ODRE Workshop:
Using SIL Arithmetic to Design Safe and Secure Systems

Catherine Menon
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
c.menon@herts.ac.uk

Saverio Iacovelli
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
saverio.iacovelli@istruzione.it

Raimund Kirner
Department of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
r.kirner@herts.ac.uk

***Abstract*—In a safety-critical system each service has a specific level of safety criticality. Safety standards use classifications like *Safety Integrity Levels* (SIL), to describe the design requirements for the individual services of a system. Techniques like redundancy can be used to achieve a higher overall dependability than the used individual components provide. Using the notion of SIL, this can be called SIL arithmetic.**

In this paper we describe the concept of SIL arithmetic and point out how different safety standards provide hints for their support of using SIL arithmetic. We highlight the principal benefits of SIL arithmetic and provide simple examples. But the use of SIL arithmetic in a concrete system design can also have its pitfalls, which we also discuss in this paper. We specifically discuss these issues in the context of scheduling techniques for mixed-criticality systems, where resource shortages are to be handled by the scheduler.

Keywords: Mixed-criticality scheduling, Cybersecurity, Safety Integrity Levels (SIL), Industrial Control Systems (ICS), Cyber-physical systems

I. Introduction

Real-time computer systems are systems in which the correctness of the system behaviour depends not only on the logical results of computations but also on the physical time at which such results must be provided [16]. Historically, hard real-time computing has been primarily developed to support safety-critical systems or industrial systems that have to guarantee certain performance requirements with a limited degree of tolerance. In such systems, most computational activities are characterised by critical timing requirements that have to be met in all operating conditions in order to guarantee the correct system behaviour and a deadline missed can have catastrophic effects on the controlled environment.

A hard real-time system must execute a set of recurrent tasks such that all time-critical tasks meet their specified deadlines and a task instance finishing after its deadline can jeopardise the whole system behaviour. In order to guarantee a predefined performance, hard real-time systems are designed under worst-case scenarios where all resources are statically allocated to tasks based on their maximum requirements. Such systems are often modelled as a set of recurrent tasks, each characterised by a *worst-case execution time* (WCET), to be executed on the selected hardware platform by a real-time scheduler. As all other computing systems, real-time systems typically provide different services according to their specifications and each service can have a different criticality.

A mixed-criticality system is a system in which multiple functionalities of different criticalities are implemented and integrated on the same platform. In most cases, criticality levels are assigned according to the importance to safety of this functionality, as is the case for most industrial standards. In such cases, mixed-criticality systems can be viewed as mixed safety-criticality systems. Depending on the domain, safety criticality and integrity in these systems may be implemented via *Automotive Safety Integrity Levels* (ASIL) in ISO 26262 [14], *Item Development Assurance Levels* (IDAL), *Design Assurance Levels* (DAL) in DO-178C [21], and *Safety Integrity Levels* (SIL) in IEC 61508 [13].

Mixed-criticality systems pose new challenges with regard to scheduling. This occurs because resource shortages can cause the higher criticality tasks' instances to miss their deadlines due to interference with the lower criticality ones. Most research on mixed-criticality scheduling focuses on a very specific pattern of resource shortage, namely the underestimation of the worst-case execution time (WCET) of tasks of lower criticality, while at the same time guaranteeing that all tasks of higher criticality are proven to be schedulable [6], [9], [2]. With this particular fault model in mind, it is guaranteed that all high criticality tasks are schedulable.

Within this paper we look at more general fault models, including cases where schedulability of all high-criticality tasks cannot be guaranteed anymore. The motivation for our view is that in order to build a robust system, tolerance of other faults, like of hardware faults, has to be taken into account.

In this paper we discuss issues of SIL arithmetic in light of such a more general fault model. With SIL arithmetic, a service A might be implemented by two or more (even redundant) tasks $D_i(A)$, with each task itself $D_i(A)$ assured to a lower SIL than the SIL of the service A which they jointly implement. This paper discusses the behaviour a robust mixed-criticality scheduler should have in order to deal with sudden or unexpected resource shortages that will not even allow the execution of all high-criticality tasks. The discussion focuses on what schedulers in general should be aware of. We do not focus on a specific scheduler and its evaluation.

In Section II we briefly describe *safety integrity levels* (SIL). Section III describes scheduling of systems with tasks of mixed criticality. SIL arithmetic is described in Section IV. Scheduling aspects considering systems with SIL arithmetic are discussed in Section V. Section VI discusses how security issues influence also the safety issues of systems with SIL arithmetic. Section VII concludes the article.

