I/O Sharing in a Multi-core Kernel for Mixed-criticality Applications

Gang Li and Søren Top

Mads Clausen Institute for Product Innovation, University of Southern Denmark
{gangli, top}@mci.sdu.dk

Abstract. In a mixed-criticality system, applications with different safety criticality levels are usually required to be implemented upon one platform for several reasons(reducing hardware cost, space, power consumption). Partitioning technology is used to enable the integration of mixed-criticality applications with reduced certification cost. In the partitioning architecture of strong spatial and temporal isolation, fault propagation can be prevented among mixed-criticality applications (regarded as partitions). However, I/O sharing between partitions could be the path of fault propagation that hinders the partitioning. E.g. a crashed partition generates incorrect outputs to shared I/Os, which affects the functioning of another partition. This paper focuses on a message-based approach of I/O sharing in the HARTEX real-time kernel on a multi-core platform. Based on a simple multi-core partitioning architecture, a certifiable I/O sharing approach is implemented based on a safe message mechanism, in order to support the partitioning architecture, enable individual certification of mixed-criticality applications and thus achieve minimized total certification cost of the entire system.

Keywords: I/O sharing, multi-core systems, mixed-criticality, safety-critical real-time kernel, safe inter-core communication

1 Introduction

A computer-based system in critical applications should guarantee to be safe, without making harm to humans or environment around, even in case of the corruption of some certain parts of safety-critical applications. In the real life, applications have different natural criticality levels. E.g. a KONE escalator [1] comprises a dual-channel safety-critical control logic system (an application) and non-safety-critical systems such as display. The criticality level of an application is defined as a Safety Integrity Level (SIL, a level of risk reduction) in the IEC 61508 [1], ranging from the least dependable SIL 1 to the most dependable SIL 4. The certification cost of an application with a SIL is in the direct proportion to the SIL since the development and certification of higher SIL applications has more rigorous requirements. When a system integrates different SIL applications

[1] http://www.kone.com

on one platform with resources sharing(processor, memory or I/O) and no isolation mechanism is taken into account, all the applications have to be certified to the highest SIL, aiming at ensuring the dependability of the highest SIL applications. Otherwise, lower SIL applications with higher failure probability would corrupt higher SIL applications. The entire system being certified to the highest SIL definitely leads to unacceptable increase in the development and certification cost. The basic approach introduces partitioning between mixed-criticality applications in terms of logical and temporal behaviour, which can prevent the propagation of a number of faults between applications. Each application is presumed to reside on a partition in the system.

Besides the traditional partitioning approach (federated architecture), the ongoing partitioning trends are to share a processing element for different SIL applications, or implement them on individual cores on a multi-core platform or the combination of both, by using partitioning mechanisms that provides isolation between applications. E.g. ARINC651 [2] standards proposed Integrated Modular Avionics (IMA) architecture which enforces system-level spatial and temporal isolation to integrate mixed-criticality applications onto a processor. [3] implements mixed-criticality applications into different isolated cores on the trusted MPSoC platforms. Due to such partitioning mechanisms, different SIL applications on one processing element or platform can be certified individually according to their own SILs, and the total cost of development and certification is subsequently reduced. This is the exact objective of the ARTEMIS project Reduced Certification Costs Using Trusted Multi-core Platforms(RECOMP) [2].

Besides performing partitioning mechanisms regarding processors and memory, I/O partitioning is also a big challenging issue when I/Os have to be shared by mixed-criticality applications. Firstly, mixed-criticality applications are not completely isolated if partitions can operate on shared I/Os directly. Shared I/Os are usually related to the functions of both high SIL applications and low SIL applications. A low SIL application could fail and possibly masquerade as a high SIL application to perform incorrect input or output on its accessible I/Os, which therefore results in the failure of the high SIL application. Additionally, even if the low SIL application doesn't masquerade as the high SIL application, it also possibly puts I/Os out of operation and thus the high SIL application is unable to perform its critical operations. Secondly, even if each partition works well from its own point of view, the I/O operation could also fail. As seen in Fig.1, a safety-critical application (Partition 1) reaches the end point of its statically-allocated time slice and has to handover I/O resources to another partition when it's in the critical section of operating on the shared I/Os. This could be unacceptable for I/O operations. Furthermore, another application (Partition 2) takes the turn and possibly changes the I/O configuration or perform its desired operations on I/Os. In the next main frame (a periodical interval), Partition 1 obtains the turn again and continues to perform the rest of the operations. This raises the problem that Partition 2 maybe has modified the I/O configuration or has performed some input or output in the critical sec-

[2] http://www.recomp-project.eu

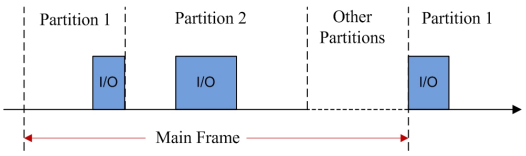

Fig. 1. Preempted I/O access by partitions

tion of Partition 1, which affects the functioning of I/O operations or disturbs the I/O operations (preempted I/O access) in Partition 1. This definitely makes Partition 1 non-functional.

