Comparably Evaluating Communication Performance within Mixed-Criticality Systems

Keegan Napier, Oliver Horst, Christian Prehofer

Comparably Evaluating Communication Performance within Mixed-Criticality Systems

Keegan Napier*†, Oliver Horst†, Christian Prehofer†
* La Trobe University, Australia. kcnapier@students.latrobe.edu.au
†fortiss GmbH, Munich, Germany. {napier, horst, prehofer}@fortiss.org

***Abstract*— In many mixed-criticality systems, tasks of differing criticality levels execute upon a common computing platform. Within such systems, it is imperative that communication between tasks, and between tasks and shared computing resources, do not impede upon the correct functionality of highly critical tasks. To evaluate the performance of communication mechanisms that offer this functionality, a number of attributes of the mechanism may be assessed. Currently, there exists no agreed upon method of evaluating such mechanisms, and so comparing the performance of differently developed mechanisms is difficult. Within this paper, we identify attributes that characterise the performance of mixed-criticality communication, offer suggestion as to how they may be comparably evaluated, and outline an evaluation framework that would simplify the process of comparably evaluating a communication mechanism with these scenarios.**

I. Introduction

To further reduce the space, weight and power consumption of embedded and real time systems, there is a trend towards integrating a number of components with differing criticality levels onto a single, common hardware platform. Inherently, tasks with a higher criticality level are accordingly more instrumental in the correct and safe operation of the system, and so there must be a higher degree of assurance that such tasks are not susceptible to failure. A system that has two or more distinct criticality levels is referred to as a mixed-criticality system (MCS) [1].

As the complexity of mixed-criticality systems increase, there is a rising interest in moving from single core processor architectures to more capable multicore and manycore architectures. Making this migration, however, introduces the issue of sharing resources, such as the system bus, memory bus, memories and caches, between tasks of multiple different criticality levels. It is necessary that shared resource accesses are managed in such a way that tasks on one core are not capable of excessively delaying tasks on other cores that are trying to access the same resource [2].

Within embedded systems, tasks will require the ability to communicate with memories and each other through the use of these shared resources. Therefore, communication must be managed so that it does not pose a significant risk of causing critical task failure. Should communication expose a critical task to an unexpected communication-related execution delay, it presents the risk that the task will not complete execution before its hard deadline. Similarly, should the communication within the system be capable of affecting the functional behaviour of the critical task in an undesirable way, this presents the risk that the critical task will malfunction. It is therefore essential that both temporal and functional properties of critical tasks are isolated from unexpected communication related faults and interference.

Providing isolation of critical task execution from communication within the mixed-criticality system, however, is a complicated problem with a number of factors, e.g. keeping critical communication predictable without detracting too much from performance, that must be considered when developing a suitable approach. Inherently, this has lead to a wide variety of approaches to mixed-criticality communication being developed, each of them aiming to take as many of these factors into account as possible [3]. Through analysis of these approaches, it can be construed that there exist a range of properties that can be evaluated to demonstrate how well the communication protocol is limiting the impact of these factors.

As there currently exists, to the best of our knowledge, no standardised method of evaluating communication mechanisms within mixed-criticality systems, generally only properties that demonstrate specific results of the designed mechanism are evaluated, while others are overlooked. This has the added effect of allowing for these properties to be evaluated in ways that would produce results that are not directly comparable with another approach that is evaluating the same property.

In this paper, we focus upon improving the evaluation methods that researchers are utilising to evaluate their mixed-criticality communication mechanisms. We investigate how evaluations are currently being conducted, and use these investigations to suggest an empirical evaluation framework that would enable comparable evaluations to be made. Our contributions can be summarised as follows:

- We analyse a range of evaluations for current mixed-criticality communication mechanisms. We use this analysis to produce a set of general properties that indicate the performance of a communication mechanism within mixed-criticality systems.
- We promote the need to evaluate each of these general properties in a more comparable manner, and propose the use of scenario-based evaluation to achieve this.
- We propose a framework that would enable a researcher to develop evaluation scenarios for their system in a way that the same scenario may be executed upon a separate system.

The remainder of the paper will be organised as follows. Within Section II, we highlight properties that researchers commonly demonstrate to be of importance when evaluating communication within their system. We then suggest execution scenarios that would enable the measurement of these communication properties to be stressed in Section III, and propose an evaluation framework that simplifies the design and implementation of these scenarios in Section IV. Related work is detailed in Section V.

