

Designing Predictable Multicore Architectures for Avionics and Automotive Systems

— extended abstract —

Reinhard Wilhelm, Christian Ferdinand, Christoph Cullmann, Daniel Grund,
Jan Reineke, Benoît Triquet *

1 Introduction

This paper deals with architectures and their design principles for embedded control systems as they are used in the automotive and aeronautics industries.

Growing software complexity in the embedded domain has led to the development of standardized frameworks which focus on composing components, possibly developed by different suppliers, on Electronic Control Units (ECUs). Examples are AUTOSAR in the automotive domain and the IMA architecture in the aeronautics industry.

Both IMA and AUTOSAR are claimed to support compositionality and composability; the behavior of a system is determined by the behavior of the system's components and the type of composition (*compositionality*), and the behavior of individual components should not change by the composition (*composability*). This is the assumption on which the *incremental acceptance* of IMA is based;

To facilitate composability, AUTOSAR abstracts from the underlying hardware, i.e. the actually deployed ECUs. For time-critical systems, composability of the timing behavior would mean that the modification of one component would only influence its timing behavior and not that of other components. This paper proposes a design philosophy supporting composability of the timing behavior. It is claimed that without such a design philosophy composability is hard to achieve without sacrificing too much performance.

The AUTOSAR timing model currently being developed mainly concerns the integration of scheduling requirements. The success of scheduling analysis depends on the predictability of the execution times of the AUTOSAR-“runnables”, the basic building blocks of a software component. When multiple software components are mapped to a hardware architecture where a high degree of interference between the components cannot be avoided (e.g. due to shared caches or buses) execution times of runnables may vary considerably endangering the possibility to predict safe and precise execution time bounds, which again

* R. Wilhelm, D. Grund, and J. Reineke are with the Department of Computer Science, Saarland University, Germany. C. Ferdinand and C. Cullmann are with AbsInt Angewandte Informatik GmbH, Germany. Benoit Triquet is with EADS Airbus, France. The research reported herein has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 216008 (PREDATOR). The findings of this paper represent no commitment from the side of Airbus as to their future architectures.

limits the success of scheduling analysis and counteracts the idea of composing software components. Thus, the applicability of the AUTOSAR idea depends on availability of architectures on which software composition does not lead to unpredictable timing behavior.

Meanwhile, it is generally accepted that the interference on shared resource is the main culprit for the variability of execution times and the resulting unpredictability.

2 Application Characteristics

Applications in our context are vehicle functions that are the granularity units for the mapping to computational units. An application may consist of several *tasks*. Engine control, electronic stability program, flight control and guidance would all be considered applications in this sense. The embedded applications already in use in the considered industrial domains or foreseen in future generations can be classified into two categories: Some realize complex control and computer vision systems requiring high-performance computer systems exploiting parallel algorithms on shared-memory architectures. Shared memory is essential to provide the necessary performance. We will call them *HP-applications* and the architectural subsystems to which they are mapped *HP-computational units (HP-CUs)*. Others realize simple control loops not requiring high performance. We will call them *LP-applications* and the computational units running them *LP-computational units (LP-CUs)*. Together, these CUs form the building blocks of the foreseen predictable execution platform. There is not much communication traffic between these CUs. Several characteristics of the control applications influence the design:

- The applications consist of a combination of control loops and some finite-state control.
- Each control loop is fully contained within one application.
- Applications to be mapped to the target architecture share little code.
- The finite-state control is partly shared between applications.
- The control loops cause transitions in the finite-state control, and the finite-state control sets parameters of the control loops.
- Reading from the shared global state happens at the beginning and writing to the shared global state happens at the end of each activation of a task.

System performance for many safety- and time-critical avionics applications is bounded by memory-system performance. Automotive suppliers currently execute the code of many applications from FLASH, which is the hardware component limiting overall performance. This means that most of the architectural complexity is badly invested.

