Monitoring and WCET Analysis in COTS Multi-core-SoC-based Mixed-Criticality Systems

Jan Nowotsch*, Michael Paulitsch*, Arne Henrichsen†, Werner Pongratz†, Andreas Schacht†
*EADS Innovation Works, Munich, Germany, {Jan.Nowotsch, Michael.Paulitsch}@eads.net
†Cassidian, Friedrichshafen, Germany, {Arne.Henrichsen, Werner.Pongratz, Andreas.Schacht}@cassidian.com

***Abstract*—The performance and power efficiency of multi-core processors are attractive features for safety-critical applications, for example in avionics. But the inherent use of shared resources complicates timing analysability. In this paper we discuss a novel approach to compute the Worst-Case Execution Time (WCET) of multiple hard real-time applications scheduled on a Commercial Off-The-Shelf (COTS) multi-core processor. The analysis is closely coupled with mechanisms for temporal partitioning as, for instance, required in ARINC 653-based systems. Based on a discussion of the challenges for temporal partitioning and timing analysis in multi-core systems, we deduce a generic architecture model. Considering the requirements for re-usability and incremental development and certification, we use this model to describe our integrated analysis approach.**

Keywords*—*WCET, multi-core, temporal partitioning, safety-critical real-time systems

I. Introduction

While Worst-Case Execution Time (WCET) analysis for single-core processors is a well studied problem, the analysis of multi-core architectures poses significant challenges. In particular the inherent sharing of resources between processor cores complicates analysis, since for example memory accesses by other cores constitute random external events that may have a huge impact on the core-local performance. Usually, a concept called partitioning is used in domains such as avionics to avoid unintended interactions between applications in the spatial and temporal dimension. Unfortunately, the resource sharing in multi-core processors violates the assumptions of today's temporal partitioning implementations.

In recent years, several approaches for timing analysis of multi-core processors have been proposed. Typical approaches are the application of deterministic execution models [1], [2], [3], joint analysis [4], [5], [6], [7] and custom-designed hardware [8], [9]. Execution models tend to avoid resource sharing by Time Division Multiple Access (TDMA) schemes, enforcing resource privatization even though resources would allow parallel access. Joint approaches analyse in-parallel executed applications in conjunction to identify possible interferences. Considering the complexity of single-core analysis, the scalability of joint solutions with rising number of cores needs to be proven. Additionally, such an analysis does not fit the commonly used incremental development and certification processes of safety-critical application domains. Incremental processes are required since different applications are often implemented by different suppliers. Finally, custom hardware solutions avoid sharing, for instance through TDMA interconnects. Unfortunately, considering the trend towards highly integrated Commercial Off-The-Shelf (COTS) components, such as Multi-Processor Systems on Chip (MPSoCs), the potential for modifications to the hardware is fairly limited.

Facing the drawbacks of existing approaches, we propose a novel integrated approach for multi-core WCET analysis and temporal partitioning. The contributions of this work are:

Multi-core WCET analysis for independent analysis of multiple hard real-time applications, enabling true mixed-criticality workloads while avoiding resource privatisation.
Temporal partitioning based on runtime resource monitoring and enforcement, according to the requirements of safety-critical application domains.

Following recent trends in embedded and safety-critical domains, we focus on COTS hardware. We target the independent analysis of applications in order to reduce analysis complexity and enable incremental development and certification. Further, we avoid the restriction of resource privatisation in order to allow the utilisation of parallel resources.

The papers is structured as follows: we discuss the issues of COTS multi-core systems and introduce our abstract multi-core model in Section II. Based on this model we propose our temporal partitioning concept and identify potential issues, especially with respect to the predictability of modern processor architectures, in Section III. In Section IV, we discuss implementation aspects. The paper is concluded with a summary in Section V.

II. Abstract Multi-core Model

The main issue of multi-core processors for timing analysis is non-determinism due to the inherent use of shared resources [10], [11]. In particular, the parallel usage of a resource with limited capacity leads to unknown latency variations for individual requesters [12]. Hence the latency for a specific access is generally unknown. In the following, we call this effect shared-resource interference. As a consequence, temporal partitioning is violated since applications influence each other's timing behaviour. This invalidates incremental development and certification, as applications can no longer be developed and analysed independently.