II. Important Communication Properties

Through the analysis of a diverse range of approaches to design and implement communication mechanisms within mixed-criticality systems, we identified a range of properties that are important to consider when evaluating such a mechanism. Within this section, we outline each of these properties with reference to research, and give an overview as to why they are important. For clarity, we have sorted each of these properties into the following categories: Predictability and Separation, Performance, and Scalability.

A. Predictability and Separation

It is a persistent trend throughout mixed-criticality systems that communication, especially critical communication, must behave in a predictable manner. Furthering this, the execution of tasks within the system should be separated from communication, i.e., communication within the system must not be capable of altering task execution in unexpected ways. To ensure an implemented communication mechanism offers suitable levels of predictability and separation, a number of properties should be present within the mechanism. These properties are discussed with reference to the field of research below:

1) Temporal Deadline adherence: First and foremost, critical transmission and resource access times must adhere to strict real time temporal deadlines, to avoid delaying the execution of critical tasks by an unexpected amount of time. Typically, these deadlines relate to the worst case scenario that would cause the greatest delays for communication, i.e., the worst case transmission time. It is therefore essential that as a basis for evaluating any communication mechanism, that it can be assured that critical temporal deadlines are consistently met.

2) Shared Resource Interference Management: When considering multi core architectures, communication delays fundamentally arise due to contention of accessing micro-architectural resources, e.g., system bus, shared memory. Should a task experience interference when communicating with any of these resources, the overall execution time of the task will increase. [4], [5]. Calculating the worst case execution time of a critical task must take into account all of the time that the task may potentially spend waiting due to interference. Nowotsch et al. [6] emphasise this requirement through their explicit requirement of incorporating all potential communication interference into their calculation of the worst case execution times. For this reason, interference related communication delays must be predictable, to ensure that the worst case execution time does not have to be overestimated.

3) Predictable Critical Communication: In addition to ensuring that interference related communication delays should be predictable, critical tasks depend on being delayed due to communication for a predictable amount of time. Cilku et al. [7][8] suggest a dual-layer arbitration of the system bus, where highly critical tasks utilise a more predictable TDMA access scheme to access the shared system bus, while less critical tasks access the bus via a less predictable round-robin method. To stress and evaluate how predictably critical tasks are able to access the shared system bus, they have a single dummy task executing on each core of a dual-core processor. Each of these tasks make continuous accesses to shared memory, and they measure the standard deviation in the average and maximum execution times of the tasks as they alter how much interference their accesses cause with one another. Their evaluation shows the interest in determining how predictable communication related delays will remain while exposed to varying levels of interference.

4) Temporal Isolation: Delays that critical tasks face due to communication related should be temporally isolated from unexpected external interference. Researchers in the field have previously empirically demonstrated that their communication mechanism will consistently meet deadlines under unexpected circumstances by exposing their system to a variety of situations that stress the communication of the system in different unpredictable ways. Brandenburg [9], for example, demonstrates that critical tasks utilising his shared resource access scheme will reliably gain access to shared resources before a calculated worst case access time, regardless of potential temporal disturbances caused by other tasks on the system. He does this by artificially causing other tasks to act in a faulty manner, and create a higher level of contention for shared resources. The faults that are introduced all aim to have an effect upon the access times of a particular critical task, including causing non-critical and critical tasks to make more frequent requests to access a shared resource, and executing a higher number of non-critical tasks upon each of the cores.

5) Functional Isolation: Communication should not be capable of causing an executing task to malfunction. As there are a range of ways that communication can influence Pellizzoni et al. [10], for example, suggest that to functionally isolate task execution from communication, a set of formal properties that define the permissable communication behaviour of tasks during runtime must be specified, and all communication should be monitored during run time to ensure that no properties are violated. Through monitoring each of these properties during runtime, they are able to reject any faulty communication before it propogates through the system, and take any recovery actions if necessary.

Through analysis of each of these five key attributes related to providing predictability and separation within a communication mechanism, the important properties can be summarised to the following points:

- Critical communication must fundamentally meet all real time communication related temporal deadlines.

- Communication interference should be handled in a way that critical task execution is affected predictably.
- Even under varying external disturbances, communication related delays must remain predictable.
- Task execution behaviour should be functionally and temorally separated from the communication mechanism.