In this paper, Section 2 discusses safety requirements of message-based I/O sharing approach on the basis of safety standards and our experiences. Section 3 introduces a simple multi-core partitioning architecture and then Section 3 investigates the design of message-based I/O sharing approach in our real-time kernel (HARTEX) [4]. Section 5 gives an example how this I/O sharing approach is used in an industrial safe stop component. Section 6 discusses related work and Section 7 concludes.

2 Safety requirements

To enable the integration of mixed-criticality applications onto a multi-core platform, robust partitioning [5] is employed to provide an assurance of intended isolation of independent applications (partitions) implemented upon one platform. Regarding I/O sharing in the partitioning, informal requirements at the architectural level are proposed:

1. A partition must be unable to masquerade as another partition to operate on shared I/Os.
2. A partition must be unable to affect the shared I/O operations of another partition in terms of logical and temporal behaviour.
3. A partition must be unable to affect the logical and temporal behaviour of another partition through shared I/Os.
4. Any attempt of the incorrect access to I/Os must not lead to a unsafe state and can be detected and recognized, and results in a corresponding action to handle the violation.

Note that the second requirement enforces that a partition can't affect I/O operations of another partition, since the partition probably changes the I/O configuration without notifying another partition or disturbs the I/O operations. The third requires a partition to be isolated from another partition even if using shared I/Os.

Since message passing is used to transfer data between partitions inside one core or across multiple cores for I/O sharing, a set of safety requirements shall be proposed to ensure safe communication according to the safety standard IEC61508.

1. Protection shall be provided to message-based communication for detecting message loss, repetition, insertion and resequencing.
2. Protection shall be provided to message-based communication for detecting message data corruption, ensuring data integrity.
3. Protection shall be provided to message-based communication for detecting message transmission delay, ensuring temporal behaviour.
4. Protection shall be provided to message-based communication for detecting message masquerade, ensuring message correct identification recognition.
5. The message-based communication shall support mixed-criticality levels.
6. The message-based communication shall support the interaction across mixed-criticality partitions.
7. Any fault of message-based communication shall not leads to a unsafe state and can be handled after being detected.

The first to the fourth requirements ensures a safe message-based communication that is the base of the I/O sharing. The fifth one requires that different SIL communication services are provided to serve different SIL applications. The sixth one enables the communication between mixed-criticality partitions, which is required by the nature of applications. E.g. a high SIL control application needs to send control results to a low SIL display application.

3 System design

To fulfill the requirements presented above and achieve an easier certification, one rule of thumb should be kept in our mind while designing the partitioning multi-core system architecture and I/O sharing in the HARTEX real-time kernel: simplification.

3.1 System architecture

The proposed system architecture supports two levels of partitioning in the context of a configurable multi-core platform(e.g. System on Chip (SoC) platform): inter-core level and intra-core level as shown in Fig. 2. At the inter-core level, physical separation of processing cores is exploited to a great extent for core independence and isolation. Each core has a private memory(or a virtual private one), and the multi-core HARTEX kernel employs the asymmetric multi-processing (AMP) multikernel architecture proposed in [6]. This multi-core architecture proposed is similar to a federated architecture but residing on a chip, in order to achieve relatively effortlessly-certifiable partitioning and exploit the methodologies and frameworks used in distributed systems. Additionally, each core support intra-core partitioning similar to IMA if mixed-safety applications have to be allocated into one processing core. However, the intra-core partitioning is surely of more certification cost since it shares more resources and complicates the architecture of the kernel and hardware (involving Memory Management Unit or Memory Protection Unit). Therefore, partitions with different SILs are recommended to be allocated respectively into isolated cores at inter-core level. If the

