Challenges of WCET Analysis in COTS Multi-core due to Different Levels of Abstraction

Hardik Shah[1], Andreas Raabe[1], and Alois Knoll[2]

[1] fortiss GmbH, Guerickestrasse 25,
80805 Munich, Germany
[2] Department of Informatics VI, Technical University Munich,
85748 Garching, Germany
{shah,raabe}@fortiss.org
www.fortiss.org

Abstract. The continuous demand of producing low cost multi-core hardware for safety critical hard real time applications has driven attempts of using Commercial Off The Shelf (COTS) components. Prior to the industrial deployment, Worst Case Execution Time (WCET) of these applications must be estimated on underlying hardware. However, even simple COTS components exhibit unpredictability at very low level which render them not suitable for conservative timing analysis. These low level unpredictability, implied from hardware specifications, are mostly invisible to MPSoC integrator and WCET tool manufacturers due to abstraction. Additionally, *predictable* hardware designers propose advance techniques for predictable and high performance multi-core systems ignoring abstraction that tool manufactures use. The combination of such advance hardware and abstraction enforces investigation of all paths through application execution.
As contribution, this paper identifies two low level unpredictability deduced from COTS interconnect specifications which render them unsuitable for WCET analysis in multi-core environment. Moreover, the paper also shows that the advance arbitration scheme, budget based arbitration, invented for better predictability and performance prevents WCET tool to apply state space reduction and enforces analysis of all paths.

1 Introduction

In state-of-the-art premium automobiles about 100 Electronic Control Units (ECUs) are used to control the vehicles' functionality ranging from passenger comfort (e.g. climate control, infotainment) to safety-critical tasks (e.g. ABS, ESP, airbag) each of which has their own distinct requirements. The complexity of the implemented functionality can be expected to even grow in the future. The traditional approach to satisfy requirements of such functionalities is to build dedicated ECUs and group them according to their class by dedicated networks e.g. CAN, LIN, MOST, FlexRay etc. It is estimated that about 4 km of cables are used in today's premium cars weighing about 60 kg. This incurs a tremendous fuel consumption. Moreover, this approach has increased complexity and vehicle cost by 30 - 40% [26].

A more centralized Information and Communication Technology (ICT) architecture would therefore exhibit tremendous advantages, such as: reduced weight (due to reduction in number of components and cabling), reduced cost (due to simplified architecture), and increased configurability and reduced maintenance cost of the overall ICT structure (due to replacement of hardware functionality by software components).

For such a centralized ICT architecture high performance ECUs capable of executing multiple demanding applications concurrently are required. Multi-core architectures are seen as a potential candidate for providing the performance needs at low energy consumption due to their high performance-per-watt ratio. Custom multi-core chip production tailored to safety critical systems is very cost intensive. Therefore, it is very tempting to apply multicore chips from other domains in order to reduce cost.

To this end we evaluated various technologies with respect to their applicability in the field. Since it is practically infeasible to evaluate each and every part of each and every chip with respect to every relevant property we needed to focus, in this paper we concentrate on the shared-memory communication subsystem, which we identified as a major source of potential problems. Certifying highly integrated systems to the applying safety standards (IEC 61509 and its automotive derivative ISO 26262) require safety-critical hard-real-time functionality to prove that they adhere to their time bounds. For example, a brake needs to react within 50ms after the braking pedal is hit by the driver. Therefore, we also focused on problems related to boundedness of memory response time (memory latency), which contributes to a major part to the worst-case-execution-time predictability of the overall platform and the quality of the achievable bound.

Neither is our goal to identify a single applicable existing chip, nor to provide a complete overview of the market. Instead we report on the problems we came across while we tried to compose a predictable multicore system from IP components. This included Commercial-Off-the-Shelf (COTS) components for parts that are usually not questioned. Here, we analyzed multiple common communication protocols e.g. AMBA, Avalon, PLB with respect to their applicability and identified problems on multiple levels. Foremost, we identified specification compliant behavior that will lead to unbounded or increased delays. Most of the research on COTS for predictability focus on either shared resource itself e.g SDRAM or predictable arbitration schemes, however, none focus on such specification compliant misbehavior of bus masters and arbiters. We also evaluate solutions such as budget based arbitration (Credit Controlled Static Priority [16], Priority Based Budget Scheduler [17] and Dynamic Priority Queue [11]) that are not yet wide spread in practice, but are propagated by research groups. Here, we also identified unsolved challenges in conjunction with WCET analysis methods.

