How to Build a Mixed-Criticality System in Industry?
— From the perspective of system architecture

Double Blind

***Abstract*—In the last decade, the rapid evolution of diverse functionalities and execution platform led safety-critical systems towards integrating components/functions/applications with different 'criticality' in a shared hardware platform, i.e., Mixed-Criticality Systems (MCS)s. In academia, hundreds of publications has been proposed upon a commonly used model, i.e., Vestal's model. Even so, because of the mismatched concepts between academia and industry, current academic models can not be exported to a real industrial system. This paper discusses the mismatched concepts from the system architecture perspective, with a potential solution being proposed.**

I. Introduction

Safety-critical systems play an important role in many medical and industrial fields [33]. In safety-critical systems, integrating components with different levels of criticality (e.g., Automotive Safety and Integrity Levels (*ASILs*) in ISO 26262 [19].) onto a shared hardware platform has become increasingly important [13]. This results from the diverse functionalities required by modern safety-critical systems (e.g., automated driving [27]), and the rapid evolution of execution platform [32]. Such systems are called *Mixed-Criticality Systems (MCS)s* [6], [9].

Nowadays, the popularity of MCS has been raised to an unexpected height in both academia and industry. In academia, almost 300 papers related to MCS have been published in the recent decade, and tens of related papers are still published each year [9]. In industry, requirements regarding 'integrating applications with different criticality levels in a shared platform[1]' have been added in the almost all safety-related standards [15], e.g., DO-178C, EN50128, ISO26262, etc..

However, numerous unsolved challenges lead to a dilemma of building MCS practically [9], [15], e.g., optimization of system model, effectiveness of task allocation and resource management etc.. From the experience of the machine learning community, a unified direction between academia and industry can significantly accelerate the developments of the whole area. However, the mismatches between academia and industry are currently slowing down the development of MCS. Some mismatches have been already recognized by different researchers and engineers. For instance, Graydon and Bate [17] highlighted different meanings of 'criticality' are applied between research models and industrial standards. Ernst and di Natale [13] argued that a fundamental methodology in academia (i.e., graceful degradation) is not applicable in an industrial system, because of the potential causes of the disastrous consequences. Esper et al. [14] highlighted that the importance of isolation is not sufficiently considered in academia.

[1]Formal definition (including naming) and specific requirements of MCS have not been finalized yet.

A. System Architecture in MCS

System architecture is a conventional and important topic in embedded and computing systems, but it has been rarely considered in the MCS arena.

It is very important to consider MCS from the perspective of the system architecture. Specifically, no matter how complicated the model built in academic research, and how the perfect standards listed in industrial standards, a consistent target of the activities is 'achieving a MCS can be applied in the real world'; and the first step of the target is building the system architecture correctly. Hence, the system architecture can be deemed as a vital interface/connection between industry and academia in the MCS area.

B. Contribution

This paper is the first work considering MCS from the system architecture perspective, which specifically discusses the vital mismatches in academia and industry, and proposes potential solutions from the perspective of the system architecture. Due to the limitation of pages, a prototype MCS architecture, specific design details, related analysis, and experimental evaluation are planned to be presented in our next paper.

It is important to highlight that the objective of this work is not to judge the correctness of different understandings of MCS. Instead, the main intention is showing the impacts of mismatches in MCS between academia and industry, and trying to provide solutions from the rarely considered system architecture perspective. The author sincerely wishes a frequent communication between academia and industry in the future.

II. The State-of-the-Art in Academia

Mismatches/gaps of MCS between academia and industry can be equally discussed 1) from research models to industrial standards, and 2) from industrial standards to research models. This paper starts with a review of the state-of-the-art in academia.

In the last decade, most of the MCS related research proceeds from the real-time community [14]. Most (not all) of the works about MCS are based on a model proposed by Vestal [35], which is also mainly discussed in this paper[2].

[2]In order to make the discussion more generic, the concepts of the earliest Vestal's model is extended in this paper, please also see Esper et al. [15].