B. Performance

One of the main research challenges regarding designing a communication mechanism for use within mixed-criticality systems, is keeping performance overheads to a minimum, while still providing the aforementioned predictable properties. Areas of focus within this field aim to reduce the overhead that communication in the system puts upon task execution times, both critical and non-critical. A general overview of each of the properties that are considered in literature when determining the performance of a communication mechanism are outlined below:

1) Communication Medium Utilisation: One of the more predominant communication-related performance overhead is due to arbitrating the communication medium in a way that hinders task execution times. A variety of approaches have therefore been explored that aim to improve the overall utilisation of the communication medium to increase performance. Cilku et al. [7][8] demonstrate that their method of dual-layer arbitration offers better performance for non-critical communication by showing that when they execute a bubble sort algorithm upon each core of a quad-core processor, with two of the cores being treated as critical and the other two non-critical. They show that the average execution time of the bubble sort algorithm upon the non-critical cores utilising their method of bus arbitration is 3.1 times faster than if all cores were accessing the bus via a TDMA access scheme. Other approaches aim to offer non-critical tasks freedom to communicate, provided that the communication interference they cause for critical tasks can be bounded. Yun et al. [11] have implemented a memory access throttling algorithm that will limit non-critical tasks to a certain number of cache misses before their access to shared memory is throttled. They evaluate the performance that this throttling scheme offers non-critical tasks by playing back high definition video on two of the cores of a quad-core processor and determining how long each of the cores were throttled throughout execution.

2) Influence of critical communication upon non-critical communication: Non-critical tasks do not require strict assurance that they will meet their execution deadlines, and so non-critical communication does not require the same level of predictability that critical communication does. The objective of providing non-critical communication is to allow non-critical tasks to utilise a platform's communication media as efficiently as possible, while not causing critical communication to become unpredictable. This property was explored by Kim et al. [12], who propose a direct memory access (DMA) scheme that offers tight bounding for critical task accesses, and better access times for non-critical tasks. To evaluate the performance benefits they give to non-critical tasks, they simulate system load by executing a number of standard embedded benchmarking programs that put differing levels of strain upon the memory. They assign programs from the Mälardalen WCET benchmark suite[1] as critical tasks, and programs from the Mibench embedded benchmarking suite[2] as non-critical tasks. For each execution run, they introduce an increasing number of critical tasks to the system, and measure how this affects the average memory access read times for the non-critical tasks. They then compare their results with average read times from a more predictable time division multiplexed (TDM) approach to show that their method offers lower average read times for non-critical tasks.

From analysis of these properties, it can be ascertained that when determining the performance of a communication mechanism, the important properties to be evaluated are:

- The communication medium should utilise the communication medium as effectively as possible, to reduce communication overheads.
- Non-critical communication should be implemented in such a way that it does not need to adhere to the same strict requirements of critical communication

C. Scalability

Due to the increase in complexity of mixed-criticality systems, one important property of a communication mechanism is how well it scales with a system that grows in complexity. An overview of properties that should be assessed with regards to scalability are detailed below.

1) Resource Arbitration Scaling: As an increasing number of tasks are added to the system, providing predictable shared resource interference can cause an increasing amount of execution overhead for tasks executing upon the system. Kelter et al. [13] highlight that for shared resource access arbitration within real time systems, the effect that the method of providing predictable interference has upon the real time execution deadline must be taken into consideration. They execute a variety of multi core embedded real-time benchmarking programs upon a single multi core ARM architecture and evaluate how each of a given set of resource arbitration methods perform as they are scaled to a higher number of cores. They demonstrate that shared resource access arbitration methods that offer higher average bus utilisation give a better average case execution performance as the number of cores within the system scale. However, they also show that these methods that offer the highest bus utilisation and lowest average case execution times often have a larger maximum overestimate on the calculated WCET of the executed tasks.

2) Multi/Many Core Scalability: As mixed-criticality systems are expanded to be executed upon multi/many core architectures, determining how well a communication mechanism scales when the number of processing cores that the system is executing upon increases is important. Sigrist et al. [14], for example, demonstrates how the overhead of differing mixed-criticality scheduling algorithms affects the range of task sets

[1] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
[2] http://www.eecs.umich.edu/mibench/

that they can assuredly schedule so that all tasks will meet their deadlines. As scaling to a larger number of cores enables more tasks to execute in parallel, this causes there to be more contention for shared communication resources. Therefore, when assessing the multi/many core scalability of a mixed-criticality communication algorithm, the focus is upon how this additional contention affects average case and worst case critical communication times.