2 Related Work

There has been a number of attempts to provide multicore hardware that is tailored to WCET analyzability (e.g., PRET [14], MERASA [12], CoMPSoC [13]). However, due to the tremendous costs of silicon implementation for small volume productions, tailored architectures are an expensive option for industry.

Therefore, a number of approaches propose techniques of providing WCET for COTS based components to enable their utilization in safety-critical, hard-real-time (HRT) applications. Pellizzonni et al [10], [6] analyze effects of memory traffic generated by I/O peripherals on WCET of the application under analysis in single core architecture. They extend their approach to multi-core architectures in [7]. Their approach can be summarized as follows. 1) Limit peripheral traffic to a certain amount per unit time. 2) Calculate the upper bound on memory traffic generated by co-existing applications 3) Estimate maximum possible interference considering those bounds. The advantage of this technique is that the worst case interference does not have to be considered which reduces the WCET of the task under analysis. The drawback is that the task cannot be analyzed in isolation, hence, a small bug fix in co-existing application enforces WCET re-analysis making it an iterative and time consuming process. Another drawback in mixed-critical system is that the safety standards [25] require all co-existing applications to be certified to the level of the most critical application integrated if temporal and spatial separation is not provided. In the worst case that might mean certifying an MP3 decoder application to the criticality level of a brake.

To simplify the certification and integration process Shah et al [8], [11], [15] and Paolieri et al [5] analyze applications in isolation and always assume worst case interference from co-existing applications. Their techniques provide WCET of application executing from highly unpredictable and inexpensive shared SDRAM. Although [5] is built for MERASA platform, we believe that their approach is applicable to any multi-core architecture with shared resources provided that the issues mentioned in this paper are taken care of. [8], [11] and [15] propose to use budget based arbitration and provide timing models of the arbitration scheme for worst case interference analysis. At first, these approaches demand change in arbiter which may be expensive. Moreover, we explain in this paper how their timing model when integrated in WCET analyzer, enforces analysis of all paths.

Recently, there have been approaches of measuring the WCET of applications on multi-core architectures by aggressively accessing the shared resource using artificial co-existing applications from Fernandez et al [22], Radojkovic et al [23] and Nowotsch et al [24]. However, we argue that such approaches cannot produce the theoretical worst case interference. Instead of proving it in this paper, we refer readers to [5]. Here, at first WCET is estimated (AMC) considering the theoretical worst case interference and compared to the Maximum Observed Execution Time (MOET) in the presence of always interfering applications. For a three master system the AMC to MOET ratio is ≈ 0.4 and it grows with number of masters.

Moreover, none of the above mentioned approaches analyze communication protocols to investigate suitability of their specifications for WCET analysis. Unpredictable behavior of COTS components are identified by Wilhelm et al [9] and Gebhard et al [4]. But, the main focus has always been the caches, pipelines, shared resource and arbiters. In this paper we extend the list especially considering temporally specification-compliant unpredictable behavior of bus masters in popular communication protocols, and the abstraction gap between hardware designers, system integrators and tool vendors.

3 Background

Fig. 1 is a basic predictable multi-core architecture with predictable cache and bounded off-chip memory access latencies. The off-chip memory is shared among all cores under predictable arbitration scheme. For static WCET analysis (aIT, OTAWA), abstract timing models of each of these components are created and worst case behavior is considered. Measurement based WCET analysis and its applied state space reduction are explained latter in Sec. 3.3. We now focus on points that we have identified causing issues while ignoring low level details.