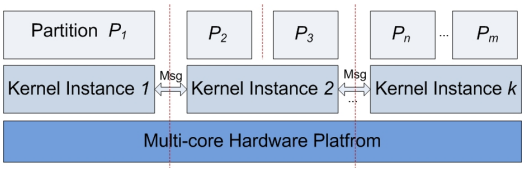

Fig. 2. System architecture

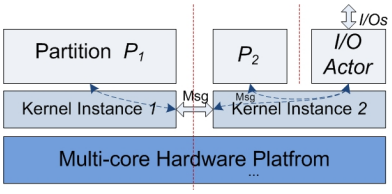

Fig. 3. I/O sharing approach

inter-core isolation can satisfy the system partitioning needs, intra-core isolation is no longer needed, avoiding the certification cost of intra-core isolation design. Therefore, the intra-core isolation mechanism is designed as a configurable component in the kernel. A violation handling mechanism of partitioning is required to process violations in a unified manner.

In this two-level partitioning architecture, I/O sharing among partitions can be addressed by a simple message-based approach, as shown in Fig.3. Each shared I/O has a dedicated actor that can perform all possible operations on the I/O. Tasks belonging to different partitions that require to share the I/O are no longer able to access the I/O directly. The I/O is only accessible to its hosted actor. The I/O with its actor works in an isolated partition and the other partitions can perform their desired operations on the I/O by sending a request via a safe message. The actor works as a server and processes all the requests buffered in a waiting queue when the actor partition gets its turn to run. All the execution of this approach is determined at the system integration time. This approach has two advantages. Firstly, less software has to be certified to the highest SIL among the mixed-criticality partitions. Each partition is isolated from the I/O partition and other partitions, so the low SIL partitions are just certified to their corresponding SILs. Of course, I/O actor has to be certified to the highest SIL, which is essentially inevitable. Secondly, this is a simpler solution for I/O sharing on a multi-core platform, comparing to multi-core mutual exclusion management. Therefore, it's easier to certify.

4 Kernel design for I/O sharing

I/O sharing in the HARTEX multikernel enforces the existence of a safe message passing manager and a certifiable I/O resource manager. A message across partitions has to be validated to a level of dependability by applying a set of

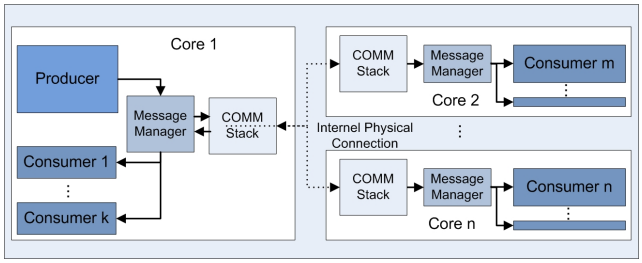

Fig. 4. Producer-consumer communication in the HARTEX multikernel

measures. The I/O sharing needs to abstract all I/O operations into a server model for message-based requests coming from the other partitions.

4.1 Safe message passing

Fig.4 presents the communication mechanism of the HARTEX multikernel in general. Each core has a message manager and a communication stack (COMM stack) for safe intra-core and inter-core communication. The message manager and the communication stack are dedicated to implement the safety layer and communication protocol respectively. The message manager takes the responsibility of managing messages and applying safety-related communication measures. If a message passing failure occurs, the message manager reports the failure to the Violation Manager in the HARTEX multikernel, which handles all kinds of violations in a unified manner. Additionally, the message manager pushes incoming requests of I/O sharing into I/O waiting queue. The communication stack is a typical layered network protocol for embedded systems, which takes care of message routing for intra-core and inter-core communication. It also provides timed-triggered communication that enables the communication subsystem to transfer a message at a specific time instance.

The communication process is fully controlled by the trusted kernel executing in the supervisor mode. The message manager provides a number of safety measures in the kernel space, listed in Table 1. A message can be configured to have several of these safety measures applied according to its SIL requirement. E.g. Cyclic Redundancy Check, Check Sum and Message Acceptance Filtering measures can be applied to one message, by putting these measures IDs into the message configuration. More safety-related measures can be added in the HARTEX multikernel. From Table 2, all the potential communication faults presented in IEC 61508 can be alleviated by the listed safety measures. Note that the pad in the table means all the measures labelled by pads should be applied together to cover this kind of faults.