The model assumes that the system has several modes of execution (i.e., $L \in \{1,2,3,4,...\}$), and contains a finite set of *sporadic tasks*. Each task τ_i is defined by its period (T_i), relative deadline (D_i), a criticality/assurance level (l_i), and a set of worst-case execution time (WCET) estimates ($\{C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}, ...\}^3$.). The different WCET estimates are meant to model estimations of the WCET in the different system modes. Specifically, the measured WCET from the normal system mode might be used as $C_{i,1}$, whereas at each higher system mode, the subsequent estimates (i.e., $C_{i,2}, C_{i,3}, C_{i,4}, ...$) are assumed to be obtained by more pessimistic WCET analysis techniques or by considering safety margins imposed by certification authorities. The system initializes from mode 1 (i.e., $L = 1$), and all the tasks are scheduled to execute on the core (s). During the run-time, if the system is running in the mode k (i.e., $L = k$), and if any task τ_i exceeds its execution budget (i.e., $C_{i,k}$), the system will switch to the mode $k+1$ (i.e., $L = k+1$). Meantime, tasks with criticality level less than k (e.g., $l < k$) are suspended.

It is important to highlight that this paper describes Vestal's model from a generalized perspective. In the earliest model [35], only a single-core MCS with two system modes (i.e., Low- and High-criticality modes) is discussed. Afterwards, the research model was further extended by different researchers, e.g., the extension on multiple system modes and criticality levels, re-activation of the dropped tasks, consideration of a multi-core and many-core architectures, etc.. (see Burns and Davis [9]). In the context of the theoretic model, practical frameworks are also being developed (e.g., Missimer et al. [28], Kim et al. [26], etc.).

A. System Architecture of Academic MCS Models

Details of implementation (including system architecture) of MCS are rarely discussed in academia. Most (not all) of the works about MCS just simply treat it as a normal embedded architecture [9]:

Hardware Level. To the best of our knowledge, the essential hardware requirements of Vestal's model are never discussed. Even though some papers introduced hardware architectures within a MCS context (e.g., [5], [23], etc.), the requirements are always ignored.

OS Level. In order to achieve the extra privilege (compared to the application level), a system monitor in charge of the mode switch has to be implemented in the OS level [29]. Meantime, the system monitor can be implemented in different ways. For example, Kim et al. [26] achieved the system monitor via modifying the Linux kernel; Missimer et al. [28] implemented the system monitor as an additional hypervisor.

Application Level. Applications with different criticality levels (l) are implemented in the application level. As summarized by Burns and Davis [9], isolation between applications is an essential requirement of Vestal's model. However, the detailed requirements of isolation have not been specified, which is even ignored by most of the academic research.

[3]The model assumes: $C_{i,1} \leq C_{i,2} \leq C_{i,3} \leq C_{i,4}, ...$

Based on the review and discussion, a preliminary system architecture of the academic MCS model can be built in Figure 1.

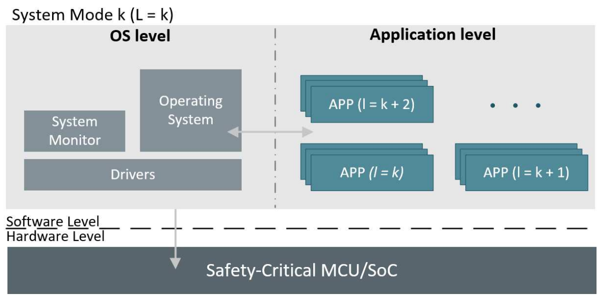

Fig. 1. System Architecture of Vestal's Model in System Mode k ($L = k$)

However, the current system architecture of the academic model (e.g., Figure 1) can not be applied to an industrial product directly, because of mismatches between academic model and industrial standards.

III. Industrial Concepts Regarding MCS

Currently, the requirements of MCS are not clearly guided by any safety-related standards from industry. In this section, we introduce the concepts related to MCS, followed by a specific discussion with system architecture perspective in Section IV.

A. Context

In order to make the discussion precise, the context of this paper is restricted in automotive systems, which is a classic safety-critical system developing towards MCS. In the automotive industry, ISO26262 [19] is the key guidance of safety concepts, which is extended from IEC61508 (a generic safety standard for all the electrical/electronic/programmable electronic (E/E/PE) elements). Hence, this paper mainly discusses the MCS-related concepts in ISO26262 and IEC61508.

