Challenges in Future Avionic Systems on Multi-core Platforms

Andreas Löfwenmark
Avionics Equipment
Saab Aeronautics
Linköping, Sweden
andreas.lofwenmark@saabgroup.com

Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University
Linköping, Sweden
simin.nadjm-tehrani@liu.se

Abstract*—Modern avionic system development is undergoing a major transition, from federated systems to Integrated Modular Avionics (IMA) where several applications with mixed criticality will reside on the same platform. Moreover, there is a departure from today's single core computing, and we need to address the problem of how to guarantee determinism (in time and space) for application tasks running on multiple cores and interacting through shared memory. This paper summarizes the main challenges and briefly describes some active directions in research regarding temporal partitioning. It also outlines the forthcoming research that we will pursue for quantifying time bounds on memory access related interference, to ensure determinism and comply with certification requirements.

***Keywords*-real-time; mixed-criticality; multi-core; temporal partitioning; memory management; avionic systems**

I. Introduction

Multi-core platforms offer greater computational capacity with less size, weight and power (SWaP), and are used in diverse domains from mobile devices to supercomputing. However, safety-critical cyber-physical systems such as avionic and automotive systems have not (yet) embraced the technology. Aerospace systems are subject to costly and time-consuming certification processes, which require a predictable behavior under fault-free and certain hazardous conditions.

Predictability is also fundamental for establishing real-time correctness. In today's multi-core platforms, different cores share hardware resources such as caches and memory, which were essentially developed with a focus on maximizing the performance, but when placed in the safety-critical context introduce challenges to predictability. However, these challenges have to be met as aerospace is moving towards higher exploitation of commercial-off-the-shelf (COTS), as well as aiming to exploit the low SWaP characteristics of multi-core.

Classic worst-case execution time (WCET) estimates for a safety-critical avionics system are pessimistic even on a single-core platform, and our ambition is to find domain-specific techniques, so that dealing with the sources of non-determinism in multi-core does not cancel out the SWaP gains.

This short exposure of the above problems is organized as follows. Section II-III contains application-specific issues from the aerospace perspective. Section IV reviews several approaches towards solving some of the stated problems. Section V discusses open issues and outlines our current research.

II. Robust partitioning and Certification

Traditional avionics uses functions implemented and packaged as self-contained units. Integrated Modular Avionics (IMA) uses a high-integrity, partitioned environment, that hosts multiple functions of different criticalities on a shared computing platform. When moving from single-core to multi-core, several new problems affecting determinism and worst-case response time (WCRT) arise.

A safety-critical avionics system has to be certified by the authorities, e.g. the Federal Aviation Administration (FAA) in United States or the European Aviation Safety Agency (EASA) in Europe, covering both hardware and software. The standard RTCA/DO-254 (ED-80) [1] provides guidance for the development of airborne electronic hardware and the standard RTCA/DO-178C (ED-12C) [2] provides guidance for the development of airborne software.

For DO-254 hardware certification, it is possible to rely on service experience in comparable applications if no problems have been discovered in previous systems [3]. The problem for multi-core processors is that the implementation has very limited service history [4]. The software certification process does not change with the introduction of multi-core processors, but it becomes more complex due to a number of new sources of non-determinism. Sutterfield et al. [5] doubt that one single cohesive strategy to manage a DO-178C certification using multi-core processors exists.

Robust partitioning, to achieve fault containment, has traditionally been implemented in the federated architecture with dedicated hardware per application or function. With the introduction of IMA and multi-core, the robust partitioning property needs to be addressed and ensured [6].

According to Jean et al. [6] robust partitioning is (re)defined and refined in several documents which makes it complicated to extract the official definition. For example RTCA/DO-297 [7] defines it by stating that it "ensures that any hosted application or function has no unintended effect on other hosted applications or functions". Jean et al. refer to work by Rushby [8] defining the *Gold Standard for Partitioning*: "A robustly partitioned system ensures a fault containment level equivalent to its functionally equivalent federated system.". They also refer to work by Wilding et al. [9] defining the *Alternative Gold Standard for Partitioning*: "The behavior and performance of software in one partition must be unaffected by the software in other partitions" which is a stronger property and a sufficient condition to establish robust partitioning.

ARINC 653 [10] is a software specification for space and time partitioning, aiming at an alternative gold standard. It contains its own interpretation of robust partitioning: "The objective of Robust Partitioning is to provide the same level of functional isolation as a federated implementation."