4.2 I/O sharing management

In a multi-core system, some I/Os are dedicatedly connected to a specific core, but some can be connected to several cores (E.g. a LCD I/O interface connected

Table 1. safety communication measures in the HARTEX

Exploitation of interconnection hardware redundancy	Cyclic Redundancy Check
	Message via dual channels
Exploitation of time redundancy	Check sum
	Double-sending of a message
Other measures	Message Acceptance Filtering
	Message sequence index
	Timed Message Scheduling

Table 2. Communication faults are coved by specific safety measures

Failure types	Hardware redundancy	Time redundancy	Message filtering	Sequence index	Time-triggered scheduling
Repetitions				√	
Deletion				√	
Insertion				√	
Resequence				√	
Corruption	√	√			
Delay					√
Masquerade	•	•	•		

to multiple cores by a common bus). This raises issues from the multi-core point of view. Firstly, some dedicated resources possibly need to be accessed by the other cores. Secondly, multiple cores could use a shared exclusive resource concurrently. This contention can reduce system scalability and performance, since in the conventional way all the relevant states of a resource (E.g. free, being occupied) should be maintained for state consistency among all the cores. This complicated interaction causes the system to be of higher certification cost. Moreover, partitioning is not taken into account while sharing I/Os.

Our approach that improves the localizability of shared resources has been proposed. Each shared I/O is designed to form a partition on a specified core, and all the possible operations on the shared I/O are only performed by this partition. A client-server model for a shared I/O is introduced such that the operations on the I/O of all the other partitions (clients) are achieved by sending request messages to the specified I/O partition. An I/O actor (as a server) integrated with the resource manager on the host specified core receives request messages, validates the messages, performs corresponding operations on the shared resource, and optionally feedbacks operation results to the requesting cores. The I/O actor is isolated from the other partitions and the message-based requests are guaranteed by the mentioned safety measures applied in the mes-

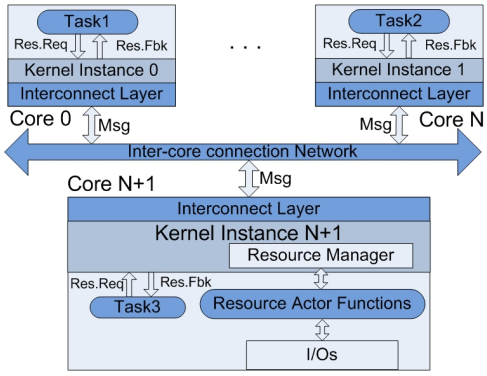

Fig. 5. Server-Client model of shared resource access mechanism

sages. Additionally, the upper limit of the number of requests from a partition in one Main Frame is predetermined in the actor partition as well as specific services of I/O operations that could be used by the partition. Fig. 5 illustrates that tasks on the same core or on different cores can send I/O requests to the resource server via the local message manager. The execution of the actor in the separation kernel is dependent on the allocated time slices of the actor partition.

This approach has several advantages. Firstly, the relevant states of shared resources are not necessary to be replicated among multiple cores. This approach directly facilitates higher system scalability and simplifies the system architecture. Secondly, an I/O resource accessible only to a core are enabled to be accessed by the other cores in a uniform manner by using this approach. Thirdly, the core-level subsystem with localized I/Os has less interference with the other cores logically besides the validated message requests, which facilitates the partitioning architecture. Therefore, the system abstraction model is more understandable and also easily analysed and certified. In the end, one crashed partition which could use a shared I/O is unable to affect the I/O resource utilization by the other cores, if the I/O actor partition works well and I/O message-based requests are well-validated. Tasks with different SILs are enabled to access the I/O without hindering the partitioning architecture, when the I/O actor is certified to the highest SIL of these tasks.

To achieve the server-client model of resource sharing, each I/O resource needs to be abstracted into an I/O actor model, which provides clients with all the operation functions. These functions are invoked by the resource manager according to the validated requests. E.g. a monitor can be simply abstracted into four functions: Monitor_config() function that can initialize and configure the monitor, Monitor_on() function that can turn on the monitor, Monitor_off() function that can turn off the monitor and Monitor_display() function that can display values on the monitor. Each function can get value parameters of a configurable size in the message-based request. All the functions corresponding

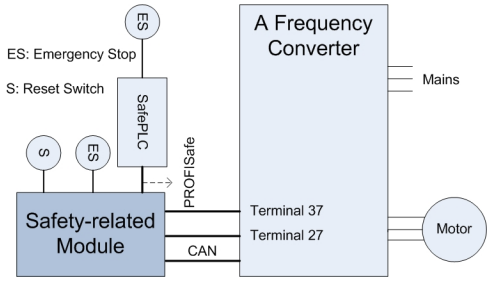

Fig. 6. A safety-related module for a frequncy converter

to different operations have to take the I/O current configuration and states into consideration before executing its desired operations.