Note that, the concepts regarding MCS in automotive systems are very similar to other safety-critical systems (e.g., avionics, railway, etc.), and the discussion in this paper is generic to be applied to other areas.

B. Criticality Level Assignment

Four different criticality levels are supported in ISO26262, which is named as Automotive Safety Integrity Level (***ASIL***) (i.e., $l \in \{A, B, C, D\}$ and $A < B < C < D$). The determination of the criticality level is normally achieved via safety analysis, i.e., Hazard Analysis (*HA*), Failure Modes and Effects Analysis (*FMEA*), and Fault Tree Analysis (*FTA*)[4].

[4]Description of the methodologies and specific procedures do not help the discussion of this paper, please see ISO26262 [19] or related textbook.

Here, we only describe the three parameters directly determining the criticality levels, as the first industrial concept of this paper (*IC-I*), i.e., *severity*, *exposure*, and *controllability*[5].

TABLE I
INDUSTRIAL CONCEPT I ($IC - I$):
CRITICALITY LEVEL DETERMINATION

Severity	Exposure	Controllability		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

S: Severity, E: Exposure, C: Controllability, QM: Quality Management

As shown in Table I, severity, exposure and controllability are used to define the criticality levels of the functions. Generally speaking, the severity defines the estimation of the extent of harm to one or more individuals that can occur in a potentially hazardous situation, the associated exposure is the likelihood of the occurrence of harm, and the controllability is the ability to avoid a specified harm or damage through the timely reactions of the agents involved (e.g. the driver of the vehicle) possibly with support from external measures.

C. Integrating Multiple Criticality Levels

Not only in ISO26262 but also almost in all the safety-related standards, an essential requirement regarding 'integrating tasks with different criticality levels in a same platform' is given (industrial concept II (*IC-II*)):

'*If freedom from interference between elements implementing safety requirements cannot be argued in the preliminary architecture, then the architectural elements shall be developed in accordance with the highest ASIL for those safety requirements.* (ISO26262-3:2018)'

D. Criticality Level Decomposition

Criticality level decomposition is one of the widely used methodologies in industry, which '*allows the apportion of the ASIL of a safety requirement between several elements that ensure compliance with the same safety requirement addressing the same safety goal.*[6]. (ISO26262-9:2018)' — i.e., industrial concept III (*IC-III*), see Figure 2 for an example.

In the design of an industrial automotive system, criticality level decomposition can be found in both hardware and software (e.g., software redundancy). Hence, the impacts from criticality level decomposition in MCS have to be particularly analysed and discussed.

[5]A specfic description of the 3 parameters is outside the scope of this work, please see ISO26262 [19].

[6]A specfic description is outside the scope of this work, please see ISO26262-9:2018 [19]

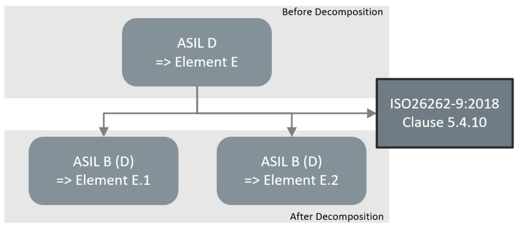

Fig. 2. Example of Criticality Level Decomposition

IV. MISMATCHES BETWEEN ACADEMIA AND INDUSTRY

Mismatches of MCS concepts between academia and industry have been already discussed and analysed by different researchers and engineers, e.g., [13], [14], [17]. Some of the mismatches have been correctly recognized and fixed in the latest research. For example, in the early academic research, the concept of *criticality level* was assigned to both system and applications, and caused confusions [17]. Recently, the 'system criticality level' has been changed to 'system mode', in order to distinct with 'application criticality'.

This section discusses the still existing mismatches, and analyse them from the perspective of system architecture.

A. Hardware Level

In academia, requirements of hardware are never specifically discussed. No matter the practical frameworks or the experiments within the context of academic models, the hardware platforms are not particularly selected [13].

Criticality Level Assignment. As defined in safety-related standards (e.g., IEC61508, ISO26262), different criticalities are also assigned to the hardware, which lead to the different requirements in the design and verification.