The space partitioning concept can be implemented on multi-core systems as it is, as long as the hardware can prevent invalid memory accesses as required. The time partitioning concept, on the other hand, is problematic as ARINC 653 states that a partition should have exclusive access to its physical resources. Since the main memory could be seen as a shared resource (as we will see in section III), achieving time partitioning in multi-core systems is not trivial.

Huyck [11] examines the impacts on deploying ARINC 653 on a multi-core processor, in asymmetric multi-processing (AMP) or symmetric multi-processing (SMP) configurations. For an AMP configuration he identifies shared resource contention impacting the ARINC 653 partition and module-level health monitoring capability. In an SMP configuration, the ARINC 653 concept of concurrency is extended to allow multiple processes in a partition to execute in parallel on different cores. Huyck notes that preemption locking needs to be handled differently since stopping all other processes on other cores would reduce the usability of a multi-core system.

III. Shared Resources

In a single-core system only one task can access a resource at a time, but problems may arise in a multi-tasking system if the accesses from two tasks are interleaved, an issue that has been studied extensively. On a multi-core system the accesses can be issued at the same time and may have to be arbitrated by the hardware somehow. This arbitration which is not present on a single-core can have impact on the predictability. Next, four instances of resource sharing with impacts on predictability will be briefly reviewed.

Parts of the cache architecture may affect the interaction between cores, e.g. *cache sharing* [3], [4], *cache coherence* [3], and the *cache replacement policy* [12]. Cache sharing is present on platforms where the cores share one or more levels of cache. Cache coherence is needed for keeping shared data consistent among the core local caches and the main memory. The cache replacement policy affects how cache lines are replaced when the cache is full.

Four types of cache interference may arise due to cache sharing: a) intra-task interference, where a task evicts its own cache lines; b) inter-task interference, where a task evicts cache lines belonging to another task; c) cache pollution caused by asynchronous events such as interrupt service routines; and d) inter-core interference where tasks on different cores evict cache lines belonging to each other [13]. The first three are not new and exist in a multitasking single-core system as well, but the last one is new in multi-core systems. Another problem is that the cores may try to access the cache at the same time and one core will block the other core's access, which will introduce delays for the blocked core and non-determinism in the system. Kinnan [4] identifies the importance of using the L2 cache for performance in time and space partitioned Real-Time Operating Systems (RTOS) in IMA systems.

The cache coherency protocol is responsible for keeping the data in caches in sync with the main memory and each other. If two cores (C1 and C2) operate on independent data, which are stored in main memory in such a way that the data ends up in the same cache line, the cache coherency protocol will invalidate C1's cache, when C2 writes to its data. C1 will suffer a delay when it tries to access its data because the cache has to be reloaded. This is known as *false sharing* and has the same impact as if the cores were accessing the same data [3]. Thus, the interference caused by the coherency protocol can add to the non-determinism of the system.

With respect to the cache replacement policy, Agrou et al. [12] state that the Least Recently Used (LRU) policy is the most analyzable policy as this policy does not suffer from domino effects. Regardless of this, manufacturers replace the LRU policy with Pseudo-LRU (PLRU), which is a policy that almost always discards one of the least recently used items, or the FIFO policy. Both PLRU and FIFO are cheaper solutions in terms of implementation cost, but introduce domino effects affecting the WCET estimates.

A single memory controller for RAM is a shared resource just like the caches, and it has the same kind of issues. One core accessing the RAM can be blocked because of another core accessing the RAM at the same time. Kinnan [4] also states that this interaction may prevent hardware synchronization of processors as is typically done, which in turn can lead to schedule skid and jitter, resulting in non-determinism since synchronization then has to be performed in software instead of hardware.

Multi-core processors are often part of a System on Chip (SoC) together with peripherals such as external memory, serial I/O and Ethernet. To handle all accesses to the shared peripherals a coherency fabric is implemented to arbitrate the accesses. How the coherency fabric prioritizes the accesses is often part of the manufacturer's intellectual property. In a safety-critical system the configuration of the coherency fabric cannot be ignored since it can cause a core to be blocked by another core accessing some resource [4].

Agrou et al. [12] and Jean et al. [6] refer to the coherency fabric as interconnect and identify it as a crucial component of the multi-core processor – making the alternative gold standard difficult to realize if the hardware has not been developed for it. The behavior or performance in one partition may be affected by another partition, violating the alternative gold standard, which is incompatible with existing guidelines.

Hardware interrupts are also a potential source of interference between cores [3]. For example, if multiple devices are connected to the same interrupt which is routed to one core, but all devices are not exclusively used by that core, then the receiving core has to notify the other cores, and the other cores have to check the interrupt status for the devices handled by them.