5 A case study: a safety-related module for a frequent converter

There is a typical safety-critical application from industrial domain [7]: a safety-related module for a frequency converter used to control the speed of an electrical motor. E.g. controlling a rotating blade in a manufacturing machine. The failure of stopping the motor safely results in the harm to people or equipments. Therefore, a safety-related module implementing a safe stop is highly required. In Fig.6, the inputs of the safety-related module are a safe field bus (PROFISafe) and the local emergency stop button. In addition, a reset switch is used to recover the module into an initial state. According to standard IEC 61800-5-2, a safe stop can be ensured by two functions Safe Torque Off and Safe Stop 1. The safety function Safe Torque Off is achieved through the safety-related interface terminal 37 which can remove the power from electronics and afterwards the motor coasts. The Safe Stop 1 is to request an internal function of the frequency converter to ramp down the speed of the motor via the terminal 27. Both functions perform in conjunction to achieve a safe stop.

To achieve SIL 3 according to IEC 61508, 1oo2D structure [1] is selected for acquiring the hardware fault tolerance of 1, which has two redundant isolated channels with mutual diagnostics. An initial model of the safety-related model in a high abstraction level is proposed in Fig.7. The model only takes care of the high level components, their implementation in hardware or software, and the interaction. It's represented intuitively by the extended safety architecture taxonomy [7]. Black color represents components implemented in hardware and grey color means in software. Solid line represents safety-related components, dotted line represents diagnostics components and dashed line represents non-safety-related components.

As illustrated in Fig.7, only one multi-core chip comprising three isolated cores is used to implement the two safety-related channels and a non-safety-related channel respectively. Core 1 performs one safe-related channel and diag-

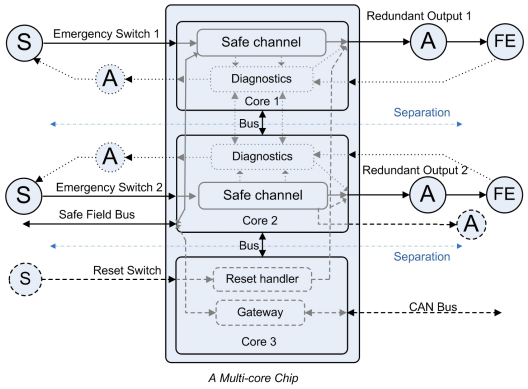

Fig. 7. A high-level abstraction model of a safety-related module for a frequncey converter on a multi-core chip

nostics, and similarly Core 2 performs the redundant safe-related channel and diagnostics. Core 3 executes the non-safety-related reset handler as well as the non-safety-related gateway. "A" in the grey dotted circle in the left are actuators that monitor their own emergency switches whether they are in the operational state or not, and "A" in the black line circle in the right are actuators that react to the external environment. Here we focus on the partitions on Core 1 as an example, aiming at investigating the sharing of the output port in the partitioning architecture. The safe-related channel (a component) can execute functional operations and the diagnostics component monitors the channels, sensors and actuators, which naturally constitutes two partitions. Both partitions share the redundant output 1 but have to be isolated. A fault taking place in the safe channel partition can not be propagated to the diagnostics partition, in order to avoid that the fault affects the judgement of the diagnostics partition and subsequently it is possibly undetectable.

In Fig.8, three partitions should be implemented upon Core 1: Safe Channel partition, Diagnostics partition and I/O partition. The Safe Channel partition performs the safety-related functions and sends safe messages to the Diagnostics and I/O partitions. The Diagnostics partition checks the correctness of the safe channel output, as well as the states of its objective sensor and actuator. If any fault takes place, the Diagnostics partition sends safe messages to the I/O partition to interact to the actuator, aiming at switching the system into a safe state in case of the fault. All the possible operations on the output 1 are abstracted into a well-defined model including the functional operations and handlers in case of different faults. This architecture to a great extent simplifies the development and certification of this case study.

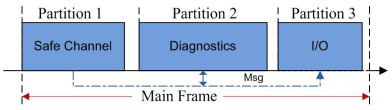

Fig. 8. I/O sharing in a safety-related module for a safe stop

6 Related work

Exclusive shared I/O operations are usually executed under the protection of mutual exclusion management, which is widely used in real-time operating systems. However, the management of shared I/Os has to take into account the isolation in a partitioning architecture that comprises mixed-criticality applications. The traditional I/O sharing approach Time division multiple access (TDMA) to I/Os can not fulfil the isolation requirements between partitions since it only solves temporal isolation.