Nowadays, commonly used safety-critical systems in industry are not particularly designed for a MCS execution context. In automotive systems, only a single-criticality system can be supported by the latest MCUs/SoCs targeting safety-critical systems from the world's top 3 (in terms of market share) automotive semiconductor suppliers [3]: FS-SBCs [2] in NXP Semiconductors, PRO-SIL family [1] in Infineon Technologies, and the RHx series [4] in Renesas Electronics are all developed towards the same direction: satisfying the highest criticality level of the hardware platform, with increased scalability, reduced complexity and decreased power consumption compared to traditional safety-related methodologies, rather than supporting a MCS context.

As strongly regulated in *IC-II*, due to lack of sufficient separation/isolation to provide freedom from interference between elements with different criticality levels, the whole hardware platform used to achieve a MCS is required to *inherit the highest criticality level of the applications in the software.*

Criticality Level Inversion. As described in the previous sub-section, current hardware platforms are not particularly designed for a MCS context, which may cause criticality level inversion during the system mode switch. Taking a classic extension of Vestal's model, i.e., Adaptive Mixed Criticality (***AMC***) [7], as an example, whilst the system executes in the high-criticality mode, low-criticality tasks are terminated to ensure the execution of high-criticality tasks. However, low-criticality tasks may have outstanding requests/instructions underway at the hardware level. Instructions/requests latched in the pipelines of the CPU(s) can be easily removed via specific instructions. However, already requested I/O operations, which are latched in the hardware buffers, cannot be cleared timely. In this case, the requests from low-criticality tasks that are already sent still block the accesses of I/O requests from high-criticality tasks — i.e., criticality level inversion.

With an increased number of processors and frequency of I/O requests [20], [24], criticality level inversion may significantly reduce the effectiveness of the system mode switch, and further cause the corruption of the whole system. Hence, in order to avoid the criticality level inversion in MCS, the platform is required to support the function *'remove the already request I/O instructions from hardware'*.

B. OS level

As shown in Figure 1, drivers, Operating System (OS), and system monitor are the three key components in the OS level. The implementation of the components can be either combined or separated.

Criticality Level Assignment. There are two possible situations of assigning criticality levels to the drivers and OSs:

- If applications with different criticality levels access shared drivers and OS, the drivers and OS are required to *keep the highest criticality level of the system*.
- If applications with different criticality levels access separated and independent drivers and OSs, the drivers and OSs are required to *inherit the criticality level of the accessing application.* For example, virtualization and kernel separation [28].

When it comes to the system monitor, according to *IC-II*, the system monitor is required to *inherit the highest criticality level of the applications in the software*, because all the tasks have to be dependant on the system monitor for resource sharing and system mode switch.

System Monitor and System Mode Switch. As reviewed in Section III-B, system mode switch is the key strategy in academia, which guarantees the execution of the more critical tasks. Specifically, the system starts from a low-criticality mode and can be changed to a higher-criticality mode in a predefined condition (e.g., over-execution of a task), which causes the termination of the low-criticality tasks. This procedure has to be achieved via a privileged system monitor (e.g., hypervisor), which is implemented in the OS level by most of the practical works within the context of academic models.

However, system mode switch is a key mismatch of MCS between academia and industry — i.e., system mode switch is never defined by any safety-related standards (according to the best of our knowledge). As discussed by Graydon and Bate [17], and Ernst et al [13], 'terminating/killing tasks in any criticality level can potentially cause a catastrophic consequence'.

Consider that, to bridge this key issue, it is very vital to understand the conclusion from Graydon et al. [17] and Ernst et al [13], which leads to an earlier stage before academic model – i.e., criticality level assignment (see *IC-II* in Section III-B). As shown in Table I, the criticality level is directly determined by three parameters, i.e., *severity*, *exposure*, and *controllability*. Among these parameters, only *severity* indicates the severity class of terminating a specific module. Therefore, with the system mode switch, terminating a low-criticality task can cause a more severe consequence than terminating a high-criticality task. For example, terminating an ASIL A task with $\{S3, E2, C2\}$[7] is much more dangerous than terminating an ASIL B task with $\{S1, E4, C3\}$.

