Towards Multicore WCET Analysis*

Simon Wegener

AbsInt Angewandte Informatik GmbH, Saarbrücken, Germany
wegener@absint.com

Abstract

AbsInt is the leading provider of commercial tools for static code-level timing analysis. Its `aiT Worst-Case Execution Time Analyzer` computes tight bounds for the WCET of tasks in embedded real-time systems. However, the results only incorporate the core-local latencies, i.e. interference delays due to other cores in a multicore system are ignored. This paper presents some of the work we have done towards multicore WCET analysis. We look into both static and measurement-based timing analysis for COTS multicore systems.

1998 ACM Subject Classification C.4 Performance of Systems, D.2.4 Software/Program Verification

Keywords and phrases Worst-Case Execution Time (WCET) Analysis for Multicore Processors, Real-time Systems

Digital Object Identifier 10.4230/OASIcs.WCET.2017.7

1 Introduction

"The problem of determining upper bounds on execution times for single tasks and for quite complex processor architectures has been solved" [23]. While this statement was true a decade ago, when safety-critical embedded systems only used singlecore processors, it no longer holds since the emergence of the multicore processor in the hard real-time context.

The problem for the timing analysis of multicore systems are the interference delays due to conflicting, simultaneous accesses to shared resources, like for example the main memory (see Figure 1). On a singlecore system, the latency of a memory access mostly depends on the accessed memory region (e.g. slow flash memory vs. fast static RAM) and whether the accessed memory cell has been cached or not. On a multicore system, the latency also depends on the memory accesses of the other cores, because multiple simultaneous accesses might lead to a resource conflict, where only one of the accesses can be served directly, and the other accesses have to wait. These interference delays need to be included in a worst-case assessment.

As part of our ongoing work towards multicore timing analysis, we looked in the last few years into methods for multicore timing analysis, strategies to reduce resource conflicts, and COTS multicore architectures. The paper at hand contains the findings of this survey. Additionally, we reviewed the AURIX TC27x and its potential for multicore timing analysis. To the best of our knowledge, such a review of the AURIX TC27x has not been published yet.

This paper is organized as follows: In Section 2, we describe approaches for multicore WCET analysis. In Section 3, we present some strategies to reduce the amount of resource conflicts, which helps to improve the results of a multicore WCET analysis. In Section 4, we

* This work was funded within the project ARAMiS II by the German Federal Ministry for Education and Research with the funding ID 01IS16025B. The responsibility for the content remains with the authors.

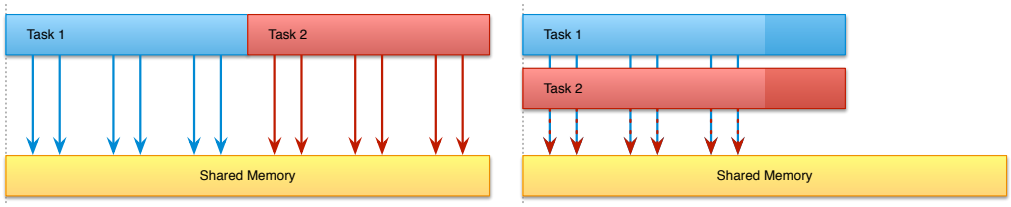

Figure 1 Two tasks which access the shared memory. On a singlecore system (left), each access takes a certain amount of time, but is not delayed due to resource conflicts. On a multicore system (right), resource conflicts happen due to simultaneous accesses to the shared memory. Consequently, the overall execution time of each task is increased.

introduce two multicore architectures and give some hints concerning their suitability for WCET analysis. In Section 5, we list some related work. Finally, in Section 6, we conclude and present some of the future work.

2 WCET Analysis for Multicore Systems

Any sound WCET analysis targeting multicore systems must take the interference delays into account that are caused by resource conflicts. Ignoring these delays might result in underestimation of the real WCET. Assuming full interferences at all times, however, is also not a solution, but might result in huge overestimation. Therefore, the interferences, and consequently, the resource conflicts, have to be analysed in order to get precise results. There a basically two possibilities for such an analysis:

- One can perform a joint analysis of all tasks and cores of the system. This way, the scheduling of the tasks and their allocation to the cores is known to the microarchitectural analysis. While this type of analysis may produce the most precise results, it is often disregarded due to the high computational complexity, rendering this approach infeasible.
- One can first perform separate WCET analyses for each task on each core, ignoring all interferences from the outside. Later, in a second step, the costs due to the interferences are analysed and incorporated into the results from the former analyses. This is the same scheme that is already applied on singlecore systems to derive the worst-case response time which incorporates communication delays and task switch/preemption costs into the WCET bound. Albeit being computationally easier, this approach needs to take extra care for the many non-timing-compositional features of modern processor architectures.