I/O virtualization is a new approach that ensures the division of I/O, where each virtual machine (partition) with its own system image instance can operate on I/Os independently. In the embedded system world, the XtratuM hypervisor is in charge of providing virtualization services to partitions, and all I/Os are virtualized as a secure I/O partition which can receive and handle I/O operation requests from the other partitions [8]. This is very similar to our separation kernel but still a little different. The XtratuM hypervisor is a virtualization layer that's prone to high-performance applications comparing to the HARTEX multikernel. The partition instance of the XtratuM can be a bare-machine application, a real-time operation system or a general-purpose operation system. The HARTEX multikernel focuses on fine-grained systems and its partition instance is a set of basic executable tasks. Therefore, the HARTEX multikernel leads to a small size of code (4300LOC) and simple architecture, and thus has less certification cost. It has advantages when it comes to small applications. Additionally, the HARTEX multikernel supports multi-core architecture with safe communication. A strongly partitioned real-time system in [9] proposes the Publish-Subscribe architecture in a microkernel for I/O sharing. Partition has the Pseudo-Device Driver to access the I/Os by sending requests to the microkernel. The microkernel layer has device queues to buffer requests, physical device drivers and a device scheduler to handle the requests. This benefits the system with high I/O bandwidth since I/Os can be operated in all authorized partitions. However, the approach results in poor portability and higher certification cost, due to the fact that, besides more complicated design, the code size of the kernel increases since the I/O drivers are added into the kernel and have to be certified together with the kernel. Once moving the kernel to another platform, the entire kernel has to be certified again with the drivers of the new platform, which is required by the safety standard IEC 61508, and thus leads to poor portability.

7 Conclusion

This paper targets simplification and reduced certification cost of mixed-criticality applications on a multi-core platform in the context of I/O sharing. This is a part of our deliverables for the RECOMP project. According to guidelines in the IEC 61508, this paper contributes a set of requirements to enable I/O sharing in a mixed-criticality system. Based on these requirements, a simplified partitioning multi-core architecture has been proposed with several advantages such as physical isolation between inter-core partitions, easy development and reduced certification. Furthermore, the safe message-based communication in the kernel can guarantee safe requests of accessing I/Os in the I/O partition with different levels of dependability in the kernel space. The simple I/O sharing approach has been fully explored to support mixed-criticality partitions without breaking the partitioning architecture. Therefore, a mixed-criticality system of isolated partitions that have to share I/Os can be allocated into one platform and partitions can be certified individually according to their own SILs and consequently the total certification cost is reduced. Since taking the overall consideration of hardware and software architecture, our I/O sharing approach in the context of partitioning systems is simple, flexible and certifiable.

References

1. "Functional safety of electrical/electronic/programmable electronic safety related systems," 2010.
2. "Arinc specification 651: Design guidance for integrated modular avionics," 1991.
3. R. Ernst, "Certificationn of trusted mpsoc platforms," in *10th International Forum on Embedded MPSoC and multi-core*, 2010.
4. C. K. Angelov, I. E. Ivanov, and A. Burns, "Hartex: a safe real-time kernel for distributed computer control systems," *Softw. Pract. Exper.*, vol. 32, pp. 209–232, Mar. 2002.
5. "Integrated modular avionics (ima) development guidance and certification considerations," 2005.
6. A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: a new os architecture for scalable multicore systems," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, (New York, NY, USA), pp. 29–44, ACM, 2009.
7. J. Berthing and T. Maier, "A taxonomy for modelling safety related architectures in compliance with functional safety requirements," in *Proceedings of the 26th international conference on Computer Safety, Reliability, and Security*, SAFECOMP'07, (Berlin, Heidelberg), pp. 505–517, Springer-Verlag, 2007.
8. M. Masmano, S. Peiro, J. Sanchez, J. Simo, and A. Crespo, "Io virtualisation in a partitioned system," in *Proceeding of the 6th embedded real time software and systems congress*, 2012.
9. R. Shah, Y.-H. Lee, and D. Kim, "Sharing i/o in strongly partitioned real-time systems," in *Proceedings of the First international conference on Embedded Software and Systems*, ICESS'04, (Berlin, Heidelberg), pp. 502–507, Springer-Verlag, 2005.