memory hierarchy will be facilitated to improve the applicability of multicore processors in embedded systems.

2. Requirements for Research on Timing Predictable Shared-memory Multicore Systems

To approach the objective of calculating tight WCET bounds for time-sensitive tasks in shared-memory multicore systems, the potential interferences on shared resources have to be identified and measured first. While such evaluations can be performed on existing hardware, potential new methods to prevent or restrict interferences require customisable hardware components.

A system that enables the modification and enhancement of individual elements in the memory hierarchy should fulfil the following requirements:

- Customisable hardware to extend or modify elements of the memory hierarchy
- Measurement of the overall performance and counting individual accesses on shared resources
- Independence of CPU architectures
- Scalable number of processor cores
- Hardware cost estimation of extensions and customisations
- Fast response on functional correctness of the implementation
- Fast and approximate evaluation of the simulated model
- Accurate full-system evaluation on an FPGA

These requirements are satisfied by our proposed framework, for which the Chipyard project provides a promising foundation. It is the predestined choice, since it is built around the open RISC-V ecosystem, and allows to customize or replace individual elements of the memory hierarchy. It further supports simulation and FPGA synthesis based on the same and identical code.

3. Overview of the Framework

The evaluation framework builds upon existing open-source projects that have been developed in recent years around the prevalent RISC-V architecture.

3.1. Chipyard

Chipyard [9] simplifies the process of designing full-system hardware by integrating all necessary parts from CPU cores to supplementing logic to connect the devices of an FPGA evaluation board. Fig. 1 illustrates the individual parts of Chipyard: Processor cores can be created for example with the Rocket Chip Generator, which generates configurable and customizable cores that implement the RISC-V instruction set, either in-order Rocket cores, or the more complex and powerful out-of-order BOOM cores. Beside the L1 caches provided by the Rocket Chip Generator, secondary level caches and different kinds of interconnecting buses can be generated. There is also code provided to connect to and communicate with peripheral devices like UART and JTAG.

The generated Verilog code can be further compiled with Verilator for a simulation of the overall system, or with FireSim, which additionally allows to simulate DDR3 main memory.

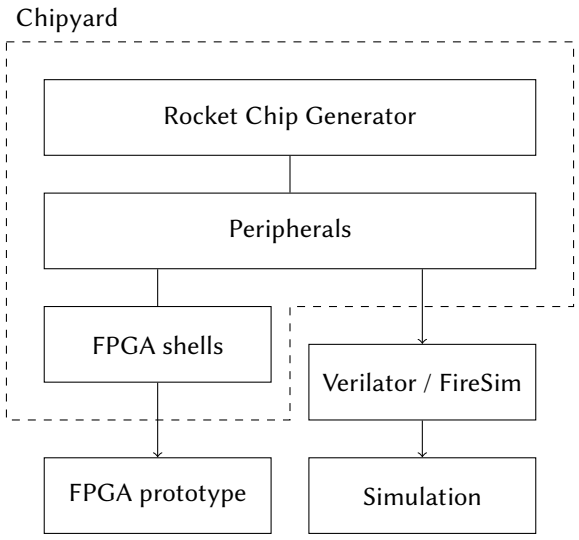

Figure 1: Overview of generators of Chipyard [9]. The resulting code can be synthesised for an FPGA or simulated.

Alternatively, individual FPGA *shells* wrap the code with a harness to connect the units of the SoC to I/O pins of a concrete FPGA, to build a prototype running on an FPGA evaluation board. Such prototype is able to communicate with the built-in peripheral devices like UART and JTAG, as well as the off-chip DDR memory.

3.2. Rocket Chip Generator

The Rocket Chip Generator [10] produces designs of a SoC with multiple processor cores, a memory hierarchy, and interconnects. Fig. 2 depicts a generated chip with four processor *tiles*, consisting of an in-order Rocket RISC-V core and L1 instruction and data caches, L2 cache banks with the memory bus, and additional buses for peripheral devices, DMA devices, and control units like the boot ROM and interrupt controllers. All processor tiles and all individual buses are connected through a shared system bus, which is typically implemented as a crossbar, but can also be configured as a ring bus.

3.3. Memory Hierarchy Evaluation Framework

A common objective of research on memory hierarchies for real-time systems is to reduce interferences on shared resources. From this, the main elements of the system under evaluation are derived: All units that control access to shared resources, like the peripheral bus, or the L2 cache, are of interest, as well as the private L1 caches that are connected to the shared system bus. In Fig. 3, these elements are shown below the processor cores, which are not of special interest for interference analysis. All accesses to shared resources that originate in the cores have to pass through the L1 instruction or data caches, which can control the communication.