2.1 Static WCET Analysis

For static WCET analysis, we propose to use the second approach to tackle multicore systems, i.e. separate singlecore WCET analysis on the code level and an interference analysis on the system level. For singlecore WCET analysis, AbsInt provides `aiT` [1]. Supported multicore architectures are, among others, the Infineon AURIX TC275 and TC277.

To support interference analysis, we currently implement Worst-Case Resource Access (WCRA) analyses in the `aiT` framework. Here, the microarchitectural analysis is used to determine the maximal number of accesses to a shared resource that can occur during a task's execution. One example for such a resource is the shared memory. This number can be used to estimate the influence of a task on the timing of other tasks.

As always, the more predictable a system is, the easier it is to analyse it statically, and the more precise are the results [6, 8, 24]. This is in particular true for the interference delays.

However, since memory accesses are orders of magnitude slower than normal instructions, one can argue that the pipeline will drain during the processing of memory accesses. Thus, the interference delays imposed by resource conflicts do not cause timing anomalies and can be added later to the singlecore WCET bound [19].

Under this assumption (timing compositionality), singlecore WCET bounds and WCRA bounds can be used to estimate the multicore WCET in a system level analysis. The precise design of such an analysis depends on the type of delay caused by resource contention. An overview on the different kinds of approaches concerning contention on accesses to shared resources is given in [10].

Clearly, the assumption that interference delays do not cause timing anomalies is a rather strong assumption. According to recent research [13], there are two possible ways to gain timing compositionality: sound penalties and compositional base bounds. A sound penalty comprises all direct and indirect effects on the execution time in case of a resource conflict. Unfortunately, there is no known method to compute such sound penalties that generally hold. However, the authors conducted some experiments to find sound penalties for a set of benchmark programs. They observed that the impact of indirect effects increases with shorter memory latencies, because the pipeline is less likely to hide these effects behind long running memory accesses. This observation gives some justification for the above assumption, although no thorough research has been conducted yet to validate it.

The second approach are compositional base bounds. Here, the singlecore WCET bound is augmented with a safe approximation of the possible indirect effects. Then, in the system-level analysis, the direct costs of resource conflicts (the interference delays) are added to the singlecore WCET bound. Since the indirect effects are already incorporated, this gives sound results – even for non-compositional architectures. However, computing the safe approximation of the indirect effects might be computationally expensive. We refer to [13] for the details of this approach.

2.2 Hybrid (Measurement-based) WCET Analysis

In the last years, we also investigated hybrid measurement-based approaches to WCET analysis of multicore systems. For example, we developed together with Accemic and TU Darmstadt a system for the non-intrusive continuous analysis of a dualcore system based on the ARM Cortex-A9 [9].

By its nature, an analysis using measurements to derive timing information is always a joint analysis, because the effects of other running cores are directly visible in the measurements. The quality of the measurements depends on the source of measured events. To avoid the probe effect, one needs to forego software instrumentation. Embedded trace units of modern processors, like Nexus 5001™ [14] or ARM CoreSight™ [5] allow the fine-grained observation of a core's program flow.

When using measurement-based methods, one needs to take care that all possible execution scenarios are observed during the measurements. With end-to-end measurements, this is normally not possible. Instead, one applies hybrid approaches where short snippets are measured and later combined in a static path analysis. The assumption is that it is easier to measure the worst case for each of the snippets the more fine-grained they are. Moreover, when the snippets match the basic block model often used in static WCET analysis, it is easier to perform the path analysis.

For modern Nexus-based ETUs like the one found in the NXP QorIQ P- and T-Series, this is unfortunately not the case. Here, Branch History Messages (BHM) are emitted for indirect branches. The program flow of the taken/not-taken direct branches since the last

BHM is encoded in the message as a sequence of bits. Thus, we do not get timing information for every branch. While the information in the BHM trace is enough to fully reconstruct the control flow, it does not directly map to the basic block model we normally use for path analysis. This is in particular true for small loops consisting of only one or two basic blocks which might be covered by only one BHM although multiple iterations have been executed.