II. Safety Integrity Levels

As defined in IEC 61508 [13], the *safety integrity* of a component is the probability of that component satisfactorily performing its specified safety function. There are four *Safety Integrity Levels* (SIL) defined in [13], and components which are more important to safety are ascribed a higher safety integrity level than those of lesser importance. This reflects the contribution of these components to the overall system safety, i.e., components of greater importance to safety must have lower probability of failure. This can be seen where the component is providing a safety function to some *Equipment Under Control* (EUC), whereby the component's safety integrity level corresponds to the relative level of risk reduction provided by that component to the EUC.

Developing a higher SIL component requires more resources than developing a lower SIL component. The rigour around development and validation at higher SILs guarantees that a component will also satisfy higher safety requirements. As defined in [13], SIL of a component is linked to a certain failure rate of it. More specifically, safety integrity requirements are based on a probabilistic analysis of failure of a specific function. The higher is the SIL, the lower is the *probability of failure on demand* (PFD) or, for continuous operations, the *probability of failure per hour* (PFH). This is reproduced in Table I.

PFD	*PFH*	*SIL*
10^{-4} to 10^{-5}	10^{-8} to 10^{-9}	4
10^{-3} to 10^{-4}	10^{-7} to 10^{-8}	3
10^{-2} to 10^{-3}	10^{-6} to 10^{-7}	2
10^{-1} to 10^{-2}	10^{-5} to 10^{-6}	1

Table I
Resulting safety integrity level (SIL) from different failure probabilities PFD and PFH

However, we note that using the development techniques recommended for a particular SIL does not guarantee the achievement of a given failure rate. As an example, Table I should be used only for determining the SIL based on the required failure rate and not in claiming satisfaction of some maximum level of failures per hour.

III. Mixed-Criticality Scheduling of Real-Time Systems

Real-time scheduling is a well established research field. For example, *rate-monotonic scheduling* (RMS) and *earliest deadline first* (EDF) are well-known optimal scheduling algorithms, the former for static priorities and the latter for dynamic priorities on a single-processor system [17]. However, the classic real-time scheduling algorithms are not specifically targeted towards systems with resource shortage. For example, having a system with multiple tasks of different criticality, it would make sense to prioritise tasks of higher criticality in case of resource shortage. Real-time algorithms are agnostic to task criticality, and by themselves are of limited use for mixed-criticality systems. Mixed-criticality scheduling is about scheduling methods based on a fault model.

Vestal et al. proposed mixed-criticality scheduling with a concrete fault model [22]. In his task model, only an optimistic scheduling of WCET is known for tasks with lower criticality. By contrast, for tasks of higher criticality a safe upper bound of the WCET is assumed also to be available. Vestal's fault model is that lower-criticality tasks may overrun their optimistic WCET estimate at runtime. A significant body of research has been established [1], [8], [2] based on Vestal's original basic fault model. The key point with Vestal's fault model is that it is possible to establish a guarantee that the higher-criticality tasks are still schedulable based on their safe upper WCET bound.

In this paper we look at fault tolerance for mixed-criticality scheduling in a broader way than Vestal's fault model. We motivate this by noting that it is possible that due to faults it may not even be possible to guarantee the schedulability of the higher-criticality tasks. To cope with such cases one might work with redundancies, possibly in combination with diverse programming, and make scheduling decisions that ensure the overall system utility is as high as possible.

To clarify the discussion of mixed-criticality scheduling throughout this paper, we introduce and describe some basic concepts. Figure 1 shows some relations between concepts of the control system and the environment. First, we distinguish between services S_i and tasks t_j. A service is a functional behaviour that a system has to provide. The services are virtual concepts as they might not be directly reflected in the components' implementation inside the system. On the implementation side, we have multiple tasks with their run-time instantiation called jobs. The services

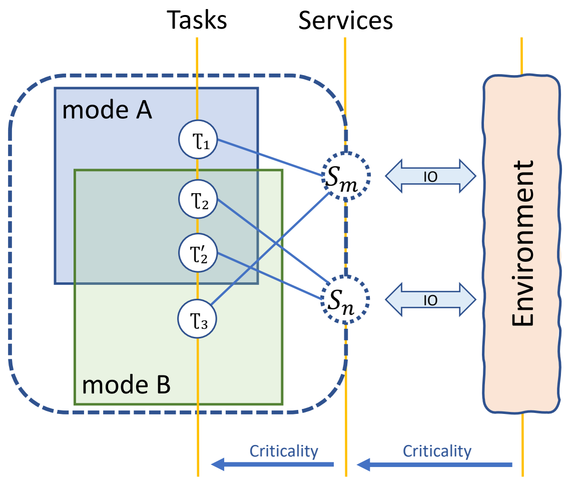

Figure 1. The Concept of Tasks and Services

describe the functional relationship of the system with its environment, i.e., the system reading input from sensors and setting values via actuators. When designing system services, it is important to consider the potential negative impact in case the failure of an individual service has.