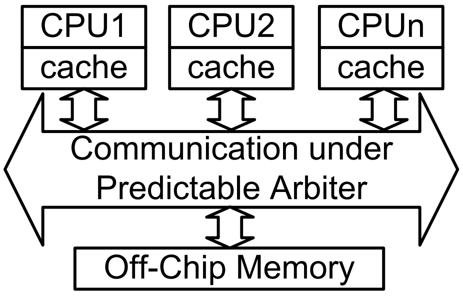
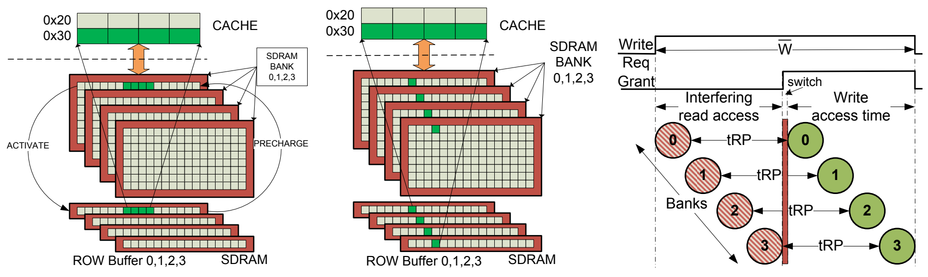


Fig. 1: Basic Predictable Multi-core Architecture

3.1 SDRAM - Off-chip Memory

SDRAM is a popular, inexpensive memory that provides large storage and high data rate. However, its timing unpredictability has restricted its use to non-realtime systems only. Recently, there have been some approaches in academia that predicts worst case latencies of SDRAMs and allows its use in safety critical, HRT systems [8], [11], [5], [18], [19]. All of them advocate to use bank interleaving (close page policy) while accessing SDRAM to reduce latency dependence on previous access. For more details on Bank Interleaving (BI), readers are referred to [8]. However, we briefly explain the concept in the following paragraph.

In BI, instead of mapping a single cacheline on a single row of an SDRAM bank (Fig. 2a), all cachelines are split and their chunks are mapped to all SDRAM



(a) Memory Mapping in Open Page Policy (b) Memory Mapping in Close Page Policy (c) Precharge to Activate delay for the same bank

Fig. 2: Bank Interleaving

banks (Fig. 2b). When a cache miss occurs, starting from Bank0, corresponding row is activated (copied to row buffer), cacheline data chunk is accessed from row buffer and the row is precharged (written back to its corresponding bank). When one row is being activated or precharged, data from other row can be supplied providing seamless data transfer. Using BI for accessing shared SDRAM results in more predictable latencies since unpredictability related to the presence of unintended row in the row buffer is eliminated. Otherwise, at first the unintended row must be precharged and the intended row must be activated which increases latency significantly (i.e. in open page policy - Fig. 2a). Activate and precharge operations are graphically represented in Fig. 2a.

Fig. 2c depicts "Precharge to Activate delay" (tRP) which dictates minimum time by which the next activation of the same bank is delayed after being precharged, hence, restricts SDRAM bandwidth. Moreover, as explained in [8], switching between read/write accesses produce worst latencies for SDRAM. Hence, for worst case estimation, when analyzing a write access we will assume it is interfered by a read access. Variable $\overline{W}$ is the worst case write latency in case of only one possible interfering master. Similarly, $\overline{R}$ is derived for read latency. Variables $\overline{W}$ and $\overline{R}$ are then used to estimate WCET of application.

3.2 Budget Based Arbitration: DPQ

Time Division Multiple Access (TDMA) arbitration scheme for accessing a shared resource is termed as the most predictable arbitration scheme [21]. In TDMA, each master is assigned a unique time slot in bus cycle. Each master is the exclusive owner of its time slot and no other master is allowed even if the slot is unused. These hard restrictions generates very predictable behavior at the cost of shared resource utilization. To provide better utilization, researchers have proposed budget based arbitration scheme in which masters are assigned fixed budget of allowed transfers in a bus cycle (replenishment period). At the beginning of replenishment period, each master is replenished with its initial budget and on every access its budget is reduced by one. When master uses its entire budget it is termed ineligible for that replenishment period and cannot access