Within a sufficient consideration of the industrial context, two more items are also required to be considered during system mode switch:

- **Dependency**: If a high-criticality application is dependant on the inputs from a low-criticality task, terminating a low-criticality application will also cause the corruption of the high criticality application, also see *IC-II* in Section III-B.
- **Criticality Level Decomposition**: As introduced in *IC-III*, Section III-D Criticality level decomposition is a commonly used methodology in industry, which allows a high-criticality component to be decomposed to several low-criticality components. Hence, in a MCS with criticality level decomposition, simply terminating all the low-criticality tasks during system mode switch will also significantly corrupt the system [8].

Although system mode switch is an undefined concept in industry, it can be somehow linked to 'graceful degradation' — i.e., a technique aimed at maintaining the more important system functions available, despite failures, by dropping the less important system functions. [19]

Hence, determining the right 'important functions' (rather than the high-criticality functions) and terminating the 'less important functions' is a methodology to implement system mode switch in industry. Here, we propose a two-level analysis: (see Figure 3).

- **Step 1: Failure Modes Effect Analysis (FMEA)**[8]: The impacts of terminating each task in the current system mode shall be analysed. Any task can cause catastrophic consequence from its termination has to be kept in the next system mode, i.e., important tasks.
- **Step 2: Dependency Analysis**: The dependency of the important tasks (output from step 1) shall be analysed. Any task that can cause corruption of an important task due to its termination, has to be kept in the next system mode.

[7] In ISO26262, a large number indicates a more critical situation.

[8] The description of FMEA is out-of-score, which can be found in ISO26262 [19] or any text book of safety.

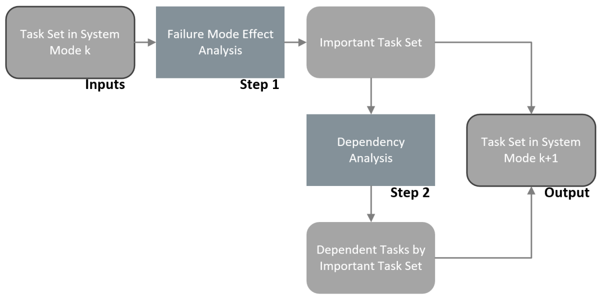
Task Set in System Mode k
Inputs
Failure Mode Effect Analysis
Step 1
Important Task Set
Dependency Analysis
Step 2
Task Set in System Mode k+1
Output
Dependent Tasks by Important Task Set

Fig. 3. 2-Level Analysis for System Mode Switch from Mode k to Mode $k+1$

The output of the two-step analysis is the complete task set for the next (more critical) system mode.

Response Time Analysis. Due to introduction of the two-level analysis, the response time analysis of the MCS model has to be re-considered.

Similar as the academic model (i.e., Vestal's model), the response time of the proposed industrial model has to be analysed in three phases, i.e., current system mode, next (more critical) system mode, and system mode change [35]. The only difference between the two analysis is the kept tasks in the next system mode. Due to the limitation of the pages, a specific analysis will be presented in our following paper.

C. Application Level

Tasks/applications with different criticality levels are normally implemented in the application level. As presented in the previous sections, almost all the research works focus on the schedulability and shared resource management among the tasks.

While integrating the tasks with different criticality levels, the essential requirement of industrial MCS is separation/isolation, see *IC-II*, Section III-B, which can be further expanded to:

- **Temporal Isolation**: tasks with different criticality levels shall be temporally separated, in order to avoid malfunction caused by consuming too high processor execution time or by blocking a shared resource by one task.
- **Spatial Isolation**: tasks with different criticality levels shall not exchange data (including using shared memory).
- **Fault Isolation**: Fault(s) from one task shall not be propagated to tasks with different criticality levels.

Temporal isolation is one of the main targets of the academic models, and TDMA-based methodologies are commonly used. For example, Carvajal and Fischmeister [10] proposed an open-source framework (Atacama) for real-time Ethernet for MCS. Cilku et al. [11] introduced a TDMA-based bus arbitration scheme. Goossens et al. [16] used a TDMA-based approach to schedule concurrent memory requests of the same physical memory. Even so, TDMA-based approaches cannot satisfy the requirements on spatial and fault separation and usually leads to resource waste.