To address unknown interference, we define an abstract multi-core model shown in Figure 1. By abstracting from resources and applications, we aim to replace unknown interference with bounded behaviour. Hence, methods based on this model are independent from concrete resource and application implementations. Considering most recent multi-core systems, the model is based on a shared-memory architecture.

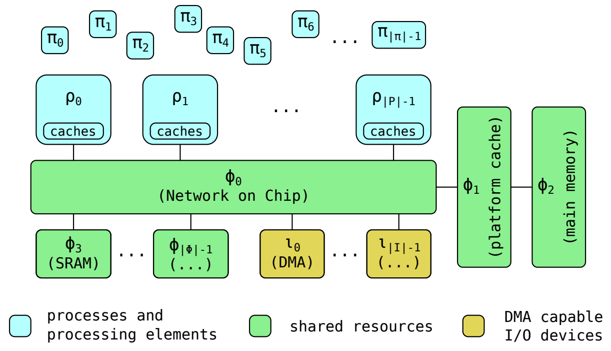

Fig. 1. Abstracted MPSoC architecture model Σ.

The abstracted model $\Sigma = (\Pi, P, \Phi, I)$ is a tuple comprising sets of processes $\pi_i\ \forall i \in [0, |\Pi|)$, processing elements (PEs) $\rho_j\ \forall j \in [0, |P|)$, shared resources $\phi_k\ \forall k \in [0, |\Phi|)$ and Direct Memory Access-capable Input/Output (DMA-I/O) devices $\iota_l\ \forall l \in [0, |I|)$. Naturally, a process π_i represents a virtual PE. Each PE implements an arbitrary pipeline architecture and local caches. While DMA-I/O devices also initiate memory accesses, they are separated from PEs, since they are not subject to software scheduling. Finally, the set of shared resources Φ includes all essential shared resources, such as the Network-on-Chip (NoC), platform caches or the main memory.

Based on the properties of the elements of Σ, we choose different representations for each of them. In particular, shared resources provide some kind of capacity, e.g bandwidth or memory space. This is abstracted as the resource capacity κ_{ϕ_k} per resource ϕ_k. PEs are under control of the underlying operating system schedule, which assigns processes to PEs. A set of processes that is scheduled in parallel over the available PEs is denoted $\Pi_{||}$, obviously $|\Pi_{||}| \leq |P|$. A process occupies a PE for some time and issues a number of requests to shared resources. The maximum execution time is commonly referred to as WCET. The maximum number of shared resource accesses has not yet a defined terminology in literature, hence we call them *Worst-Case number of shared Resource Accesses (WCRA)*. The WCRA can also be understood as the resource capacity or resource usage bound. Respectively, a process is represented by its WCET τ_{π_i} and its resource usage bounds $\kappa_{\pi_i}^{\phi_k}$ per used shared resource ϕ_k. Since DMA-I/O devices are not subject to process scheduling, they are only represented by their WCRA $\kappa_{\iota_l}^{\phi_k}$.

III. Timing Analysis and Temporal Isolation

The proposed concept is an integrated approach of WCET analysis and temporal partitioning. Runtime monitoring enforces the resource capacities of processes, which ensures temporal partitioning. Based on temporal partitioning and the abstraction of resources and processes, we introduce a WCET analysis that bounds the interference between processes without detailed information on in-parallel scheduled processes.

A. Temporal Partitioning

Temporal partitioning is a system property that isolates the timing behaviour of processes from each other. In effect, the execution of one process does not influence the timing behaviour of others. For single-core processors this has been ensured via multiplexing of processor time and careful handling of Input/Output (I/O) devices that may contend with the processor on a shared bus [13].

To enforce the resource capacity usage of our model, partitioning is split into a monitoring and a suspension task. The monitoring is responsible to track the actual resource usage of each process and trigger the suspension task once any process exhausts either of its capacities $\kappa_{\pi_i}^{\phi_k}$. In reaction, the suspension task must prohibit further usage of the respective resource by the process. For the purpose of a safe mechanism, it is required to fulfill functional and temporal transparency. Functional transparency ensures that the partitioning cannot be undermined by the processes. Temporal transparency is required to avoid unbounded influences on the process timing behaviour.