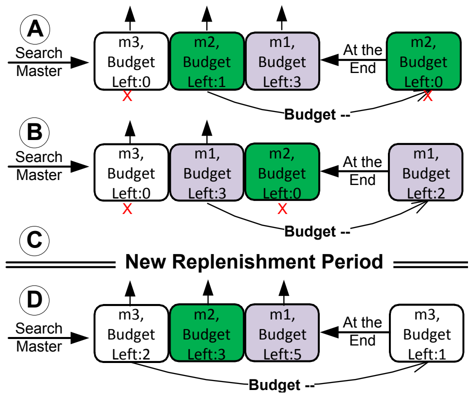


Fig. 3: DPQ Arbitration. Budget[m3,m2,m1] = [2,3,5]

the shared resource until it is replenished again at the beginning of next replenishment period. Due to the budget based assignments, masters do not have to wait until their slot arrives which increases resource utilization. Conflicts among masters in case of simultaneous accesses are resolved using priorities.

In this paper we will take Dynamic Priority Queue (DPQ) [11] as a case study which is fairer than other budget based arbitration schemes. In DPQ, as shown in Fig. 3, priority of a master depends on its position in the queue. At point A, master `m3` has the highest priority since it is at the left end of the queue. However, it has consumed its entire budget ($BudgetLeft = 0$), hence, it is ineligible in current replenishment period. Master `m2` is the next eligible master in the queue and has a request pending. Hence, `m2` is scheduled, its budget is reduced by one and it is enqueued at the end of the queue (lowest priority). However, priority of `m3` does not change. Only masters who had lower priority than `m2` (`m1` in this case) are shifted left to achieve next higher priority level. At point B, `m1` is the only eligible master with a pending request. Hence, `m1` is scheduled, its budget is reduced by one and it is enqueued at the end. At point C, a new replenishment period starts and all masters get their initial budget back making all of them eligible.

3.3 Measurement Based WCET

In this section we briefly explain the measurement based WCET analysis technique. As a case study we take an example of hybrid approach employed by Rapitime tool developed by Rapita systems [20]. Rapitime analyzes application code statically and inserts instrumentation points (I - Points) at the beginning of each basic block (Fig. 4(a)). By definition, a basic block is a block of code that has a single entry and a single exit points. Hence, branch instruction itself is one basic block. The i - point simply writes down the time when it was visited during execution in a trace. The generated trace is then analyzed to determine the highest time spent in each basic block. Hence, the test pattern must drive

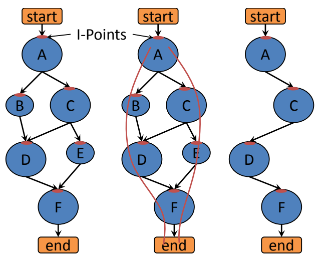
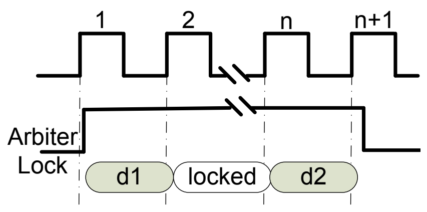


Fig. 4: (a) Basic Blocks (b) Tested Paths (b) WCET Path

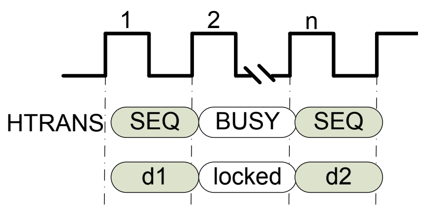
execution through all i - points at least once. However, testing all paths can be omitted. For example, in Fig. 4(b), the test patterns test paths ABDF and ACEF which cover all instrumentation points. After executing the test patterns, execution time of each basic block is determined (shown as big/small circles proportional to their execution time). The tool statically notices that there exists a path ACDF which is not tested. However, the path ACDF is the longest execution path. Hence, ACDF is the worst case path and total time consumed while executing basic blocks A, C, D and F is the WCET of the application.

4 Bus Locking

In this section, we investigate communication protocol required in the basic predictable architecture of Fig. 1. We present issues related to predictability in popular communication protocols, such as AMBA - AHB [1], Avalon [2] and PLB [3], which are invisible while WCET analysis due to abstraction. Overwhelming number of researchers use multi-core platforms built on these communication protocols and abstracts off-chip memory access time as a constant (typically 70 ns) for calculating cache miss penalty. Certainly, they take into account worst case interference and ideal memory controller, however, it is not enough as we will show.