This measure of the negative impact of failure is called *criticality*. A higher criticality means a higher negative impact in case of failure. To determine the criticality of a service it is important to consider the application context of the system. For example, a simple service to control the position of a valve would have a quite different criticality if it is used in an aeroplane to control the cabin pressure (which is a safety-critical function) or to control the water level in a water tank for a toilet flush (which would be of relative low criticality). The criticality of a service is then used to determine the SIL that is needed for this particular service by following safety standards relevant to this application area. While we recognise that it may be useful to differentiate between multiple levels of criticality, for simplicity in this paper we will use high and low criticality to explain concepts.

What is important here is that there is no 1:1 relationship between services and tasks. For example, in Figure 1 we have service S_m implemented by tasks τ_1 and τ_3, and service S_n be implemented by tasks τ_2 and τ_2'. Furthermore, the system might have different modes, where each mode can have its own set of tasks to be executed. Depending on the stage of a mission or external events, the system might switch between different modes. For example, in previous work on mixed-criticality scheduling, fault events like the overrun of the optimistic WCET estimate have been used to switch from a normal operation mode into emergency modes where tasks of higher criticality are given preference [3], [4], [5], [12].

In our example in Figure 1 we have service S_m being implemented by different tasks depending on the concrete mode, i.e., task τ_1 in mode A and task τ_3 in mode B.

Authors have called the inclusion of a task to a specific mode the *task's importance* for that mode in order to make clear that criticality is not the sole criterion to determine scheduling decisions [7]. While the importance of a service depends from outside on the concrete control application, the importance of a task depends on internally which mode is currently active.

Tasks τ_1 and τ_2 might require a different amount of resources, and consequently provide a different quality of service. Service S_n is implemented by the same two tasks τ_2 and τ_2' in both modes. That service S_n is implemented by two tasks could have different reasons, e.g., τ_2 and τ_2' both implement just parts of the function of S_n, or maybe τ_2 and τ_2' are just redundant tasks both implementing S_n on their own. A task's criticality is inherited from its service. If a task implements more than one service, then the task's criticality is the maximum of the criticalities of all the services it implements.

IV. SIL Arithmetic

SIL arithmetic, or SIL synthesis [13], is the practice of combining together multiple components at a relatively low integrity level to realise a function at a higher integrity level [24]. SIL arithmetic is dependent on the concept of functional redundancy, which involves the duplication of certain critical system components which all provide a defined function. If any one of these components fails, the remaining components will still be able to provide that functions. The practice of SIL arithmetic leverages redundancy to permit claiming a higher achieved SIL for the function than the achieved SIL of any of the individual redundant components.

SIL arithmetic in the automotive domain is known as ASIL decomposition, and is commonly used where system-level requirement have been decomposed into redundant sub-requirements allocated to different components [10]. If one of the components fails to satisfy its sub-requirement then the other component may still do so, meaning the overall system-level requirement remains satisfied. From this, we see that a system implementing redundancy correctly has a higher likelihood of satisfying its requirements than an otherwise-identical system without redundancy.

We note that an effective system of redundancy management [11] is required in order to detect primary component failure and to reconfigure the system to use the redundant component in place of the primary. Effective redundancy also requires independence of the redundant components such that multiple components will not be affected, for example, by a common mode failure. Systems with redundancy built in can continue to operate - in some cases up to several days [15] - in the event of partial failure.