The monitoring task can either be implemented based on software instrumentation or by exploiting available hardware support. Software instrumentation would account for the resource usage on a basic block level. That is, at the beginning of each basic block the instrumented code increments a special purpose register, representing the resource counter, and compares its value against $\kappa_{\pi_i}^{\phi_k}$. Hardware-assisted implementations can for instance be based on performance counters, as proposed in [14]. Such counters are available in most modern architectures, e.g. in PowerPC [15], ARM [16] and x86 [17]. In addition to performance counters within the PEs, System-on-Chip (SoC)-based architectures usually provide platform-wide monitoring facilities with comparable properties, e.g. the ARM CoreSight architecture [18] or the Nexus [19] implementation within the Freescale P4080.

Once a capacity violation has been detected, the suspension task has to ensure that the respective process does not introduce further interference on the affected resource. Depending on the target hardware and resource it might be possible to avoid the usage of only the affected resource, but still allowing the process to use other resources. However, if this is not possible or if the respective resource is essential for the process to deliver its result, the process has to be suspended entirely.

Considering the required functional transparency we propose an implementation in the operating system layer. Naturally, this avoids bypassing either mechanism. With respect to temporal transparency and considering additional runtime overhead of software instrumentation, we favor hardware-assisted approaches to implement the monitoring. At runtime, the resource usage bounds $\kappa_{\pi_i}^{\phi_k}$ are used to configure a single monitoring counter per shared resource ϕ_k and process π_i. Once either of these counters detects a violation of $\kappa_{\pi_i}^{\phi_k}$ it raises an exception. In reaction, the corresponding Interrupt Service Routine (ISR) halts the core until the processes on the remaining cores have finished execution. Applying this policy to all processes ensures that none of them is influenced more than specified by the resource usage bounds $\kappa_{\pi_i}^{\phi_k}$.

Obviously, a valid configuration has to fulfill $\kappa_{\phi_k} \geq \sum_{i=0}^{|\Pi_{||}|-1} \kappa_{\pi_i}^{\phi_k}\ \forall k \in [0, |\Phi|)$, i.e. each resource ϕ_k has to provide sufficient capacity κ_{ϕ_k} to serve all in-parallel schedule processes $\pi_i \in \Pi_{||}$.

B. Multi-core WCET Analysis

WCET analysis is required to determine the execution time demand of each process. Single-core analysis is a well understood problem and even architectures with complex pipelines and caches can be analysed [20]. As explained earlier, the main issue for timing analysis of multi-core processors is the non-determinism in shared resource access latencies. Further, high variations between the best-case and the worst-case increase the severity of the problem and render naive approaches - which assume the worst-case for every access - infeasible to gain tight analysis results. The proposed approach extends existing single-core analysis techniques to additionally determine the *resource usage bound* (WCRA) per shared resource. Based on the single-core WCET and WCRAs we propose a novel phase, called the *interference-delay* analysis, to account for shared resource interferences.

1) Single-core Analysis: Well known approaches to single-core WCET analysis are end-to-end measurements, static analysis and measurement-based hybrid techniques. When applying end-to-end measurements, multiple executions of the whole process are performed, using their input data to trigger worst-case behaviour. Static analysis as well as hybrid methods apply a more or less standard architecture, consisting of control flow construction and analysis, micro-architecture analysis and path analysis [20]. First a control flow representation such as a Control Flow Graph (CFG) is constructed. Information on the control and data flow are used to annotate constraints to the CFG and identify infeasible paths. During micro-architecture analysis the basic blocks of the CFG are annotated with their maximum execution times. Finally, path analysis is required to compute the longest path through the CFG, solving an optimisation problem such as an Integer Linear Program (ILP). Only the micro-architecture analysis is performed differently for static and hybrid approaches. Static analysis applies a model of the target architecture to analyse e.g. pipeline and cache behaviour. Hybrid approaches utilise measurements on real hardware to acquire execution times for basic blocks. End-to-end measurements are not further discussed due to their similarities to hybrid approaches.

Due to their nature, the techniques significantly deviate with respect to the tightness of their results and the achieved assurance. In the context of certification, either of the approaches can be applied as long as the achieved assurance is suitable for the assurance requirements of the target application, cf. [21].

As already mentioned, in addition to the WCET also the WCRAs need to be computed. Due to their close relation it makes sense to apply similar techniques as for single-core WCET analysis. This needs modification of the micro-architecture and path analyses phases. To assign WCRAs per basic block, hybrid approaches can apply similar mechanisms as used for the described runtime monitoring. On the other hand, static analysis can for instance use the available cache hit/miss information to distinguish core-local and shared resource accesses. To finally compute the WCRAs over the whole process, an additional path analysis per shared resource is required. However, instead of optimising for the longest execution time, an optimisation problem for the maximum number of shared resource accesses must be formulated.