The prototyping flow from implementing a design of one or more specific parts of the memory hierarchy to code generation and simulation or evaluation is depicted in Fig. 4. Unit tests can

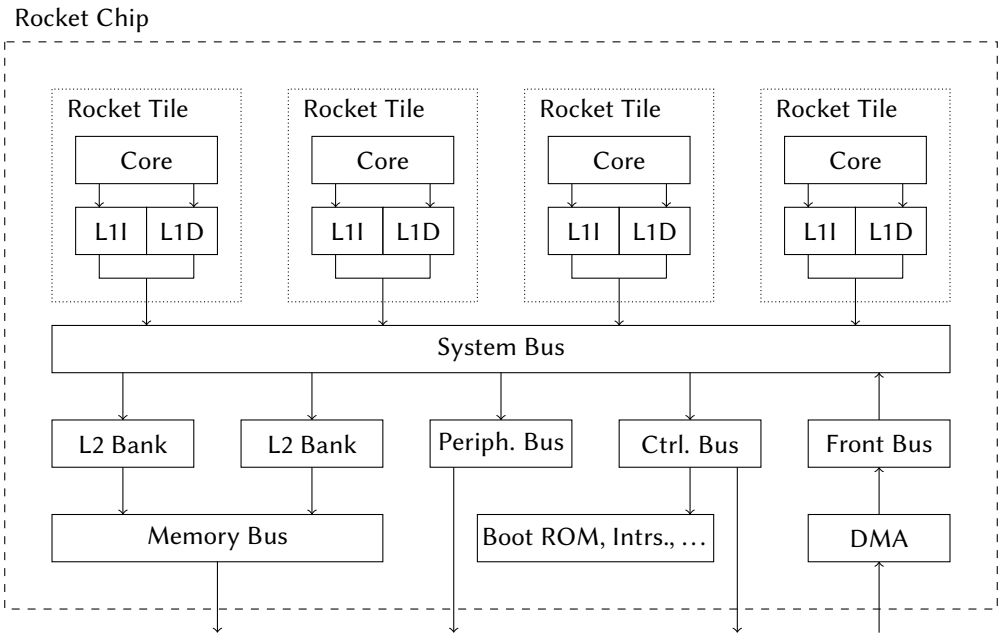

Figure 2: The Rocket Chip [10] consists of multiple processor tiles, and devices connected through dedicated buses, like memory and peripherals. All parts of the chip communicate through the system bus.

provide fast checks of the functional correctness of the implemented or modified mechanisms. After passing these tests, Verilog code is generated, which can be simulated with Verilator to test the design with a set of benchmarks. The simulation provides fast feedback on the behaviour of the system, to compare different potential implementations before running the full evaluation of the synthesised bitstream on the FPGA. The evaluation of the design on the FPGA provides accurate timing measurements of the individual tasks, and a trace log of accesses on shared resources. These results allow to quantify the improvements of the implemented memory hierarchy modifications, and enable the detection of timing violations or forbidden

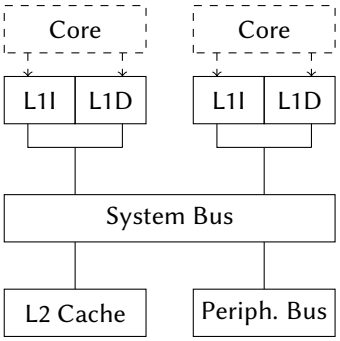

Figure 3: Elements of interest to evaluate interferences in the memory hierarchy: private L1 caches, shared L2 caches, buses that connect shared resources, and the shared system bus itself.