Besides the obvious problem that one needs to generate enough input data to observe all possible (or all important) scenarios for measurement-based approaches, there are (at least) two more multicore-specific problems to consider:

- When multiple cores are running, they also generate a multiple of the trace messages a singlecore system would emit. Thus, the limited bandwidth of the system's trace port might not suffice to transport all trace messages to the debugger. Consequently, some messages are lost (the trace contains "holes") and important timing information cannot be obtained. This leads to lower confidence in the resulting WCET estimate.
- Measurements also observe the timing effects of events like DRAM refreshes and resource conflicts. While this is exactly the reason why we consider measurement-based approaches for multicore timing analysis, this might also lead to huge overestimation. Consider the interference delays induced by the use of a shared interconnect. Assume now that one in ten accesses will suffer a severe interference delay. These delays will possibly occur for any access in the program. During the observation of the program's execution we measure for most accesses both the case where no interference happens as well as the case where the delay occurs. Due to the worst-case assessment, we will incorporate the delay for all these accesses. Thus, we overestimate the real WCET because many more accesses on the critical path will incorporate the delay than the "one in ten" ratio suggests.

Besides their use in WCET analysis, measurements can be used to construct execution time profiles [7]. These are particularly useful for performance analysis, because they also cover the average case.

3 Strategies to Reduce Resource Conflicts

There are several strategies to reduce the amount of resource conflicts, either by controlling when accesses to shared resources happen or by limiting the amount of accesses to shared resources. In the following, we want to present some of them.

3.1 Privatisation of Shared Resources

One idea to avoid resource conflicts is the privatisation of shared resources. This can be done by assigning the resource to a specific core for a limited amount of time. In this timespan, the core has the sole ownership over the resource and no other core is allowed to access the resource.

Schranzhofer et al. presented in [22] a TDMA-based resource scheduling, where each task is split into an input/output phase at the beginning, a computation phase in the middle, and an input/output phase again at the end of its runtime. The tasks on the different cores are then started with an offset in such a way that the input/output phases of the different tasks do not overlap (see Figure 2). This scheduling avoids the resource conflicts altogether, but needs changes to existing code (and maybe operating system (OS) support) in order to be implemented.

Another scheme for resource privatisation was presented in [8]. Here, warm-up and cool-down phases are added to each task (see Figure 3). In the warm-up phase, data is copied

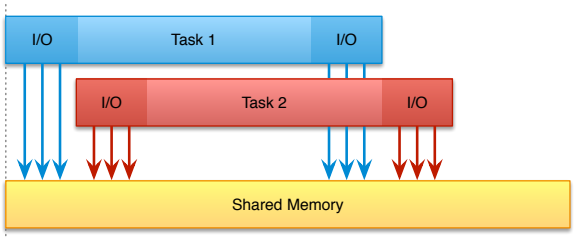

Figure 2 Privatisation of shared resources by applying a TDMA-based scheduling of accesses to shared resources. Resource conflicts are avoided because the input/output phases of different tasks do not overlap.

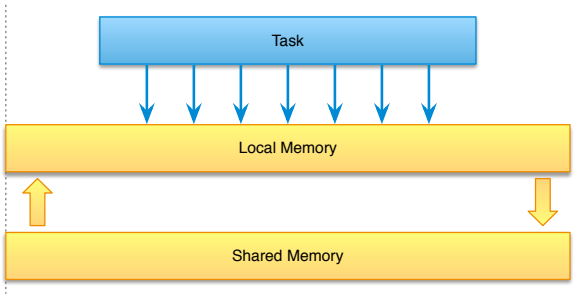

Figure 3 Privatisation of shared resources by copying data from shared to local memory and back again in warm-up and cool-down phases. Resource conflicts are avoided because no accesses happen to shared memory during the task's execution.

from the shared memory to local memory. In the cool-down phase, data is copied from the local memory to shared memory. This approach gives some control when resource conflicts occur, as they can only happen during warm-up and cool-down phases. To completely avoid them, use a TDMA-based scheduling for the warm-up and cool-down phases, similar as above. Naturally, this scheme only works if the target architecture has some local memory that can be used for privatisation. Moreover, tool or OS support is needed to implement the warm-up and cool-down phases. On the other hand, existing code does not need to be changed. However, the scheme may have severe performance overhead.