In essence, single-core analysis determines the WCET and WCRAs, while any of the discussed techniques can be applied, as long as the achieved assurance suits the target application. Further, since no information on in-parallel scheduled processes is required, the single-core analysis can be performed independently for each process.

2) Interference-Delay Analysis: The interference-delay analysis computes the maximum possible inter-process interference and the resulting increase in execution time. Therefor, the WCRAs $\kappa_{\pi_i}^{\phi_k}$ of all in-parallel scheduled processes are required, abstracting their specific behaviour. The interference-delay represents the maximum overlap of shared resource accesses, i.e. the combination of parallel and sequential resource accesses that leads to the longest possible execution time. It can be shown that the worst-case overlap occurs, if all requests are issued in parallel. According to the computed overlap, the individual accesses of each processes can be combined with the respective access latencies. The latency δ_n depends on the number of in-parallel issued accesses n and has to be safely determined for the target platform. Finally, the combination of single-core WCET τ_{π_i} and interference-delay constitutes the multi-core WCET τ_m of π_i, cf. Equation 1.

$$\tau_m(\pi_i) = \tau_{\pi_i} + \delta_{|\Pi_{||}|} \cdot \kappa_{\pi_0} + \sum_{i=1}^{x} \Big(\delta_{|\Pi_{||}|-i} \cdot (\kappa_{\pi_i} - \kappa_{\pi_{i-1}}) \Big) \quad (1)$$

C. Timing Composability

A system is composable with respect to a property, if that property remains valid on system level once it has been established on sub-system level [22]. For single-core WCET analysis timing composability is required for instance if features, such as pipeline and caches are analysed separately, while their results are added up to form the WCET. Accordingly, the proposed approach requires composability, since the final timing bound is the combination of the results of single-core and interference-delay analysis.

Unfortunately, so called timing anomalies violate the assumption of timing composability [23]. In effect, if an architecture suffers timing anomalies, it is not safe to rely on local worst-case decisions to compute the global worst-case. For instance, it is not safe to conservatively assume a cache miss, if a particular load/store instruction cannot be classified as either hit or miss. Instead, both paths need to be followed, which drastically increases the search space and hence analysis complexity. In [24], architectures are classified as fully timing compositional, compositional with bounded effects, and non-compositional based on the presence of timing anomalies.

In general, timing analysis requires a predictable architecture, cf. [25]. Fortunately, hardware manufacturers also provide multi-core platforms that were designed with analysability in mind, e.g. [26]. But, if an architecture does not fulfill the requirements, a deterministic configuration can help to improve its analysability. This for instance includes Least Recently Used (LRU) replacement strategies where possible, partial cache locking to emulate LRU replacement, separate instruction and data caches with write-through policy, caches used as scratchpad memories, cache partitioning on shared caches, static or disabled branch prediction, and Translation Lookaside Buffer (TLB) pre-loading.

Considering the proposed approach, the single-core analysis does not pose different assumptions or restrictions with respect to composability and determinism than existing approaches. On the other hand, the composition of single-core and interference-delay analyses requires composability and bounded latencies, i.e. the applied NoC protocol must not allow starvation. We argue that variations in NoC access latencies do not cause processor pipeline timing anomalies, since naturally even the best-case latency is orders of magnitudes higher than typical pipeline latencies. Thus, the processor's pipeline will drain in any case, while the access is processed. Hence, the interference delays do not effect pipeline analysis in the sense of timing anomalies.

IV. Implementation Aspects

A. Monitoring Facilities

As discussed in Section III-A, the monitoring can either be implemented based on available hardware support or by software instrumentation of the target application. We have specifically analysed the PowerPC architecture and especially the Freescale P4080 [27] multi-core SoC, with respect to the available features. We identified the e500mc [28] processing cores and the built-in SoC-wide debugging facility as promising implementation alternatives for non-intrusive monitoring.

The e500mc cores implement so called Performance Monitor Counters (PMCs). These are pipeline-independent counters which are able to trace certain events within the core. This, beneath others, includes stall cycles within the execution units, number of instruction fetches, miss predicted branches and caching related events. We identified the Bus Interface Unit (BIU) access event to be of special interest when monitoring the main memory accesses of an application. According to the manual and the discussions with the system vendor, the BIU event covers every access of the core to the shared interconnect and thus the main memory. This also includes accesses due to speculative execution. Beneath counting of events, the PMCs can be configured to trigger an exception once an overflow is detected. By setting the initial value of the counter appropriately, it is possible to precisely implement the described monitoring. Naturally the exception is handled by an ISR. Accordingly, the suspension routine has to be called subsequently.