There are two important aspects of a mixed-criticality communication mechanism that should be evaluated when determining how scalable it is:

- The employed shared resource arbitration method should utilize the communication medium well, even if the number of tasks executing upon the system grows.
- The increase in average and worst case execution times of tasks upon the system as the number of tasks executing in parallel should be assessed.

III. Evaluating Important Properties

As it was outlined within Section II, there are a number of potential properties that can be evaluated when assessing the performance of the communication within a mixed-criticality system. There currently exists no standardised method of evaluating mixed-criticality communication performance, which has lead to a wide variation in how researchers are evaluating the performance of their systems. Due to the wide variation of attributes that are available to a researcher when evaluating the communication within their system, it is quite common that a researcher will:

- Evaluate only specific properties that their particular communication method excels in and disregard others,
- And/or evaluate these properties in a way that differs to how other research has evaluated these attributes.

Within this section, we discuss possible evaluation strategies for the properties presented in Section II. For each of these properties, we suggest potential scenarios that would strain the system in a way that each property can be accurately measured. We are focussing exclusively on an empiral measurement of the properties through a common evaluation framework, presented in Section IV. We favor this approach as it is easy applicable to new systems and thus suport our comparable evaluation goal. However, it should be noted that in certain cases an analytic approach might be favourable to exactly evaluate a property.

A. Predictability and Separation

1) Temporal Deadline Adherence: As it is imperative that critical tasks do not miss execution deadlines, throughout the entire evaluation process, there should be no critical communication related deadlines missed. As demonstrated by Brandenburg [9] and Bate et al. [15], exposing a mixed-criticality system to a variety of different communication composure is one method of assessing whether all deadlines will be met in a range of circumstances. To alter the communication composure, the ratio of critical to non-critical tasks can be changed, and the frequency with which each of these tasks are attempting communication upon the system. It should be mentioned however, that should no critical deadlines be missed throughout any of the evaluation scenarios, this will not be enough to formally prove that critical communication related deadlines will not be missed under any circumstances. Analytic approaches could help here to provide formal proofs that critical deadlines will not be missed.

2) Shared Resource Interference Management: To assess how predictably and efficiently communication interference will be handled, there are three properties of interest, especially if communication interference within the system rises:

- The worst experienced critical communication latency
- The average critical communication latency
- The standard deviation of critical communication latencies

To reliably increase the amount of interference that critical tasks within the system are experiencing, we propose the following scenario: Separate tasks are executing upon differing cores of a platform, with each communicating a payload at specific time intervals (similar to [7]). Gradually, in successive runs of the scenario, the timing that the payloads are transmitted will overlap to produce an increasing amount of interference.

3) Temporal Isolation: Through observation of research, highlighting that critical communication cannot be temporally affected by external factors is not commonly done using empirical results alone. Brandenburg [9], for example, demonstrates temporal isolation through a combination of mathematical proofs demonstrating that accesses from other tasks in the system can only affect critical tasks to a certain extent; and through runtime evaluation, by demonstrating that should other tasks upon the system start communicating in a faulty manner, critical tasks communication latencies will remain unaffected. Taking from this example, developing a scenario that demonstrates that critical communication is temporally isolated is a challenge. To measure this, tasks within the scenario, both critical and non-critical, would need to begin communicating in a faulty manner to demonstrate that critical tasks are continuing to meet their worst case execution deadlines. To guarantee that task execution is temporally isolated, a combination of an empirical and analytical approach should be employed.

4) Functional Isolation: Similarly to temporal isolation, functional isolation is not easily guaranteed through runtime analysis alone as there are a wide range of ways in which functional isolation may be breached (tasks sending false messages, shared resources being left in an unsynchronised state etc.). Pellizzoni et al. [10], for example, describes and models all acceptable communication behaviour within the system to ensure that task execution will not affect any tasks in the system in a detrimental manner. They do not, however, reinforce these claims with an empirical evaluation. Beside utilizing scenarios to determine whether critical communication within the system will be functionally isolated, we also think of questionnaire or analytic approaches that aim to determine whether the communication is suitably isolated.

B. Performance

1) Impact Upon Critical Communication: Measuring the impact that non-critical communication has upon critical communication can be achieved by introducing a higher ratio of non-critical to critical traffic and observing how this affects the average case communication related latency that critical tasks experience. A scenario that would offer suitable testing conditions for this consists of a set of critical tasks, and an increasing number of non-critical tasks. The communication latency that critical tasks experience can be measured after each addition of a non-critical task to determine whether non-critical traffic is having an adverse effect.