3.2 Runtime Resource Capacity Enforcement

Instead of avoiding resource conflicts, one can also limit them to an amount such that all tasks still do not miss their deadline, even when we assume full interference for the amount of accesses which equals the aforementioned limit. This concept is called runtime resource capacity enforcement and was presented in [19].

Here, one analyses each task regarding WCET and WCRA (see Section 2.1). Then, one determines how many accesses assuming full interference are allowed for a task t_0 in order to not miss its deadline. This is the amount of resource accesses that all other tasks are allowed to perform during the execution of t_0. Each other task t_x is then assigned its capacity, i.e. its fair share of this amount. A runtime monitoring system observes the number of actual resource accesses of each task. The OS checks that the capacity of a task is not exceeded (and suspends the task otherwise). This scheme is in particular helpful for mixed-criticality scenarios.

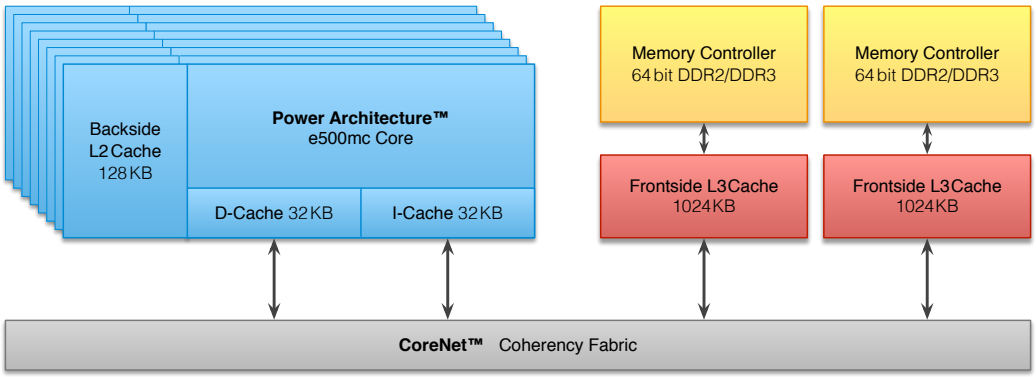

Figure 4 Block diagram of the Freescale P4080.

4 Multicore Platforms

In the following, we assess two different multicore platforms and their suitability for WCET analysis. The Freescale QorIQ P4080 has been used as one of the evaluation platforms in the German research project ARAMiS [3]. It was also the platform of choice for several research papers (e.g. [18, 19]). The Infineon AURIX TC27x is considered as an evaluation platform in the German research project ARAMiS II [4].

4.1 Freescale QorIQ P4080

The Freescale P4080 [11] is a prominent example of a multicore platform where the interference delays have a huge impact on the memory access latencies. Nowotsch et al. [19] measured maximal write latencies of 39 cycles when only one core was active, and maximal write latencies of 1007 cycles when all eight cores were running.

The P4080 consists of eight PowerPC e500mc cores running at 1.5 GHz. The memory hierarchy of the P4080 looks as follows (see also Figure 4): Each core has separate L1 instruction/data caches and a unified L2 cache. The L1 caches have a capacity of 32 KB[1] each, whereas the L2 cache has a capacity of 128 KB. The cores communicate with each other and the main memory over a shared interconnect, the CoreNet Coherency Fabric. Via this interconnect, two memory controllers attach DDR2/DDR3 RAM to the cores. Each of them sits behind 1 MB of unified L3 cache.

The P4080 is a non-compositional architecture according to the classifications in [24] due to its use of PLRU and FIFO replacement policies in caches, translation lookaside buffers, branch target buffers and branch history tables. However, it can be configured in a more predictable way avoiding domino effects:

- Dynamic branch prediction can be switched off.
- TLBs can be preloaded to avoid misses.
- The L1 data cache can be used in write-through mode.
- The L2 cache can be used as scratchpad memory.
- Partial cache locking can be used to gain LRU replacement policy.

[1] For reasons of compatibility with the processor manuals, we use the abbreviation KB for 2^{10} bytes, and MB for 2^{20} bytes.