While the e500mc PMCs provide a sufficient mechanism to cover the PEs, it is not possible to differentiate between accesses to the main memory and platform caches, thus accounting both with the same access latency. Further, the core PMCs cannot be used to monitor DMA-I/O devices. Hence, we further analysed SoC-wide monitoring facilities. The P4080 implements a Nexus-based [19] debug and performance monitoring architecture. Parts of this architecture provide similar mechanisms as the core PMCs, i.e. they are able to trace the NoC requests of connected devices and trigger a variety of events. For that purpose, they can be used to monitor DMA-I/O devices as well as cores. However, our analysis revealed that the implementation only implements a limited set of tracing units, such that it is not possible to monitor more than two devices within the system. Since the P4080 contains eight e500mc cores and a plenty of Peripheral Component Interconnect Express (PCI/PCIe), serial RapidIO (sRIO) and network interface units, we did not consider the debug architecture to monitor processing cores.

Since adequate hardware-assisted monitoring facilities are commonly available in modern processors [15], [16], [17], we did not investigate software instrumentation approaches and its definite overhead due to execution of additional instructions.

B. Operating System and Basic Software

According to the described temporal and functional transparency properties of the partitioning approach, we consider the operating system and basic software layer as most considerable for the implementation. Since software is commonly used as COTS product, the source code of the underlying operating system is not necessarily under control of the system integrator. For that purpose it is desirable to have an operating system independent implementation of monitoring and suspension. For that purpose, a custom driver module is considered. This module is responsible for configuring the monitoring facilities, e.g. the PMCs, and handle the related exceptions to suspend a process.

On the operating system side, the PMC exceptions need to be handled appropriately. For instance, if the BIU events are used to implement the monitoring, every bus access will cause the counter to increment. On an exception the appropriate exception type will be masked to prevent immediate re-interruption. Accordingly, the first-level ISR has to read the PMC value and re-configure the counter before enabling the exceptions again, in order to avoid an infinite loop of exceptions. If the source code of the operating system containing the ISR is not accessible different means for monitoring need to be used.

During execution of the ISR, the suspension routine needs to be executed. This routine can, for instance, be implemented as a callback handler within the monitoring device driver. The routine itself has to suspend the causing application. Again, this requires appropriate interfaces to the process management of the operating system. If this is not possible, the routine can be implement to postpone execution until the respective PE is allowed to again access the interconnect and main memory again.

V. Summary

In this paper we presented an integrated approach for temporal partitioning and WCET analysis. We discussed a novel approach to determine the maximum inter-process interference due to the use of shared resources, which is the main issue for multi-core timing analysis. Partitioning is used to bound the maximum resource usage per process. The approach allows the independent analysis of applications, which enables incremental development and certification, and software re-use as it is for instance required in the avionics industry. Furthermore, mixed-criticality workloads with arbitrary hard real-time applications are supported, which elevates its usability above approaches that only allow a single real-time application. Additionally, the restriction of resource privatisation is avoided, hence the benefits of multi-core architectures, such as parallel NoCs, can be fully utilised.

We discussed different ways to implement the proposed approach. This covered the monitoring as well as the analysis part, considering available monitoring facilities, operating system dependencies and state of the art analysis techniques, namely end-to-end measurements, static analysis, and hybrid techniques. The analysis of monitoring facilities of platforms revealed essential aspects that have to be considered during platform selection and monitoring implementation. Also the required operating system mechanisms influence the decision towards the target system. Further it has been discussed, that the modifications to the single-core analysis, as well as the computation of the inter-process interference can be implemented based on either of these techniques, showing the generality of the analysis part of the approach. Since predictability is a major concern for modern architectures, we discussed the requirements of composability. In essence, the approach requires a predictable architecture, similar to any other timing analysis techniques. Therefore, the single-core analysis phase does not pose additional assumptions or restrictions with respect to predictability and composability, compared to state-of-the-art techniques. We finally argued the absence of timing anomalies between NoC and pipeline analyses. The discussion also pointed out that the predictability of an architecture can be improved by configuration, if composability is not initially fulfilled.