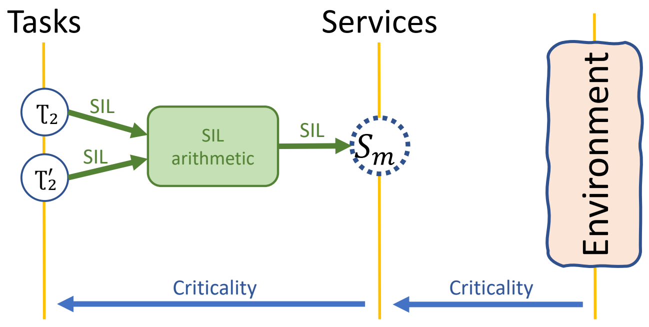

Figure 2. The Principle of SIL Arithmetic

Figure 2 visualises the idea of SIL arithmetic. The example shows a service S_m, being implemented by two independent but redundant tasks τ_2 and τ_2'. As already described in Section III, the criticality of the service is derived from the criticality of the environment of the control system, i.e., the application. The criticality of the tasks is inherited as the maximum from the services they implement. As both τ_2 and τ_2' only implement one service S_m, their criticality is the same as that of service S_m. On the other hand, the SIL of the service S_m is then derived from the SIL of τ_2 and τ_2' via SIL arithmetic. The resulting SIL of service S_m has to be appropriate for its criticality determined by the application of the control system.

A. *Benefits of SIL Arithmetic*

SIL arithmetic can be beneficial in terms of reducing development time and costs since it is generally regarded as less resource-intensive to develop components at lower SILs, although demonstrating sufficient independence between these components to permit SIL arithmetic may still be a non-trivial task [20]. SIL arithmetic also allows for the commercial pressures of developing and procuring systems. In some cases these pressures mean that components have to be procured before their SIL can be assured, or that use of legacy components at a lower SIL is required [23].

Another benefit of developing components to a lower SIL is the consequent reduction in complexity of these components. Components of lower complexity are easier to develop, and the risk of an undetected failure mode is lessened. Furthermore, these components may also be easier and cheaper to maintain.

B. *SIL Arithmetic and Standards*

In IEC 61508 [13], SIL arithmetic in hardware systems is endorsed as part of a discussion of the ways in which systems of different SILs can be combined, and the effect on the SIL of the resultant combined system. Where a safety function is implemented via multiple channels with a given hardware fault tolerance, the overall SIL is calculated by identifying the channel with the highest SIL, and adding a number of integrity levels dependent on the hardware fault tolerance of the combined channels. There is a limit on the SIL increment which can be claimed using this method.

ISO 26262 [18] also discusses SIL arithmetic, terming it ASIL decomposition. Here, a safety function assigned a nominated ASIL (*Automotive Safety Integrity Level*) can be decomposed into redundant safety requirements, satisfied by independent architectural elements. A commonly-used implementation of this is to decompose a safety requirement into a functional requirement and a safety mechanism, which acts against failure of that functionality. As with the SIL arithmetic discussed in [13], the combinations which represent an acceptable ASIL decomposition are limited.

This is described in Table II, where ASIL D is the most rigorous level and ASIL QM equivalent to SIL 0. Where an ASIL is decomposed into less rigorous ASILs, the original ASIL is shown in brackets following the decomposition.

ASIL	*Acceptable Decomposition*		
ASIL D	ASIL C(D)	+	ASIL A(D)
	ASIL B(D)	+	ASIL B(D)
	ASIL D(D)	+	ASIL QM(D)
ASIL C	ASIL B(C)	+	ASIL A(C)
	ASIL C(C)	+	ASIL QM(C)
ASIL B	ASIL A(B)	+	ASIL A(B)
	ASIL B(B)	+	ASIL QM(B)
ASIL A	ASIL A(A)	+	ASIL QM(A)

Table II
ASIL DECOMPOSITION

A third standard which endorses SIL arithmetic is ARP 4754 which contains the recommended practices for development cycle of civil aircraft and systems [19]. This guideline, maintained from the *SAE International*, addresses both functional safety and design assurance process and it is suppoted by the aviation standards like DO-178C and DO-254. The safety level, named *Design Assurance Level* (DAL), is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system. There are five safety levels with A being the most critical and E the less critical one. As with both ISO 26262 and IEC 61508, there are further constraints on the extent of SIL arithmetic that can be performed in accordance with this.

C. *Pitfalls of SIL Arithmetic*

Although SIL arithmetic confers benefits as described in Section IV-A, it can lead to some potentially ambiguous situations related to *safety assurance*. Mixed-criticality tasks are scheduled according to the criticality of the service they implement and in case of services composed by individual tasks the SIL level exactly reflects the task criticality.