IV. Alternative Approaches

A common means to achieving determinism on a multi-core system has been to disable all but one core, but this reduces the SWaP effect. The manufacturers improve throughput and reduce resource access interference by increasing the number of memory controllers [12], but this will not eliminate the problem unless enough memory controllers can handle accesses from all cores simultaneously.

How to use all cores in a safety-critical system is a research topic receiving a substantial amount of interest. The main approaches for solving the above problems, while not being overly pessimistic in WCET estimates and temporal partitioning analysis are: *resource management* e.g. dividing the cache between tasks to minimize the interference; *resource arbitration* where access to the resource is scheduled and the task gets private access for a period of time, which can be seen as a special case of resource management requiring special hardware; and *resource monitoring* where the resource usage is monitored and access can be restricted if a given limit is exceeded. We now focus on recent proposals to address resource management and resource monitoring with deterministic outcomes.

A. Resource Management

Some hardware platforms allow the accesses to shared resources to be scheduled using e.g. time division multiple access (TDMA). This way access to the shared resources can be arbitrated and the interference is eliminated. Schranzhofer et al. [14] analyze the worst-case response time (WCRT) for systems using TDMA to arbitrate the accesses to the shared resource. Three different access models are examined: dedicated, general and hybrid access. The authors propose an analytical worst-case analysis framework considering blocking/non-buffered access to a shared resource. They evaluate the framework for all three models, and show that separating computations and accesses to the shared resource is very important for the worst-case response time.

If no such hardware support is available another means of management is required. Giannopoulou et al. [15] use the dedicated access model from [14] and suggest a scheduling policy for a mixed-criticality multi-core system with resource sharing. This policy prevents timing interference among the tasks of different criticality levels by allowing only a statically known set of tasks with the same criticality level to execute on the cores at any time. A flexible time-triggered scheduling strategy is used on the core level and static and dynamic barriers are used for synchronization on the global level to achieve a more efficient resource utilization. This enforces timing isolation between criticality levels and enables composable and incremental certifiability. The cost is run-time overhead for the clock and barrier synchronization between the cores.

A similar approach is proposed by Anderson et al. [16] and further extended by Mollison et al. [17] who present a two-level hierarchical scheduling framework for mixed-criticality tasks. The top-level scheduler schedules container tasks, which in turn contain the "normal" tasks. Each container consists of tasks of the same criticality level and uses a given scheduling strategy.

In addition to task (CPU) scheduling, the partitioning of task data in the shared memory is also important. A memory organized in several banks enables tasks on different cores to access the memory in parallel without delaying each other provided that they access different banks. This is used by Giannopoulou et al. [18], where they propose memory mapping optimization to minimize the timing interference of tasks when accessing the memory. They also pinpoint that the memory mapping optimization is interdependent with the task scheduling and mapping used [15] and propose an integration of the two. An industrial application evaluation shows that memory mapping optimization can reduce the task response times.

The abovementioned works all assume each core has its own private local memory and Mollison et al. [17] do not consider caches. However, shared caches introduce a number of problems and are a major source of non-determinism. Ward et al. [19], following Anderson et al., Mollison et al. and Herman et al. [16], [17], [20] add cache management to make the shared cache more predictable and reduce WCET for high-criticality tasks. Page coloring is used to ensure that pages with different colors cannot cause cache conflicts. Since allocating the colors optimally is NP-hard in the strong sense, conflict-free color assignments may be impossible. To mitigate this they propose treating the colors as shared resources to which accesses must be arbitrated.

Mancuso et al. [13] address all four sources of cache sharing interference by proposing a two-stage solution. The first stage profiles the critical real-time tasks to determine the memory access patterns, and the second stage is a deterministic cache allocation strategy called "Colored Lockdown". It combines page coloring and cache locking. Page coloring is used to optimize the cache usage for frequently used memory pages and lockdown is used to override the cache replacement policy to ensure that the pages stay in the cache.

Another way to avoid the shared resource interference while still using multiple cores is described by Fuchsen [3]. An ARINC 653 compliant partitioning model is used where a safety-critical partition may be run on its own core and during that time window no other partitions are allowed to run on any of the other cores. This eliminates shared resource interference on the critical applications, but it is also quite expensive since many cores may be unused in several time windows.