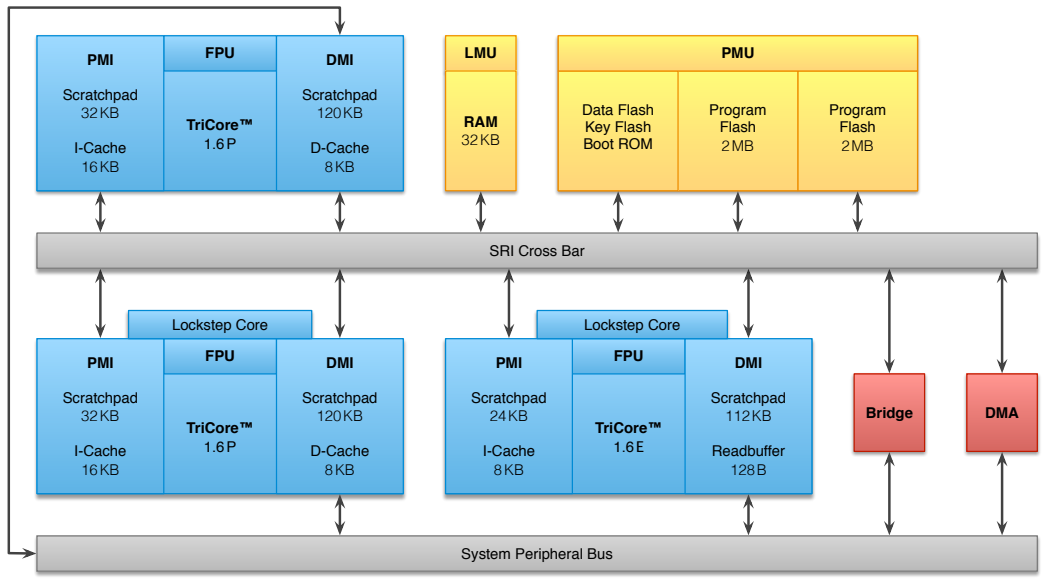

Figure 5 Block diagram of the Infineon AURIX TC27x.

Unfortunately, there is not enough documentation publicly available concerning the CoreNet. Thus, no static analysis predicting its behaviour can be developed. Latencies of memory accesses crossing the interconnect must therefore be obtained by other means, e.g. measurements and incorporated in a response time analysis.

The e500mc core is a complex processor architecture. Constructing and validating a mathematical model suitable for static singlecore timing analysis might take several man-years. `aiT` currently does not support the P4080. However, `aiT` supports the Freescale MPC7448 which has a similarly complex core but is a singlecore architecture.

4.2 Infineon AURIX TC27x

The Infineon AURIX TC27x [15] (Figure 5) is a multicore processor widely used in the automotive domain. It consists of two TC1.6P (performance) cores and one TC1.6E (efficiency) core. One of the performance cores and the efficiency core have attached a checker core such that they can run in lockstep mode. This lockstep mode does not affect the timing behaviour of the system (except in case of disagreement, then a failure is reported to the Safety Management Unit, which may trigger a handler).

Memory and Caches. Each of the performance cores has 120 KB of data scratchpad RAM and 32 KB of program scratchpad RAM. Additionally, each of them has 16 KB of instruction cache and 8 KB of data cache. The efficiency core has 112 KB of data scratchpad RAM and 24 KB of program scratchpad RAM. Additionally, it has 8 KB of instruction cache and a 128 bytes wide read buffer. The caches are organized as two-way set associative caches with LRU replacement strategy. The data caches are write-back caches, but they can be bypassed.

The three cores of the AURIX TC27x are attached via the Program Memory Interface (PMI) to the SRI Cross Bar (SRI), and via the Data Memory Interface (DMI) to the SRI and the System Peripheral Bus (SPB). All peripherals are attached to the SPB except the On-Chip Debug Support (OCDS), which is connected to the SRI via the DMA interface.