However, within systems where two or more redundant components are linked via SIL arithmetic to realise a service at a higher SIL, if one component fails then the "protective" element of redundancy is removed. A consequence of this is that the entire service can no longer be adequately assured at the higher SIL as it is now provided only by a single component that itself is at a lower SIL.

The second issue that is necessary to consider is related to component failure. If multiple dependent not redundant components are linked to implement a service, a failure of one of these components could result in a failure of the overall service since an important part of the sub-goal would be no longer achievable. In this case, the scheduler may choose to abandon all the related tasks implementing the entire service.

Therefore component or, more generally, sub-system failure is a significant concern for safety-critical systems since this can jeopardise the correct functioning also of the related system components.

V. SIL-Arithmetic-aware Scheduling

Safety-critical systems are hard real-time systems providing different services S_i implemented by one or more tasks τ_j. Table III and Table IV contain respectively a description of tasks within an *Unmanned Aerial Vehicle* (UAV) used for monitoring purposes, and the SIL that each task and function can be adequately assured to. Apart from service S_3, each service is implemented by one task.

Service (Task)	*Description*	*SIL*
S_1 (τ_1)	trajectory	3
S_2 (τ_2)	earth monitoring	2
S_3 (τ_3 and τ_3')	communication with station	2
S_4 (τ_4)	logging of tasks' events	1

Table III
UAV Example: tasks prior to failure

The service most important to safety is S_1, which is responsible for keeping the drone trajectory steady, and consequently is designed to SIL 3.

Service S_2 and S_3 are responsible respectively for monitoring the soil via camera and for communication with base station. Lastly, service S_4 records every metadata and event related to the above tasks. In this table service S_3 represents a function realised at SIL 2 by incorporating two redundant and independent SIL 1 tasks τ_3 and τ_3'.

Table III provides an example of the first problem identified in Section IV-C. If the scheduler suddenly drops τ_3 when resources are scarce, then service S_3 can no longer be assured to the same SIL, owing to the loss of redundancy.

Table IV shows the situation after a failure of one of the redundant tasks implementing the service S_3. Here, a failure of τ_3 has resulted that service S_3 is now solely provided via task τ_3', which is still scheduled at the same criticality level assigned after the design phase.

Service (Task)	*Description*	*SIL*
S_1 (τ_1)	trajectory	3
S_2 (τ_2)	earth monitoring	2
S_3 (τ_3' only)	communication with station	1
S_4 (τ_4)	logging of tasks' events	1

Table IV
UAV Example: tasks after failure

One solution can be to appropriately manage the hardware redundancy. This may be done via hotswapping independent and redundant components to increase the SIL level provided by the overall function affected as soon as possible. Together with this, we propose that systems incorporate a system monitor that signals to the scheduler when sub-components fail so as to undertake the appropriate countermeasures. Whenever any task suddenly fail, the scheduler should make a new schedulability test to check if the newly available task set is still schedulable and find a novel suitable schedule.

VI. Implications of Security on System Safety

The implementation of critical services with redudant independent tasks via SIL arithmetic enhances the overall robustness in terms of safety in case of malfunctioning due to a cyberattack. In general, tasks that are vulnerable to attacks are those that communicate with the outside world or those that are dependent from tasks communicating with the external environment. As an example, considering the use case used for Table III and Table IV, service S_3 can be vulnerable to a malicious attacker. Different types of attacks are possible:

- *Man in The Middle Attack* (MTMA): can cause a malfunction leading task τ_3 to be dropped so to increase its probability of failure of service S_3.
- Malicious software (e.g., virus or trojan): can be injected in the system.
- *Distributed Denial of Service* (DDoS): can lead to drop task τ_3 first and, if countermeasures are not taken, task τ_3' afterwards, leading to completely stop service S_3 (service disruption).

In all cases, a security breach can automatically have an impact on the mixed-criticality scheduling process since some task can unexpectedly become unavailable. If safety-critical tasks are affected, this can have severe consequences. As an example, rail critical services accessible via wireless network can be damaged by an attacker that can send forged speed and braking profile information to a locomotive using a train control application and affect the dependent and even more critical braking task.

Therefore, adding functionalities that allow to communicate with the external world can help to build more useful and flexible systems but, at the same time, can expose critical services to novel and unexpected threats. The mixed-criticality scheduling so far protects highly critical tasks by isolating them from the execution interference of low critical ones in case of malfunctioning or resource shortages. However, as modern crafts become more and more connected to the outside world, there is also the need to protect such critical tasks from new potential threats. As an example, it can be possible to alter the task scheduling or the overall system safety where high critical task is dependent from data coming from a low critical task communicating with the external environment or affected by malicious software. An even more critical situation to manage could be when a highly critical task is directly exposed to the external environment.