Many of these systems have rigid timing constraints and need efficient methods for the derivation of timing guarantees. Exact worst-case execution times are impossible or very hard to determine, even for the restricted class of real-time programs with their usual coding rules. Therefore, these guarantees consist of

safe and precise upper bounds on the execution times of tasks. So, the combined requirements for the needed methods are

- *soundness*, to ensure the reliability of the guarantees,
- *efficiency*, to make them useful in industrial practice, and
- *precision of the results*, to increase the chance to prove the satisfaction of the timing requirements.

3 Reasons for Unpredictability

Current high-performance mono-processor architectures used for embedded control are designed to support some mix of assumed applications and are optimized for average-case performance. They use caches, speculation, and intra-processor parallelism, often dynamically scheduled, to overcome the CPU-memory bottleneck. A consequence of this design is a strong dependence of the execution times of instructions on the architectural execution state, a high variability of these execution times, and a low predictability of the timing behavior of programs [8]. Timing-analysis tools such as aiT of AbsInt have to search through a vast space of architectural states to determine safe upper bounds on the execution times of basic blocks in their control-flow contexts. The state space to be searched is spanned by two levels, i.e., control flow and architecture flow:

- *sequential control flow*: in general, different inputs result in different control flow. Static analysis therefore has to take all inputs and consequently all resulting control flow paths into account.
- *interleavings of concurrent control flow*: Many embedded systems consist of components with potentially concurrent execution. The resulting number of possible interleavings of accesses to shared resources, e.g. different sequences of accesses to a shared cache resulting from different threads executed on different cores may be quite large.
- *architecture flow*: Different initial and intermediate architectural execution states lead to different paths through the architectural finite state machine. In an out-of-order processor, these paths are dynamically scheduled so that again a large number of paths needs to be analyzed. The number, the size, and the mutual interdependence of architectural components influencing the timing behavior determines a large state space.

In the case of multi-core architectures, this means that caches shared between several cores with loosely synchronized threads will increase the number of interleavings such that sound timing analysis will become infeasible.

4 Architecture follows Application

The PROMPT (PRedictability Of Multi-Processor Timing) architecture design principles, see [11], aim at embedded hard real-time systems in the automotive

and the aeronautics industry requiring *efficiently predictable good worst-case performance*.

The small amount of sharing existing in the set of applications allows to design a target architecture with little interference on shared resources and thus little variance of execution times and high predictability. Our principle is *Architecture follows Application*. The goals of this design discipline is to *improve the worst-case performance* and to *make the derivation of reliable and precise timing guarantees efficiently feasible*. This design discipline will support the IMA and AUTOSAR movements in the aeronautics and the automotive industries. We conjecture that without this or a similar design discipline the required modular development process will not be realizable without an unacceptable loss of performance.

We expect that the improved precision of the execution-time bounds will

- avoid the need of over-commissioning and will therefore save resources, and at the same time
- allow for efficient timing-analysis methods.

5 Non-Interference

The PROMPT principles are

- the simplification of components of the architecture for better predictability wherever the advanced features are not needed for performance, e.g. greatly simplifying the pipeline architecture in systems whose performance is bounded by the performance of the memory system, anyway,
- the systematic elimination of interference on shared resources wherever they are not needed for cost or space reasons, e.g. avoiding shared caches or memories,
- the possibility to map a set of applications to the target architecture without introducing sharing not existing between the applications.

6 The Design Process

The architecture is designed in a multi-phase process. It starts with the design or the selection of the cores. Criteria for predictable cores have been described in detail in [11]. In short, the recommendations were:

- predictable caches [6], i.e., having an LRU replacement strategy, and/or scratchpad memories,
- fully compositional pipelines [11] having constant penalties for all timing accidents,
- no buses with several masters.

This article is concerned with the design above the cores. It actually assumes that the cores have already been developed with optimal predictability.