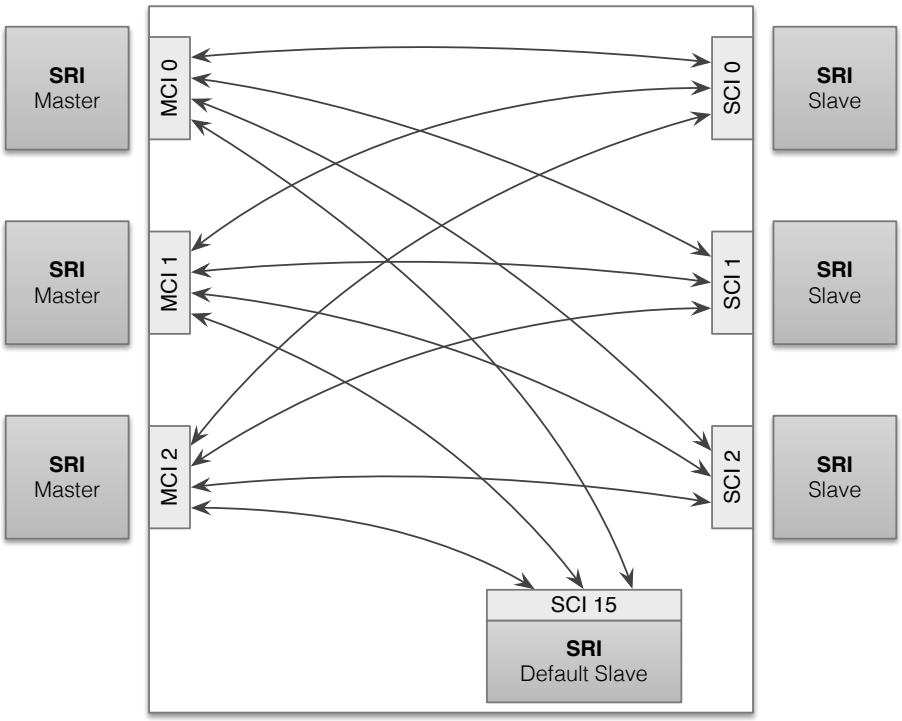

Figure 6 Overview of the structure of the SRI Cross Bar.

Attached to the SRI are also the Local Memory Unit (LMU) and the Program Memory Unit (PMU). The LMU provides 32 KB of shared SRAM to the system, whereas the PMU provides two independent program flash memories (2 MB each) and a data flash memory (464 KB). Each of the flash memories has its own port on the SRI.

SRI Cross Bar

The SRI (Shared Resource Interconnect) Cross Bar connects up to 16 bus masters with up to 15 slaves (and one additional default slave) via point-to-point connections (see Figure 6). The default slave has two purposes. First, it handles the accesses to the SRI configuration registers. Second, the error handling concerning the SRI is done here. The three DMIs and three PMIs are master devices for the SRI, as is the DMA interface. A resource conflict happens when two or more master devices try to access the same slave device. Each slave has its own arbiter to handle these resource conflicts (see Figure 7).

The arbitration rules are as follows: On the top level, the priorities of the master devices decide which request is handled first. The highest priority is 0, the lowest priority is 7. Only one master is allowed to have the same priority, except for the priorities 2 and 5. Here, an additional level of arbitration is performed. All masters with priority 2 form a round robin group, the masters with priority 5 form another one. Within these groups, round robin scheduling is done for arbitration.

Moreover, the arbiters contain a mechanism for starvation prevention. Starvation can happen if some high priority master continuously accesses the same slave such that a master with lower priority never gets its access granted. To prevent this, some kind of priority ceiling is performed when an access is not granted for a configurable timespan.

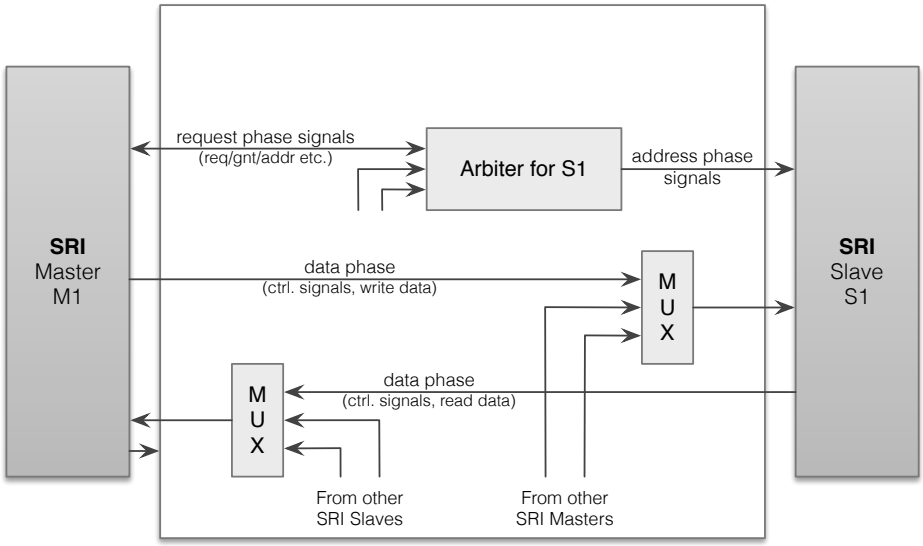

Figure 7 Detailed view on one point-to-point connection in the SRI Cross Bar. Each slave has its own arbiter.