In this regard, the challenge woud be that to check consistency and integrity of data managed by safety-critical instances and, generally, to be sure that bogus data exchanged during the scheduling process do not have an impact on completion of safety-critical instances.

VII. Summary and Conclusion

This paper discusses the development of SIL arithmetic within safety-critical systems, and the interaction with scheduling concerns. It identifies how to appropriately manage the scheduling process within a mixed-criticality system in order to minimise the impact on safety in the case of malfunction or cyberattack.

We extended the scope of mixed-criticality scheduling by replacing Vestal's fault model focusing on WCET overruns with a generic fault model. We also described where criticality of a task and service is derived from, and how SIL arithmetic is used in practice. We have extended this to identify some potential pitfalls of using SIL arithmetic and proposed that mixed-criticality schedulers need to be aware of any implementations of SIL arithmetic within systems in order to make adequate decisions. We have also considered how SIL arithmetic may impact security issues. Our identification of current challenges for SIL arithmetic and scheduling has highlighted the utility of the SIL arithmetic technique, but also raised the concerns that this must be supported adequately by the mixed-criticality scheduler.

Future work will be to propose a concrete mixed-criticality scheduler for systems which implement SIL arithmetic, and to demonstrate the effectiveness of this scheduler.

References

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

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

[3] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 34–43, Washington, DC, USA, November 2011. IEEE Computer Society.

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

[5] I. Bate, A. Burns, and R. I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, 43(4):298–320, Apr. 2017.

[6] K. Bletsas, M. A. Awan, P. F. Souto, B. Akesson, A. Burns, and E. Tovar. Decoupling criticality and importance in mixed-criticality scheduling. In *Proc. Workshop on Mixed Criticality (WMC'18)*, pages 25–32, Dec. 2018.

[7] K. Bletsas, M. A. Awan, P. F. Souto, B. Akesson, A. Burns, and E. Tovar. Decoupling criticality and importance in mixed-criticality scheduling. In *6th International Workshop on Mixed Criticality Systems at the Real Time Systems Symposium (RTSS 2018) - Nashville, Tennessee, USA*, December 2018.

[8] A. Burns and R. I. Davis. Mixed criticality systems - a review. Research Report V4-31/7/2014, University of York, Department of Computer Science, York, UK, July 2014.

[9] A. Burns, R. I. Davis, S. Baruah, and I. Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, Oct 2018.

[10] J. D'Ambrosio and R. Debouk. Asil decomposition: The good, the bad, and the ugly. *Proceedings of the Society of Automotive Engineers World Congress*, 2013.

[11] A. Frigerio, B. Vermeulen, and K. Goossens. Component-level asil decomposition for automotive architectures. *Proceedings of the 2019 International Conference on Dependable Systems and Networks Workshops*, pages 62–69, 2019.

[12] S. Iacovelli and R. Kirner. A lazy bailout approach for dual-criticality systems on uniprocessor platforms. *Designs*, 3(1), 2019.

[13] International Electrotechnical Commission. Functional safety of electrical / electronic / programmable electronic safety-related systems. IEC standard 61508, 1998.

[14] ISO/DIS. Road vehicles – functional safety. ISO/DIS standard 26262, Nov 2011. International Standard.

[15] John Rushby. A comparison of bus architectures for safety-critical embedded systems. (NASA/CR-2003-212161), 2003.

[16] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Systems*. Real-Time Systems Series. Springer, second edition, 2011.

[17] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[18] I. S. Organization. www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en, 2018.

[19] I. S. Organization. www.sae.org/standards/content/arp4754a/, 2018.

[20] A. Piovesan and J. Favaro. Experience with iso 26262 asil decomposition. *Proceedings of the Automotive SPIN Italia Workshop*, 2011.

[21] R. SC-205. Software considerations in airborne systems and equipment certification. RTCA/DO-178C, http://www.rtca.org, Dec. 2011.

[22] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE Real-Time System Symposium (RTSS 2007)*, pages 239–243, December 2007.

[23] D. Ward and S. Crozier. The uses and abuses of asil decomposition in iso 26262. *Proceedings of the Society of Automotive Engineers World Congress*, 2012.

[24] P. Wu. Preventing interference between subsystem blocks at design time, 2015. US8938710B2.