(a) Explicit Bus Lock



(b) Implicit Bus Lock

Fig. 5: Shared Bus Locking

4.1 Explicit Bus Lock

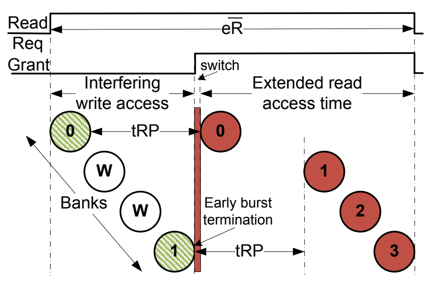
All above mentioned communication protocols offer a dedicated signal called `ArbiterLock` which can be used to inform the arbiter that once granted, the shared resource must remain granted to the master until the `ArbiterLock` signal is released as depicted in Fig. 5a. `ArbiterLock` signal is provided to improve average case performance by significantly reducing shared resource access latencies for masters which transfer large amount of data in a burst fashion.

It is clear that in all HRT systems, arbiter must not provide such explicit locking mechanism since one master can lock the bus indefinitely starving all other masters leading to deadline misses for applications being executed on them. In mixed critical systems with only one HRT application, the processor which executes HRT can have `ArbiterLock` signal. However, the shared resource can also be locked implicitly as we will show next.

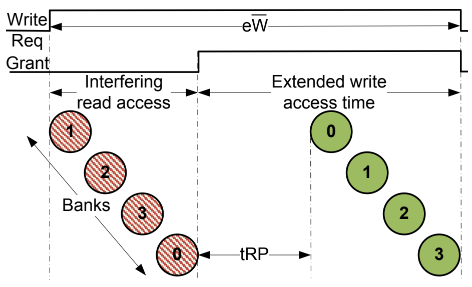
4.2 Implicit Bus Lock

Each of the above mentioned communication protocol supports burst transfers. In a burst transfer, a master requests ownership of the bus for a certain amount of consecutive accesses (`BurstLength`). The arbiter gives the guarantee that the bus ownership will not be terminated until the `BurstLength` transfers are done. Burst transfers are almost always used for filling up a cacheline and hence present in all modern processors with caches. During write burst transfers, if master itself cannot transfer data due to unavailability or wrap around in internal buffer, it can insert wait states (we have observed this using onchip debugger on an FPGA). In AMBA - AHB such wait states can be inserted by giving `BUSY` response (Fig. 5b), and in Avalon and PLB, by releasing `WRITE` signal. In this case, the master implicitly locks the bus until it completes `BurstLength` transfers.

4.3 Early Burst Termination



(a) Early Burst Termination



(b) Wrap Around Burst

Fig. 6: Extended latencies due to tRP Requirements

To avoid one master monopolizing the bus ownership, (ONLY) AMBA - AHB employs `EarlyBurstTermination` mechanism. Here, the ownership of a master can be terminated before `BurstLength` transfers are done if it takes longer than specified time to complete its burst transfer. However, this mechanism can increase latencies of co-existing applications when Bank Interleaving (Sec. 3.1) is used for accessing shared SDRAM. As shown in Fig. 6a, after writing first chunk of its burst on Bank0, the interfering master inserts two wait states. Afterwards, it writes the second chunk on Bank1. At this point, the arbiter intervenes and terminates its burst since it already took the specified time for a write transfer and the master under investigation is granted an access. The master under investigation reads first element from Bank0, however, reading the second element is delayed because of the tRP requirement (Sec. 3.1) as shown in Fig. 6a. Thus, the the worst case read latency is increased, depicted by $\overline{eR}$, invalidating previously analyzed theoretical worst case read latency $\overline{R}$. Although, master under investigation did not insert any wait state and the predictable arbiter enforces time by employing `EarlyBurstTermination`, the above mentioned "side-effect" invalidates theoretically analyzed WCET of application under investigation.