The SRI Cross Bar is well documented (about 70 pages) in the AURIX TC27x user manual [15]. It should be possible to derive the necessary formulas to predict the number of wait cycles depending on the number of conflicting accesses. However, the concrete derivation of these formulas remains future work.

Branch Prediction. Both the TC1.6P and the TC1.6E cores use branch prediction mechanisms to improve the performance. The TC1.6E core uses a static branch prediction scheme with the following rules: Branches in the 16-bit instruction format are predicted taken. Branches in the 32-bit instruction format which point backwards are predicted taken. Branches in the 32-bit instruction format which point forwards are predicted not taken. The TC1.6P core uses a dynamic branch prediction scheme that is not further explained in the manual. A sound static analysis thus has to take both cases – correct prediction and misprediction – into account.

Predictability and Proposed Configuration. Given the fact that the TC1.6P pipeline can execute up to three instructions in one clock cycle, and its use of dynamic branch prediction, the occurrence of timing anomalies is likely. However, we assume that the AURIX is a compositional architecture with constant-bounded effects [24].

`aiT` supports the Infineon AURIX TC27x and can be used to compute singlecore WCET bounds for each of the three cores. The only thing that is not supported by `aiT` are the write-back caches. Hence, the data caches need to be bypassed. Moreover, the analysis model of aiT assumes that no other SRI master devices besides the analysed core access the same slaves, i.e. that no resource conflicts happen in the SRI Cross Bar.

In the following, we propose a configuration to minimise the amount of resource conflicts when used in a multicore scenario:

- Use one dedicated program flash memory for each of the performance cores to avoid conflicting accesses. Use the data flash for the efficiency core, if needed.

- Use the core-local data scratchpad whenever possible instead of the shared RAM to reduce conflicting accesses. If possible, preload data from the shared RAM and data flash to the local scratchpad memories to control when accesses to the shared memory happens.
- Place the stack in the core-local data scratchpad.
- Do not access the core-local scratchpad memories from other cores.
- I/O channels (CAN, FlexRay, ...) should not be accessed by multiple cores. Assign each I/O channel in use to a specific core.

5 Related Work

WCET Analysis for multicore systems is an active area of research. At least 17 publications concerning multicore timing analysis have been presented at the past five instances of the International Workshop on Worst-Case Execution Time Analysis. These cover all areas of timing analysis, e.g. static approaches [12], measurement-based approaches [17], resource arbitration [16], hardware [21], mixed-criticality [2] and response time analysis [20].

6 Conclusion and Future Work

In this paper, we presented some of the work that has been carried out at AbsInt to provide tool support for multicore WCET analysis. We currently focus on WCRA analysis to estimate the influence of resource conflicts on the timing of a task, and on BHM traces for hybrid measurement-based timing analysis. Moreover, we gave hints how the Freescale QorIQ P4080 and the Infineon AURIX TC27x, two popular multicore architectures, can be used for real-time embedded systems. In the course of ARAMiS II, we want to evaluate our newly developed analyses in industrial multicore scenarios.

Acknowledgements. I want to thank Michael Schmidt for providing me with the neatly drawn pictures, and Gernot Gebhard and Markus Pister for their valuable comments regarding the AURIX TC27x and the P4080.

References

1 AbsInt Angewandte Informatik GmbH. aiT Worst-Case Execution Time Analyzer. `http://www.absint.com/ait/`.

2 Sebastian Altmeyer, Björn Lisper, Claire Maiza, Jan Reineke, and Christine Rochange. WCET and Mixed-Criticality: What does Confidence in WCET Estimations Depend Upon? In Francisco J. Cazorla, editor, *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*, volume 47 of *OpenAccess Series in Informatics (OASIcs)*, pages 65–74, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.WCET.2015.65`.

3 ARAMiS. `http://www.projekt-aramis.de/`.

4 ARAMiS II. `https://www.aramis2.org/`.

5 ARM Ltd. CoreSight™ Program Flow Trace™ PFTv1.0 and PFTv1.1 Architecture Specification, 2011. ARM IHI 0035B.

6 Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard von Hanxleden, Reinhard Wilhelm, and Wang Yi. Building timing predictable embedded systems. *ACM Transactions on Embedded Computing Systems*, 13(4):82:1–82:37, 2014. `doi:10.1145/2560033`.