2) Communication Delay: Non-critical communication typically will experience delays within the system due to giving priority to critical communication. To determine the delays that are caused due to critical communication, the change in non-critical communication related latencies are measured as an increasing amount of critical traffic is introduced to the system. To measure this, similarly to what was described within [12], a scenario that contains a set number of non-critical tasks and an increasing number of critical tasks can be utilised, and the effect that the increasing critical traffic has upon the non-critical communication delays can be observed.

C. Scalability

1) Scalability on Multi/Many Core Architectures: To evaluate how well a mixed-criticality algorithm scales to a higher number of cores, there are three key properties to assess as the system is expanded to cover a wider number of cores: The overestimation in the calculated worst case execution time for critical tasks when compared to the observed worst case execution time, the average execution time of critical tasks, and how effectively the communication medium is utilised throughout the evaluation period.

To provide a scenario that provides a common means of evaluating scalability, we adapt the evaluation approach utilised within [13], which was designed for assessing the scalability of single criticality real time resource arbitration schemes. The scenario will consist of a specific set of standard benchmarking tasks taken from common embedded benchmarking suites to provide the system with a realistic workload. Each of these tasks are divided into critical and non-critical tasks depending on whether the work within the benchmarking task is deemed to be of a critical nature.

To perform an evaluation of scalability using this scenario, the task set is first executed upon a single core, to determine their average case execution time without communication interference coming from tasks running in parallel. The scenario will then be performed further times, with the task set being divided across 2, 4 and 8 cores in subsequent runs to determine how the average case execution time of each critical task differs as more parallelism is introduced to the system.

IV. Evaluation Framework

In this section, we present a concept for an evaluation framework that will enable a researcher to design and reuse prebuilt evaluation scenarios for evaluating their specific communication mechanism. Each of the components that the framework is comprised of are described, focusing upon how each aims to enable reproduceable evaluation scenarios to be constructed. Figure 1 shows a block diagram overview of components that the framework is composed of, and a typical workflow that would enable a user to utilise a scenario to evaluate a communication property.

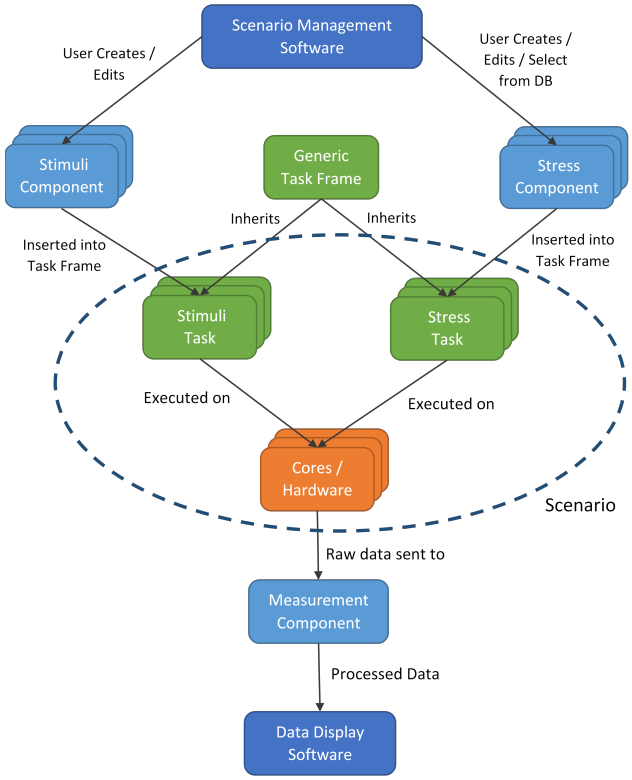

Figure 1. Block overview of Evaluation Framework

Scenarios will be created or added using the included scenario management software. This scenario management software will enable a user to construct scenarios from premade components, or import a scenario that another user has previously utilised in their evaluation. Each scenario that is created will consist of a set of *stimuli tasks* and *stress tasks*. Each of these tasks will have a different purpose throughout the evaluation process. Stimuli tasks will be specialised tasks, created by the user within the scenario management software, that will be utilised to stimulate the communication within the system in a particular way. For example, should the scenario require a task to flood the communication media with requests, a stimuli can be created that exhibits this behaviour during runtime. Tracepoints will automatically be inserted into the stimuli task upon creation, which will trace important communication information throughout execution, allowing for a specific result to be compiled by the measurement component. Stress tasks will be more standard tasks that can be inserted into the scenario to simulate a realistic system load. Each stress task

will run a single program, for instance, a program taken from an embedded benchmarking suite. Multiple stress components can be integrated into a single scenario, allowing for distinctive and complex system loads to be created. The collection of tracepoint data will be transmitted to an external computing source, which runs the included measurement software. The measurement software is responsible for analysing tracepoints and presenting the analysed data to the user in a usable way.