As the number of cores in a processor increases, scaling the traditional memory hierarchy becomes a problem since caches consume a lot of power and space on the chip. A platform architecture without caches and only a local scratchpad memory (SPM) for each core is more power-efficient and more scalable [21]. In a software managed multi-core system the code and data have to be moved to the scratchpad memory from main memory using direct memory access before it can be executed, if all code and data fit in the SPM it is trivial to handle. However, if it doesn't fit it must be dynamically handled and this makes dynamic code management techniques a vital component.

Kim et al. [22] present two novel WCET-aware techniques for mapping data and code to the SPM (previous techniques have focused on the average-case execution time instead of WCET). They construct a variant of a control flow graph (CFG) called inline CFG which is used as input to the analysis. They perform analysis based on integer linear programming to find an optimal function-to-region mapping leading to the lowest WCET, but the technique is not scalable when the number of functions grows. A heuristic algorithm that is scalable but sub-optimal is therefore proposed.

B. *Resource Monitoring*

Resource monitoring can be used in two different ways: it can be used during development in order to estimate the parameters needed for WCRT estimation or interference bounding analysis. An example in our context is the work by Dasari et al. [23]. Monitoring can also be used at run-time to starve a task that demands more than the predefined share of resources during well-defined periods. We will refer to the latter as run-time monitoring.

For single-core systems based on ARINC 653 [10] the accesses to shared resources (basically only the CPU) are monitored by the RTOS which ensures that the statically defined partition execution schedule is followed.

The CPU monitoring concept of ARINC 653 is also usable for multi-core systems, but in addition to considering the CPU, the memory accesses should also be monitored. This can be implemented by using performance monitor counters found in most CPUs to track the consumed memory bandwidth and a scheduler (i.e. a run-time monitor) could use the performance monitor counters to prevent contention and to spread the memory accesses [24], thereby improving predictability.

A mix of the two monitoring approaches is found in the work by Yun et al. [25] who use performance counters to get information on memory accesses to separate real-time and non-real-time tasks. One core (critical core) contains the critical tasks and the other cores (interfering cores) contain non-critical tasks. The memory accesses are throttled on the interfering cores (via run-time monitoring) if they perform more memory requests than allocated to them.

Reserving one core for critical tasks may result in under-utilization if the critical tasks do not execute very often. Also, the approach is not suitable if there is more than one critical application. A more general approach is proposed by Inam et al. [26] where a multi-resource server (MRS) [27], [28] is presented. The server maintains two different budgets; one for the CPU usage and one for the number of memory requests, but the memory throttling is proposed per server instead of per core, and several servers can execute on a core. They show that the MRS can limit the interference between applications running on different cores, but also that cache pollution affects timing properties of tasks. Thus, MRS should be complemented with e.g. cache management.

Nowotsch et al. [29] propose a novel approach for integrating temporal partitioning and WCET analysis. Their proposed approach extends the existing single-core techniques with determining the maximum number of shared resource accesses per shared resource. Based on these estimates they introduce a new phase, the interference-delay analysis to account for the additional interference caused by shared resources. The temporal partitioning is split into a monitoring and a suspension task where the monitoring task uses performance counters to track the resource usage of each process, and the suspension task is invoked once a process exhausts its (preallocated) capacity.

V. Open Issues and Planned work

There are many challenges and open issues, but here we focus on a few that are relevant to our ongoing work.

Multi-core mixed-criticality systems need to add space partitioning to MRS and account for the additional budget needed for CPU and memory accesses. Hence, a question to address is: How do we account for the memory accesses made by the RTOS during e.g. a translation lookaside buffer (TLB) miss? This could affect the total number of memory accesses made by one core and could delay other cores more than anticipated. If the run-time monitoring function counts the memory accesses made by the RTOS, the tasks may get too little execution time due to reaching the access limit faster than calculated, which could lead to a deadline miss. On the other hand if the accesses are not accounted for by the (development time) monitoring function, the extra accesses to the memory may interfere with the other cores.

A deeper study is needed to investigate the relationship between server parameters such as server period, CPU budget and memory budget [26], [27], and to determine if the RTOS accesses should be included in the memory budget or not.

As avionics systems often operate on high altitude they are exposed to cosmic radiation, which could introduce faults in the system. Some of the effects can be mitigated by the use of error detection and correction for e.g. memory, but this will impact the access times. A cache that is able to detect single-bit errors may invalidate the cache, thus affecting the number of memory accesses performed by the task and the execution time.

Very few experiments have been performed on an actual avionics RTOS with a relevant workload, and therefore a still open question is to study alternative implementations, both for better performance and for higher relevance to the avionics industry.