Data and fault isolation are typically achieved via two approaches:

- **Physical separation** segregates the tasks by allocating unique hardware resources to tasks with different criticality levels.
- **Virtual separation**: separate the components by establishing partitioned hardware provisions that allow multiple software components to run on the same hardware platform.

Due to the lack of a hardware platform particularly designed for MCS (see Section IV-A), achieving physical separation can only utilize the technologies designed for other purposes, e.g., ARM TrustZone. ARM TrustZone technology is centered around the concept of two hardware-enforced protection domains (secure world and non-secure world). Each world is granted uneven privileges, with non-secure software prevented from directly accessing secure world resources. LTZVisor [30] and TZDKS [12] proposed MCS architectures using TrustZone technologies, by implementing high-criticality tasks in the secure world and low-criticality tasks in non-secure world.

In virtual separation, virtualization technology is mostly used. Specifically, the virtual machines (*VM*s) are logically isolated in a virtualized system, which means the applications executed in one VM can never affect another VM, even if it breaks down. This is highly linked to the requirement on isolation of data and fault. For example, Groesbrink et al. [18] and Li et al [28] utilised hypervisor-based virtualisation to separate system to independent partitions (i.e. VMs). MultiPARTES [34] and BlueIO [22] used para-virtualisation [31] to establish an I/O virtualization system for a MCS. In these methodologies, different system modes are assigned to the VMs, and a secondary scheduling between the VMs is also built to guarantee the more critical requests can be served earlier. However, virtualization technologies involve complicated resource management and complex path of instructions, which significantly affect the performance and predictability of the system, which is also the essential requirements of a real-time system [21], [25].

D. Industrial Architecture of MCS

Based on the discussion in previous sub-sections, the industrial MCS model is shown in Figure 4.

As shown in Figure 4, shared components among tasks with different criticality levels inherit the highest criticality of the executing tasks, including hardware platform, drivers, system monitor, and OS. Meantime, tasks and OS with different criticality levels are separated in the independent environment, in order to avoid interference from time, space and faults.

V. Conclusion

Mixed Criticality Systems (MCS)s are a vital direction of safety-critical systems in both academia and industry. However, due to the mismatched concepts between academia and industry, it is nearly impossible to export an academic MCS model to an industrial system directly. This paper specifically

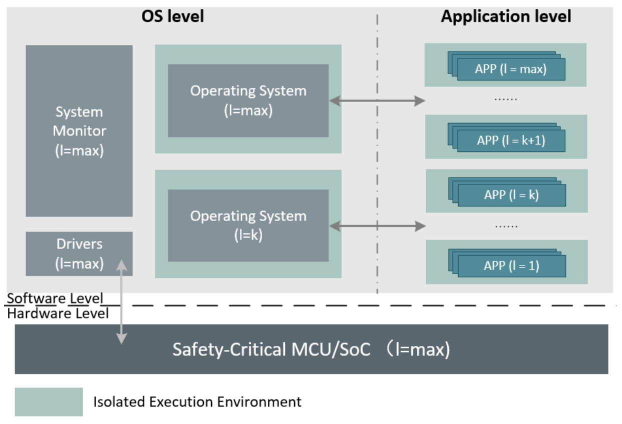

Fig. 4. System Architecture of Ideal Industrial MCS Model (System Mode $L = 1$)

discusses and analyses the mismatches from a rarely considered perspective (i.e., system architecture), with the potential solutions.

The key intention of this paper is encouraging more frequent communication between academia and industry, which is able to accelerate the evolution of the MCS area significantly.

References

[1] Infineon: 32-bit TriCore Microcontroller. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/. Accessed Jan 12, 2019.

[2] The official website of nxp. https://www.nxp.com/. Accessed Jan 12, 2019.

[3] The overview of global automotive semiconductor markets from 2018 to 2020. https://www.researchandmarkets.com/reports/4702372/automotive-semiconductor-market-report-trends. Accessed Jan 12, 2019.