Wrap Around Burst: The similar problem exists for Wrap Around Burst (WAB). WAB is employed mainly in instruction caches and typically occurs when branch targets are not aligned to cache line boundaries. Here, instead of issuing sequential burst starting from the first element of the cacheline, burst is issued from the element where the branch target instruction is mapped to receive the blocking data (target instruction) first, which reduces processor stall time. However, as shown in Fig. 6b, wrap around burst issued by an interfering access extends latency of the master under investigation when bank interleaving is used.

4.4 Discussion

Above mentioned issues arise when system integrator integrates components from different vendors which are compliant to the same communication protocol. The system integrator focuses on compliance to the protocol and may not have information about inner functionalities of connected components. Moreover, the tool builder assumes theoretical worst case shared memory latency. Hence, these issues go completely unnoticed leading to invalid (bus locking) or underestimated (Early burst termination and Wrap around burst) WCET bound.

Issues related to bus locking can be fixed by publishing a subset of communication protocol (e.g. AHB^R - Realtime capable AHB) which does not allow explicit or implicit bus locking. All Intellectual Property (IP) vendors then have to adhere to this subset. If not, a delay buffer must be inserted at the output port of each master which at first receives all data from the master and then issues a sequential burst transfer without any wait state. The issue of wrap around burst can also be fixed at application code level by aligning all branch target instructions to cacheline boundary using compiler directives and thus, completely eliminating possibility of wrap around burst.

5 DPQ - Timing Model

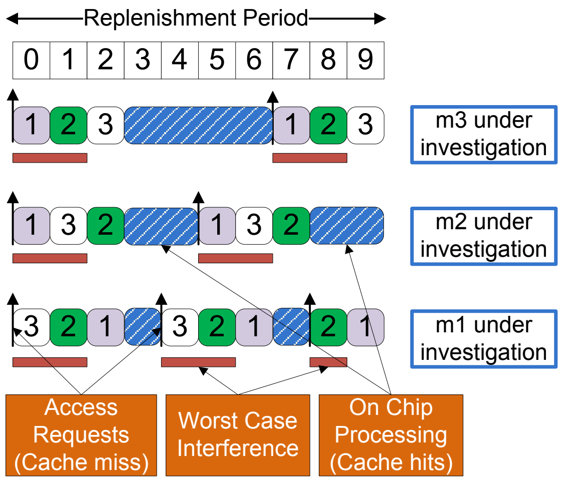


Fig. 7: DPQ - Worst Case Latency Analysis. Budget[m3,m2,m1] = [2,3,5]

The Fig. 7 depicts analysis view of a replenishment period under DPQ arbitration scheme [11]. During worst case interference analysis, the master under investigation is always assumed to be at the end of the queue when it requests an access to the share resource. For example, in the figure `m1` under investigation, cache miss occurs on `m1` at the beginning of the replenishment period. The co-masters (`m2` and `m3`) are assumed to experience a cache miss also at the same time. Since `m1` is assumed to be at the end of the queue, `m2` and `m3` are scheduled before `m1` is scheduled. The same is assumed for the second cache miss in the replenishment period. However, for a third cache miss of `m1` in the same replenishment period, `m3` is considered ineligible since it has already used its budget in current replenishment period. Here, only interference from `m2` is assumed. Thus in analysis, the worst case interference is assumed for early accesses in a replenishment period. However, in real execution, worst case interference can occur to any access provided that other masters have budget to do so.

The above mentioned technique eliminates requirement of assuming worst case interference always. However, that makes access latency dependent on the accesses done in near past. For example, consider a burst of accesses arriving in relatively short period. The early accesses of the burst are assumed to have high interference (high latency) provided that master under investigation has budget. Intermediate accesses are assumed to have less interference (low latency) since other masters have consumed their entire budget during high interference phase. Latter accesses of the burst are assumed to have high latency since the master does not have budget left in the current replenishment period and has to wait till the beginning of next replenishment period. Such dependence on accesses done in near past prevents application of deduced WCET paths as explained in Sec. 3.3 since the current access latency depends on accesses done in near past on that path, hence, that path must be tested. Thus, this technique makes WCET estimation an exhaustive process by enforcing analysis of all paths.