The architecture design starts from a set of applications with some known characteristics:

1. *Hierarchical privatization* will decompose the set of applications according to their sharing characteristics on the shared global state. The resulting partitioning of the set of applications could be used to define an isomorphically structured target architecture with no more shared resources than required by the set of applications. This architecture would offer high predictability. Note that there may well be HP-CUs in the hierarchical architecture. However, they will probably share little code and data with other nodes. Thus, mapping code to shared memory seems to be a bad idea as argued in Section 2. Therefore, we advocate separate memory hierarchies for non-shared code and even suggest replication for shared code.

2. *Sharing of lonely resources* would introduce sharing of costly and infrequently accessed resources. Input/output devices will most likely have to be shared, for cost and space reasons. Typical examples of high-performance devices are FlexRay ports in automotive or AFDX ports in avionics systems. They are used for communication between the different embedded systems. FlexRay only allows a bandwidth below 10 MBit/s and even the AFDX based on 100 MBit Ethernet allows only a fraction of this bandwidth to be utilized. So, they can be considered typical representatives of *lonely resources*, i.e., resources accessed during only a very small fraction of the available bandwidth of the accessing processor.

Lonely resources can be shared without much loss of predictability as any over-estimation will be scaled down by this low fraction of exploited bandwidth. Compared to the overall used memory bandwidth of a typical automotive or avionics application, the input/output devices make up only a small percentage. Even very conservative approaches to bound the interference costs on these devices won't lead to significant loss of precision for the bounds of the overall runtime.

3. *Controlled socialization* would try to satisfy cost constraints with an acceptable loss of predictability. It would introduce sharing while controlling the loss in predictability.

The main problem is to determine safe and sufficiently small delays for the access to shared resources. There seem to exist several alternatives for this. [7] compute deterministic access protocols for the accesses to shared resources from the access patterns to these shared resources. This deterministic protocol will allow to control the size of the access delay and to derive still safe and precise bounds on the overall execution times. [5] perform “cumulative” analyses using upper bounds for resource consumption to determine safely bounded access delays. [4] extends the concept of compositional architectures with static bounds in the sense of [11] to multi-core architectures. They propose a hierarchical arbitration scheme to statically bound the delays for the accesses to shared resources.

7 Related Work

The design of embedded systems with predictable timing behavior is gaining increased attention. However, there is little agreement even about the notion of predictability. The first formally defined notion and associated theory of predictability, fully compatible with existing observations [3], was developed for cache architectures [6].

For space reasons, we only want to mention two approaches aiming at an overall architectural approach. The PRET architecture developed by Lee and Edwards [1] aims at repeatable timing behavior by fixing instruction execution times and balancing program paths. The CoMPSoC-approach offers templates for the composition of predictable stream-processing architectures [2].

Descriptions of sound and precise methods for timing analysis can be found in [9, 10].

References

1. S. A. Edwards and E. A. Lee. The case for the precision timed (PRET) machine. In *DAC*, pages 264–265. IEEE, 2007.
2. A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. Comsoc: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):1–24, 2009.
3. R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *IEEE Proceedings on Real-Time Systems*, 91(7):1038–1054, 2003.
4. M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for wcet analysis of hard real-time multicore systems. In S. W. Keckler and L. A. Barroso, editors, *ISCA*, pages 57–68. ACM, 2009.
5. R. Pellizzoni and M. Caccamo. Toward the predictable integration of real-time COTS based systems. In *RTSS*, pages 73–82. IEEE Computer Society, 2007.
6. J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing predictability of cache replacement policies. *Real-Time Systems*, 37(2):99–122, 2007.
7. J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, pages 49–60, 2007.
8. L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Systems*, 28(2-3):157–177, 2004.
9. R. Wilhelm. Determining Bounds on Execution Times. In R. Zurawski, editor, *Handbook on Embedded Systems*, pages 14–1, 14–23. CRC Press, 2005.
10. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem—overview of the methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 2008.
11. R. Wilhelm, D. Grund, J. Reineke, M. Pister, M. Schlickling, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future time-critical embedded architectures. *IEEE TCAD*, 28(7), 2009.