We consider the discussed approach a promising alternative to related solutions. Even though some level of detail is lost by abstracting the architecture, we believe that reduced analysis complexity and the independent analysis of applications are overly dominant properties for the analysis of multi-core-based safety-critical real-time systems. Especially, since the isolation of applications is required by the targeted application domains. Further, the introduced runtime monitoring provides an additional safety layer to detect and handle faulty application behaviour, also considered as safety-nets [29].

VI. Acknowledgement

The ARAMiS project funds this work (German Federal Ministry for Education and Research, funding ID 01IS11035).

References

[1] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing predictability on multi-processor systems with shared resources," *Embedded Systems Week - Workshop on Reconciling Performance with Predictability*, 2009.

[2] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," *Proc. of Real-Time and Embedded Technology and Applications Symp.*, 2011.

[3] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on COTS hardware," *Proc. of Int. Conf. on Architecture of Computing Systems*, 2012.

[4] J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," *Proc. of Real-Time and Embedded Technology and Applications Symp.*, 2008.

[5] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," *Proc. of Real-Time Systems Symp.*, 2009.

[6] D. Hardy, T. Piquet, and I. Puaut, "Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches," *Proc. of Real-Time Systems Symp.*, 2009.

[7] S. Chattopadhyay, C. L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A unified WCET analysis framework for multi-core platforms," *Proc. of Real Time and Embedded Technology and Applications Symp.*, 2012.

[8] J. Rosen, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," *Proc. of Int. Real-Time Systems Symp.*, 2007.

[9] M. Paolieri, F. J. Cazorla, M. Valero, and G. Bernat, "Hardware support for WCET analysis of hard real-time multicore systems," *Proc. of Int. Symp. on Computer architecture*, 2009.

[10] O. Kotoba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multi-Core In Real-Time Systems Temporal Isolation Challenges Due To Shared Resources," *Proc. of Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems (at DATE Conf.)*, 2013.

[11] D. Dasari, B. Akesson, M. A. Awan, and S. M. Petters, "Identifying the sources of unpredictability in cots-based multicore systems," *Symp. on Industrial Embedded Systems*, 2013.

[12] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," *Proc. of European Dependable Computing Conf.*, 2012.

[13] J. Littlefield-Lawwill and L. Kinnan, "System considerations for robust time and space partitioning in integrated modular avionics," *Proc. of Digital Avionics Systems Conf.*, 2008.

[14] F. Bellosa, "Memory access - the third dimension of scheduling," University of Erlangen, Tech. Rep., 1997.

[15] IBM, *Power ISA*, 2010.

[16] ARM Ltd., "Cortex-A8 technical reference manual," 2010.

[17] Intel Corporation, "Intel 64 and IA-32 architectures software developers manual, part 2," 2013.

[18] ARM Ltd., "CoreSight components technical reference manual," 2009.

[19] IEEE-ISTO Nexus 5001, "The nexus 5001 forum standard for a global embedded processor debug interface," 1999.

[20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, and J. Staschulat, "The worst-case execution time problem overview of methods and survey of tools," *Transactions on Embedded Computing Systems*, 2008.

[21] RTCA, "DO-178C/ED-12C - software considerations in airborne systems and equipment certification," 2012.

[22] H. Kopetz, *Real Time Systems - Design Principles for Distributed Embedded Applications (page 34)*. Kluwer Academic Publisher, 1997.

[23] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," *Proc. of Real-Time Systems Symp.*, 1999.

[24] 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," *IEEE TCAD*, 2009.

[25] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, and B. T. R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," in *Proceedings of Embedded Real Time Software and Systems*, 2010.

[26] J. Harnisch, "Predictable hardware: The AURIX microcontroller familiy (presentation)," *Int. Workshop on Worst-Case Execution Time Analysis*, 2013.

[27] Freescale Semiconductor, *P4080 QorIQ Integrated Multicore Communication Processor Family Reference Manual*, 2012.

[28] ——, "e500mc core reference manual," 2011.

[29] B. Green, J. Marotta, B. Petre, K. Lillestolen, R. Spencer, N. Gupta, D. O'Leary, J. Lee, J. Strasburger, A. Nordsieck, B. Manners, and R. Mahapatra, "DOT/FAA/AR-11/2 - handbook of the selection and evaluation of microprocessors for airborne systems," Federal Aviation Administration (FAA), Tech. Rep., 2011.