6 Conclusion

This paper has revealed issues related to WCET estimation in COTS communication protocol and advance arbitration schemes in multi-core architectures. The first two issues are related to implicit and explicit bus locking, and invalidation of theoretically analyzed worst case shared resource access latencies. These issues arise from specification of communication protocol. A subset of COTS specification with trivial fixes for real time capability (e.g. AHB^R) is required to mitigate the issues or delay buffers in hardware must be employed by an MPSoC integrator leading to an increase in latencies and WCET. These issues are mostly invisible to WCET analyzer due to the abstraction. However, they must be addressed before WCET estimation of application. The last issue is related to budget based arbitration. Here, researchers have proposed an advance arbitration algorithm and its timing model for worst case latency analysis. However, the technique prevents the WCET analyzer from applying path reduction leading to exhaustive analysis.

Above mentioned issues may go unnoticed due to the different levels of abstractions at which hardware designers, MPSoC integrators and WCET tool manufacturers operate. We believe for a safe WCET estimation, especially on multi-core architectures, unified efforts must be put from low level hardware designers to tool manufacturers for every component in the architecture.

7 Acknowledgment

This work is partly funded by German BMBF projects ECU (grant number: 13N11936) and Car2X (grant number: 13N11933). Authors would like to thank Dr. Kai Huang for his comments on improving the paper.

References

1. Advance microcontroller bus architecture (amba). www.arm.com
2. Avalon interface specifications. www.altera.com
3. Processor local bus (plb). www.ibm.com
4. G. Gebhard. Timing Anomalies Reloaded. In *WCET 2010*, Dagstuhl, Germany.
5. M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero. An Analyzable Memory Controller for Hard Real-Time CMPs. *Embedded Systems Letters, IEEE*, 2009.
6. R. Pellizzoni and M. Caccamo. Impact of peripheral-processor interference on wcet analysis of real-time embedded systems. *Computers, IEEE Transactions on*, 59(3):400 –415, march 2010.

7. R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *Proc. DATE*, 2010.
8. H. Shah, A. Raabe and A. Knoll. Bounding WCET of Applications Using SDRAM with Priority Based Budget Scheduling in MPSoCs. In *Proc. DATE*, 2012.
9. R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. 28(7), 2009.
10. R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of cpu and i/o transactions in cots-based embedded systems. In *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, pages 221–231, Washington, DC, USA, 2008. IEEE Computer Society.
11. Dynamic priority queue: An SDRAM arbiter with bounded access latencies for tight WCET calculation. Technical report, 2012. http://arxiv.org/abs/1207.1187
12. T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinandones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse and, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30:66–75, September 2010.
13. A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.*, 14:2:1–2:24, January 2009.
14. B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee. Predictable programming on a precision timed architecture. CASES '08, NY, USA.
15. H. Shah, A. Knoll and B. Akesson. Bounding Resource Interference: Detailed Analysis vs. Latency-Rate Analysis. In *Proc. DATE*, 2013.
16. B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration. In *Proc. RTCSA*, 2008.
17. M. Steine, M. Bekooij, and M. Wiggers. A priority-based budget scheduler with conservative dataflow model. In *Proc. DSD*, 2009.
18. B. Akesson, K. Goossens, and M. Ringhofer. Predator: a predictable SDRAM memory controller. CODES+ISSS '07, NY, USA.
19. J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. Pret dram controller: bank privatization for predictability and temporal isolation. CODES+ISSS '11, NY, USA.
20. http://rapitasystems.com/
21. C. Pitter and M. Schoeberl. A real-time java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–9:34, Aug. 2010.
22. M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla. Assessing the suitability of the ngmp multi-core processor in the space domain. In *Embedded Software (EMSOFT)*, pages 175–184, Tampere, Finland, 2012. ACM.
23. P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *ACM Trans. Archit. Code Optim.*, 8(4):34:1–34:25, Jan. 2012.
24. J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *Dependable Computing Conference (EDCC), 2012 Ninth European*, pages 132 –143, may 2012.
25. IEC 61508, functional safety of electrical/electronic/programmable electronic safety-related systems. http://www.iec.ch/functionalsafety/standards/
26. C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stahle, and A. Knoll. The software car: Building ICT architectures for future electric vehicles. In *Electric Vehicle Conference (IEVC), March 2012.*