We have currently implemented a prototypic version of the framework upon an Intel® Core 2 Duo platform. We are looking, however, to extend this implementation to an 8 core ARMv8 architecture to enable us to demonstrate a wider range of scenarios with the framework, and evaluate the framework's performance. Currently, we are able to create and insert stress and stimuli components into a scenario to be executed, and then trace basic communication properties of this scenario. These tracepoints are processed and output as both a formatted CSV file, and PGF plot. The framework is abstracts from the communication mechanism so that only a few C functions are required to be updated by the user to port the framework to their specific communication mechanism.

V. Related Work

Although research into the field of providing communication for mixed-criticality systems has generated a range of different approaches, few have utilised or devised comparative evaluation methods. There are however previously utilised scenario-based methods of evaluation to commonly evaluate the properties of mixed-criticality scheduling schemes.

Harbin et al. [16] have designed a singular scenario based upon industry requirements to be used as a benchmark for stressing and evaluating the communication related performance of real-time mixed-criticality networks-on-chip. They thoroughly describe the benchmark that they have used, however, a single scenario does not give an evaluator the flexibility to demonstrate particular properties of their communication.

Bate et al. [15] have utilised a scenario based evaluation method to comparably evaluate the performance of a variety of mixed-criticality scheduling schemes. They generate scenarios that simulate realistic workloads for the system and measure an array of important properties relating to the scheduling scheme. As their scenarios are generated, however, a truly comparable result could only be made should the evaluator have access to the exact scenario that they were utilising.

Sigrist et al. [14] similarly utilises a set of generated scenarios that cause the system to experience a workload with a normalised level of utilisation to evaluate a set of mixed-criticality scheduling schemes. Throughout the execution of each of these scenarios, they measure overheads produced by each scheduling scheme. For a separate researcher to evaluate their scheduling method comparably against their results, they would need to again, have access to the generated scenarios that they utilised within their evaluation.

VI. Conclusion

To summarise the contributions of this paper, we have identified a range of attributes that may be utilised to indicate the performance of communication within mixed-criticality systems, outlined how each of these attributes may be evaluated through the use of scenarios, and portrayed a more simplified method of evaluating using scenarios through the construction of an evaluation framework.

For future work, our aim is to make each of the scenarios more concise, and to implement the evaluation framework so that it may be used across a range of platforms.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n° 645119.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007.

[2] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010.

[3] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[4] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems–temporal isolation challenges due to shared resources," in *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2014.

[5] D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke, "Temporal isolation on multiprocessing architectures," in *Proceedings of the 48th Design Automation Conference*. ACM, 2011.

[6] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*. IEEE, 2014.

[7] B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro, "A memory arbitration scheme for mixed-criticality multicore platforms," in *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, 2014.

[8] ——, "A TDMA-based arbitration scheme for mixed-criticality multi-core platforms," in *Event-based Control, Communication, and Signal Processing (EBCCSP), 2015 International Conference on*. IEEE, 2015.

[9] B. B. Brandenburg, "A synchronous IPC protocol for predictable access to shared resources in mixed-criticality systems," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*. IEEE, 2014.

[10] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha, "Handling mixed-criticality in SoC-based real-time embedded systems," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009.

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

[12] H. Kim, D. Broman, E. A. Lee, M. Zimmer, A. Shrivastava, and J. Oh, "A predictable and command-level priority-based DRAM controller for mixed-criticality systems," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2015.

[13] T. Kelter, T. Harde, P. Marwedel, and H. Falk, "Evaluation of resource arbitration methods for multi-core real-time systems," in *OASIcs-OpenAccess Series in Informatics*, vol. 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[14] L. Sigrist, G. Giannopoulou, P. Huang, A. Gomez, and L. Thiele, "Mixed-criticality runtime mechanisms and evaluation on multicores," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015.

[15] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015.

[16] J. Harbin, T. Fleming, L. S. Indrusiak, and A. Burns, "GMCB: An industrial benchmark for use in real-time mixed-criticality networks-on-chip," *Proc. WATERS, 27th ECRTS*, 2015.