[4] Renesas: Micro controllers. https://www.renesas.com/eu/en/products/microcontrollers-microprocessors.html. Accessed Jan 12, 2019.

[5] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul. I/o contention aware mapping of multi-criticalities real-time applications over many-core architectures. 2016.

[6] S. Baruah. Mixed-criticality scheduling theory: Scope, promise, and limitations. *IEEE Design & Test*, 35(2):31–37, 2018.

[7] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43. IEEE, 2011.

[8] K. Bletsas, M. Ali Awan, P. Souto, B. Åkesson, A. Burns, and E. Tovar. Decoupling criticality and importance in mixed-criticality scheduling. In *6th International Workshop on Mixed Criticality Systems (WMC 2018)*, pages 25–30, 2018.

[9] A. Burns and R. Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.

[10] G. Carvajal and S. Fischmeister. An open platform for mixed-criticality real-time ethernet. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 153–156. IEEE, 2013.

[11] B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro. A tdma-based arbitration scheme for mixed-criticality multicore platforms. In *Event-based Control, Communication, and Signal Processing (EBCCSP), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[12] P. Dong, A. Burns, Z. Jiang, and X. Liao. Tzdks: A new trustzone-based dual-criticality system with balanced performance. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 59–64. IEEE, 2018.

[13] R. Ernst and M. Di Natale. Mixed criticality systemsa history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[14] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pages 139–148. ACM, 2015.

[15] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. An industrial view on the common academic understanding of mixed-criticality systems. *Real-Time Systems*, 54(3):745–795, 2018.

[16] S. Goossens, J. Kuijsten, B. Akesson, and K. Goossens. A reconfigurable real-time sdram controller for mixed time-criticality systems. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 2. IEEE Press, 2013.

[17] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. *Proc. WMC, RTSS*, pages 19–24, 2013.

[18] S. Groesbrink, S. Oberthür, and D. Baldin. Towards adaptive resource management for virtualized real-time systems. In *4th Workshop on adaptive and reconfigurable embedded systems*, 2012.

[19] I. ISO. 26262: Road vehicles-functional safety. *International Standard ISO/FDIS*, 26262, 2011.

[20] Z. Jiang. *Real-Time I/O System for Many-core Embedded Systems*. PhD thesis, University of York, 2018.

[21] Z. Jiang and N. Audsley. Vcdc: The virtualized complicated device controller. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[22] Z. Jiang, N. Audsley, and P. Dong. Blueio: A scalable real-time hardware i/o virtualization system for many-core embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(3):19, 2019.

[23] Z. Jiang, N. Audsley, P. Dong, N. Guan, X. Dai, and L. Wei. Mcs-iov: Real-time i/o virtualization for mixed-criticality systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 326–338. IEEE, 2019.

[24] Z. Jiang and N. C. Audsley. Gpiocp: Timing-accurate general purpose i/o controller for many-core real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 806–811. IEEE, 2017.

[25] Z. Jiang, N. C. Audsley, and P. Dong. Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 75–84. IEEE, 2018.

[26] N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter. Supporting i/o and ipc via fine-grained os isolation for mixed-criticality real-time tasks. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pages 191–201. ACM, 2018.

[27] M. Kyriakidis, R. Happee, and J. C. de Winter. Public opinion on automated driving: Results of an international questionnaire among 5000 respondents. *Transportation research part F: traffic psychology and behaviour*, 32:127–140, 2015.

[28] Y. Li, M. Danish, and R. West. Quest-v: A virtualized multikernel for high-confidence systems. *arXiv preprint arXiv:1112.5136*, 2011.

[29] J. L. Peterson and A. Silberschatz. *Operating system concepts*, volume 2. Addison-Wesley Reading, MA, 1985.

[30] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral. Ltzvisor: Trustzone is the key. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[31] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE, 2010.

[32] K. Seeger. *Semiconductor physics*. Springer Science & Business Media, 2013.

[33] N. R. Storey. *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[34] S. Trujillo, A. Crespo, and A. Alonso. Multipartes: Multicore virtualization for mixed-criticality systems. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 260–265. IEEE, 2013.

[35] 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*, pages 239–243. IEEE, 2007.