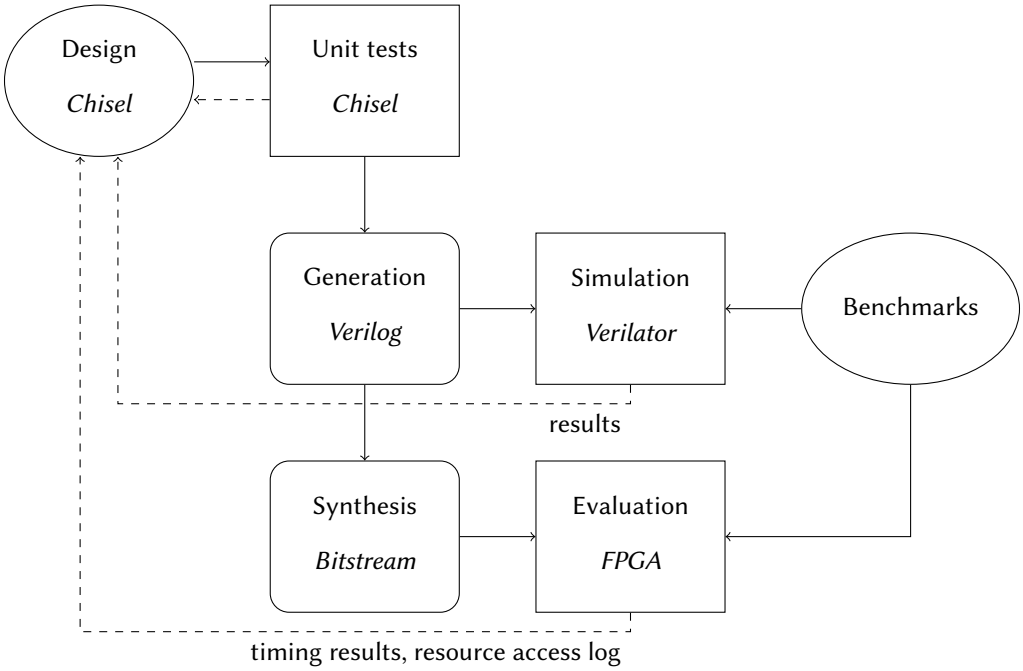

Figure 4: Overview of the prototyping flow with the evaluation framework. Results of the unit tests provide immediate feedback, which can be further tested in the Verilator simulation. The evaluation of the bitstream on the FPGA provides accurate timing results and logs of the resource accesses.

interferences that should not occur.

The possibility to connect a debugger to the simulation, as well as to the system on the FPGA, facilitates the detection of implementation faults, and provides detailed insight into the behaviour of the system under specific circumstances when needed.

With the feedback loop between the design and the simulation, available computational capabilities can be leveraged to compare numerous different design variations, to select a few designs of interest for the full evaluation of the FPGA.

4. Conclusion & Future Work

This paper described the design of a prototyping and evaluation framework to research on memory hierarchies, for getting closer to the overall objective of enabling high-performance multicore processors in embedded real-time systems. The framework is built upon existing open-source projects around the RISC-V architecture, connecting the different tools together. It integrates all the required steps to automatically generate the Verilog code, compile and run the simulation, to synthesise the bitstream and program the FPGA with it, and to run the evaluation.

The next step is to implement the basic tool chain for automatic unit tests, code generation, simulation, and synthesis. Afterwards, measurement facilities in the individual components of the memory hierarchy have to be added to evaluate the behaviour of the system under parallel workloads. Such workloads first have to be identified based on use-cases from different

industries, and reconstructed by a set of different benchmarks.

Based upon the proposed framework, research on new approaches for controlling interferences on shared resources within shared-memory multicores can take off.

Acknowledgments

This work is partially supported by the CERCIRAS COST Action no. CA19135 funded by COST.

References

[1] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, R. I. Davis, A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems, ACM Comput. Surv. 52 (2019). doi:10.1145/3323212.

[2] J. Freitag, S. Uhrig, T. Ungerer, Virtual timing isolation for mixed-criticality systems, in: Euromicro Conference on Real-Time Systems (ECRTS), 2018, pp. 13:1–13:23. doi:10.4230/LIPIcs.ECRTS.2018.13.

[3] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, L. Sha, MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms, in: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 55–64. doi:10.1109/RTAS.2013.6531079.

[4] F. Haas, S. Weis, T. Ungerer, G. Pokam, Y. Wu, Fault-Tolerant Execution on COTS Multi-core Processors with Hardware Transactional Memory Support, in: Architecture of Computing Systems (ARCS), 2017, pp. 16–30. doi:10.1007/978-3-319-54999-6_2.

[5] F. Haas, Fault-tolerant Execution of Parallel Applications on x86 Multi-core Processors with Hardware Transactional Memory, Phd thesis, Universität Augsburg, 2019. URL: https://opus.bibliothek.uni-augsburg.de/opus4/59566.

[6] R. Amslinger, S. Weis, C. Piatka, F. Haas, T. Ungerer, Redundant Execution on Heterogeneous Multi-cores Utilizing Transactional Memory, in: Architecture of Computing Systems (ARCS), 2018, pp. 155–167. doi:10.1007/978-3-319-77610-1_12.

[7] C. Piatka, R. Amslinger, F. Haas, S. Weis, S. Altmeyer, T. Ungerer, Investigating transactional memory for high performance embedded systems, in: Architecture of Computing Systems (ARCS), 2020, pp. 97–108. doi:10.1007/978-3-030-52794-5_8.

[8] R. Amslinger, C. Piatka, F. Haas, S. Weis, T. Ungerer, S. Altmeyer, Hardware multiversioning for fail-operational multithreaded applications, in: International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2020, pp. 20–27. doi:10.1109/SBAC-PAD49847.2020.00014.

[9] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, B. Nikolić, Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs, IEEE Micro 40 (2020) 10–21. doi:10.1109/MM.2020.2996616.

[10] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, et al., The Rocket Chip Generator, Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, 2016.