We plan to extend the memory budget in an MRS-based approach by also accounting for the number of accesses caused by RTOS TLB miss handling, modeling the budget as $M_T = M_S + M_{OS}$, where M_S is the memory access budget for all tasks contained in the server during its period, and M_{OS} is the budget for the memory management of the RTOS during that time. We will assume M_S is given and focus on M_{OS}. To estimate M_{OS} we will model the memory management unit (MMU) to get an estimate of the number of TLB misses during a server period (TLB_S). From this we get $M_{OS} = TLB_S \cdot M_{TLB}$ where M_{TLB} is the number of memory accesses for each TLB miss. The upper bound of TLB_S is when each of the memory accesses up to the limit (M_S) results in a TLB miss, i.e. $M_{OS} \leq M_S \cdot M_{TLB} \Rightarrow M_T \leq M_S \cdot (1 + M_{TLB})$, but using the MMU model and memory usage information for the tasks we seek to lower that bound. The approach will then be evaluated with a prototype extension to an available RTOS.

References

[1] RTCA, Inc, "RTCA/DO-254, design assurance guidelines for airborne electronic hardware," 2000.

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

[3] R. Fuchsen, "How to address certification for multi-core based IMA platforms: Current status and potential solutions," in *Proceedings of the 29th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2010.

[4] L. Kinnan, "Use of multicore processors in avionics systems and its potential impact on implementation and certification," in *Proceedings of the 28th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2009.

[5] B. Sutterfield, J. Hoschette, and P. Anton, "Future integrated modular avionics for jet fighter mission computers," in *Proceedings of the 27th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2008.

[6] X. Jean, D. Faura, M. Gatti, L. Pautet, and T. Robert, "Ensuring robust partitioning in multicore platforms for IMA systems," in *Proceedings of the 31st IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2012.

[7] RTCA, Inc, "RTCA/DO-297, integrated modular avionics (IMA) development, guidance and certification considerations," 2005.

[8] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," 1999.

[9] M. M. Wilding, D. S. Hardin, D. A. Greve, and R. C. Inc, "Invariant performance: A statement of task isolation useful for embedded application integration," in *In Dependable Computing for Critical Applications, DCCA-7*, 1999, pp. 287–300.

[10] Aeronautical Radio Inc (ARINC), "ARINC 653: Avionics application software standard interface part 1 - required services," 2010.

[11] P. Huyck, "ARINC 653 and multi-core microprocessors - considerations and potential impacts," in *Proceedings of the 31st IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2012.

[12] H. Agrou, P. Sainrat, M. Gatti, and P. Toillon, "Mastering the behavior of multi-core systems to match avionics requirements," in *Proceedings of the 31st IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct 2012.

[13] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2013.

[14] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2010.

[15] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, Sept 2013.

[16] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, April 2009.

[17] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Proceedings of the 10th International Conference on Computer and Information Technology (CIT)*, June 2010.

[18] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Mapping mixed-criticality applications on multi-core architectures," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2014.

[19] B. Ward, J. Herman, C. Kenna, and J. Anderson, "Making shared caches more predictable on multicore platforms," in *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2013.

[20] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson, "RTOS support for multicore mixed-criticality systems," in *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2012.

[21] J. Lu, K. Bai, and A. Shrivastava, "SSDM: Smart stack data management for software managed multicores (SMMs)," in *Proceedings of the 50th ACM / EDAC / IEEE Design Automation Conference (DAC)*, May 2013.

[22] Y. Kim, D. Broman, J. Cai, and A. Shrivastava, "WCET-aware dynamic code management on scratchpads for software-managed multicores," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, April 2014. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/1049.html

[23] D. Dasari, B. Andersson, V. Nelis, S. Petters, A. Easwaran, and J. Lee, "Response time analysis of COTS-based multicores considering the contention on the shared memory bus," in *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Nov 2011.

[24] R. Inam, M. Sjödin, and M. Jagemar, "Bandwidth measurement using performance counters for predictable multicore software," in *Proceedings of the 17th IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2012.

[25] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012.

[26] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin, "The multi-resource server for predictable execution on multi-core platforms," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, April 2014. [Online]. Available: http://www.es.mdh.se/publications/3482-

[27] M. Behnam, R. Inam, T. Nolte, and M. Sjödin, "Multi-core composability in the face of memory-bus contention," *SIGBED Rev.*, vol. 10, no. 3, pp. 35–42, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2544350.2544354

[28] R. Inam, J. Slatman, M. Behnam, M. Sjödin, and T. Nolte, "Towards implementing multi-resource server on multi-core linux platform," in *Proceedings of the 18th IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2013.

[29] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht, "Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2014.