7 Thomas Ballenthin, Boris Dreyer, Christian Hochberger, and Simon Wegener. Hardware Support for Histogram-based Performance Analysis of Embedded Systems. In *20th IEEE International Symposium On Real-time Computing (ISORC 2017)*. IEEE, 2017. (accepted).

8 Christoph Cullmann, Christian Ferdinand, Gernot Gebhard, Daniel Grund, Claire Maiza (Burguière), Jan Reineke, Benoît Triquet, Simon Wegener, and Reinhard Wilhelm. Predictability Considerations in the Design of Multi-Core Embedded Systems. *Ingenieurs de l'Automobile*, 807:26–42, 2010.

9 Boris Dreyer, Christian Hochberger, Alexander Lange, Simon Wegener, and Alexander Weiss. Continuous Non-Intrusive Hybrid WCET Estimation Using Waypoint Graphs. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2016.4.

10 Gabriel Fernandez, Jaume Abella, Eduardo Quiñones, Christine Rochange, Tullio Vardanega, and Francisco J. Cazorla. Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art. In Heiko Falk, editor, *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *OpenAccess Series in Informatics (OASIcs)*, pages 31–42, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2014.31.

11 Freescale Semiconductor, Inc. QorIQ™ P4080 Communications Processor Product Brief, Rev. 1, 2008. http://cache.freescale.com/files/32bit/doc/prod_brief/P4080PB.pdf.

12 Andreas Gustavsson, Jan Gustafsson, and Björn Lisper. Toward Static Timing Analysis of Parallel Software. In Tullio Vardanega, editor, *12th International Workshop on Worst-Case Execution Time Analysis*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 38–47, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2012.38.

13 Sebastian Hahn, Michael Jacobs, and Jan Reineke. Enabling compositionality for multicore timing analysis. In *Proceedings of the 24th International Conference on Real Time and Networks Systems*, October 2016. URL: http://embedded.cs.uni-saarland.de/publications/EnablingCompositionalityRTNS2016.pdf, doi:10.1145/2997465.2997471.

14 IEEE-ISTO. IEEE-ISTO 5001™-2012, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface, 2012.

15 Infineon Technologies AG. AURIX™ TC27x D-Step 32-Bit Single-Chip Microcontroller User's Manual V2.2 2014-12, 2014.

16 Timon Kelter, Tim Harde, Peter Marwedel, and Heiko Falk. Evaluation of resource arbitration methods for multi-core real-time systems. In Claire Maiza, editor, *13th International Workshop on Worst-Case Execution Time Analysis*, volume 30 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–10, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2013.1.

17 Leonidas Kosmidis, Davide Compagnin, David Morales, Enrico Mezzetti, Eduardo Quinones, Jaume Abella, Tullio Vardanega, and Francisco J. Cazorla. Measurement-Based Timing Analysis of the AURIX Caches. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2016.9.

18 Jan Nowotsch and Michael Paulitsch. Leveraging multi-core computing architectures in avionics. In Cristian Constantinescu and Miguel P. Correia, editors, *2012 Ninth European Dependable Computing Conference, Sibiu, Romania, May 8-11, 2012*, pages 132–143. IEEE

Computer Society, 2012. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6214741, doi:10.1109/EDCC.2012.27.

19 Jan Nowotsch, Michael Paulitsch, Daniel Buhler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 109–118. IEEE Computer Society, 2014. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6909098, doi:10.1109/ECRTS.2014.20.

20 Dumitru Potop-Butucaru and Isabelle Puaut. Integrated Worst-Case Execution Time Estimation of Multicore Applications. In Claire Maiza, editor, *13th International Workshop on Worst-Case Execution Time Analysis*, volume 30 of *OpenAccess Series in Informatics (OASIcs)*, pages 21–31, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2013.21.

21 Martin Schoeberl, David Vh Chong, Wolfgang Puffitsch, and Jens Sparsø. A Time-Predictable Memory Network-on-Chip. In Heiko Falk, editor, *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *OpenAccess Series in Informatics (OASIcs)*, pages 53–62, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.WCET.2014.53.

22 Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. Timing predictability on multi-processor systems with shared resources. In *Workshop on Reconciliating Predictability and Efficiency at EMSOFT 2009*, 2009.

23 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008. doi:10.1145/1347375.1347389.

24 Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, and Christian Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 28(7):966–978, 2009. doi:10.1109/TCAD.2009.2013287.