
Proceedings Paper:
Jiang, Zhe, Audsley, Neil orcid.org/0000-0003-3739-6590, Dong, Pan et al. (3 more authors) (2020) MCS-IOV:Real-time I/o virtualization for mixed-criticality systems. In: Proceedings - 2019 IEEE 40th Real-Time Systems Symposium, RTSS 2019. 40th IEEE Real-Time Systems Symposium, RTSS 2019, 03-06 Dec 2019 Proceedings - Real-Time Systems Symposium . IEEE , CHN , pp. 326-338.

https://doi.org/10.1109/RTSS46320.2019.00037

MCS-IOV: Real-Time I/O Virtualization for Mixed-Criticality Systems

Zhe Jiang[1], Neil Audsley[1], Pan Dong[2], Nan Guan[3], Xiaotian Dai[1], Lifeng Wei[4]

[1]University of York, United Kingdom
[2]National University of Defense Technology, China
[3]The Hong Kong Polytechnic University, China
[4]Kylin Information Technology, China
{zhe.jiang, neil.audsley}@york.ac.uk

***Abstract*—In mixed-criticality systems, timely handling of I/O is a key for the system being successfully implemented and functioning appropriately. The criticality levels of functions and sometimes the whole system are often dependent on the state of the I/O. An I/O system for a MCS must provide simultaneously isolation/separation, performance/efficiency and timing-predictability, as well as being able to manage I/O resource in an adaptive manner to facilitate efficient yet safe resource sharing among components of different criticality levels. Existing approaches cannot achieve all of these requirements simultaneously. This paper presents a MCS I/O management framework, termed *MCS-IOV*. MCS-IOV is based on hardware assisted virtualisation, which provides temporal and spatial isolation and prohibits fault propagation with small extra overhead in performance. MCS-IOV extends a real-time I/O virtualisation system, by supporting the concept of mixed criticalities and customised interfaces for schedulers, which offers good timing-preditability. MCS-IOV supports I/O driven criticality mode switch (the mode switch can be triggered by detection of unexpected I/O behaviors, e.g., a higher I/O utilization than expected) and timely I/O resource reconfiguration up on that. Finally, We evaluated and demonstrate MCS-IOV in different aspects.**

I. Introduction

Safety-critical systems play an important role in many medical and industrial fields (e.g. automotive, aerospace, etc.) [27], [45], [68]. In safety-critical systems, integrating components with different levels of criticalities (e.g. Automotive Safety and Integrity Levels (*ASILs*) in ISO 26262 [41].) onto a shared hardware platform has become increasingly important [23], [26], [30]. This results from the diverse functionalities required by modern safety-critical systems (e.g. automated driving [49]), and the rapid evolution of executed platforms [65]. Such systems are called *Mixed-Criticality Systems (MCS)s* [26].

Traditionally, the MCS design enforces strong temporal, spatial and fault isolation among software components of different criticality levels, in order to prevent interference from a lower criticality part to disrupt functionality, timing and performance of a higher criticality part [26]. However, strong isolation usually leads to huge resource waste and poor performance [44]. In recent years, a significant trend in MCS design is to provide certain form and level of resource sharing (e.g., virtualization technology [64]) while keeping sufficient temporal, spatial and fault isolation [53], [55].

In MCS, meeting real-time constraints is usually a key requirement. According to safety assessment methods, e.g., Hazard Analysis (*HA*), Failure Modes and Effects Analysis (*FMEA*), and Fault Tree Analysis (*FTA*), the failure of meeting a deadline from a safety-related task can cause a hazard, which may lead to a system-wide failure and eventually cause an accident [42], [63], [67]. For example, the deadline miss of a braking operation in an electric vehicle may cause a severe car crash. In summary, a MCS usually should meet the following requirements simultaneously: (i) **Isolation**, (ii) **Performance/Efficiency**, (iii) **Timing Predictability**.

In the context of real-time systems, a widely studied MCS model assumes that the Worst-Case Execution Time (WCET) is estimated with different levels of confidence [22], [26], [39]. To certify the timing correctness of high-criticality tasks, it is required to use WCET estimations with extremely high confidence but typically very pessimistic (obtained by, e.g., static analysis). On the other hand, the timing correctness of low-criticality tasks only requires to be validated using WCET estimation with relatively lower confidence but much less pessimistic (obtained by, e.g., testing) [70]. As for other aspects in MCS design, resource sharing and isolation among tasks of different criticality levels is a dilemma [26]. Complete isolation leads to huge resource waste as the WCET estimations used in the certification of high-criticality tasks is very pessimistic. However, if resource sharing between high-ciriticality and low-ciriticality tasks is allowed, both low-ciriticality and high-criticality tasks have to be certified on the high-criticality level due to the possible interference between them [70].

Adaptive resource management [23], [24] is an effective approach to address the above problem, the common idea of which can be described as follows. The system first executes at the low-criticality mode, in which the scheduling policy is designed under the assumption that the execution time of each task (either high-criticality or low-criticality) does not exceed its low-criticality WCET bound. If this assumption is violated, the system switches into the high-criticality mode, in which the scheduling policy is designed assuming execution time of a

task may exceed its low-criticality WCET bound but does not exceed its high-criticality WCET bound. In the high-criticality mode, the scheduler either drop or only maintain a minimum level of service for low-criticality tasks.

I/O is a vital part of embedded computer architectures [43], [52] to interface with the physical world via sensors and actuators, etc. The I/O behaviour greatly affects the execution time of a computational task. However, I/O issue has been ignored in most existing work on real-time MCS. In this paper, we propose MCS-IOV, a new I/O architecture design for MCS, which can fulfil the above mentioned common requirements of MCS, namely, isolation, performance/efficiency and timing-predictability, as well as support adaptive I/O resource management. More specifically, MCS-IOV has the following features:

- MCS-IOV enables I/O virtualisation, and virtual machines are implemented as independent execution environment for tasks, called *criticality world*. Hence, I/O operations requested from different criticality worlds are isolated in time, space and faults.
- MCS-IOV implements the majority of virtualisation and I/O drivers in the hardware, which offloads system overhead from the software and simplifies the I/O access paths – the I/O performance can be improved compared to native systems.
- MCS-IOV integrates the ready-built real-time I/O system (BlueIO) and customised interfaces for theoretical analysis, which guarantees the timing predictability of I/O operations.
- MCS-IOV provides a hardware-based system mode manager, which supports (i) *I/O-driven* mode switch by monitoring the I/O status and trigger mode switch timely (instead of waiting until the overrun actually happens) and (ii) adaptive I/O resource management up one that.

There have been existing work on MCS design and analysis considering the I/O issues. However, MCS-IOV is the first systematical solution that fulfils all the three common requirements of MCS (isolation, performance/efficiency and timing predictability) as well as supports adaptive I/O resource management. Moreover, MCS-IOV supports I/O-driven mode switch. In existing work, a mode switch is triggered by detection of actual task overrun. With MCS-IOV, the mode switch can be triggered by detection of unexpected I/O behavior (e.g., a higher I/O utilization than expected) to enable the system early detect potential overrun and better satisfy the timing requirement in the high-criticality mode.

The rest of this paper is organised as follows: Section II reviews and discusses the related work on MCS I/O systems in both industry and academia. Section III proposes the design of a mixed-criticality I/O system (MCS-IOV) based on the proposed methodology, followed by implementation details in Section IV. Section V presents the evaluation of the MCS-IOV system. A discussion and conclusions are given in Sections VI.

II. State-of-the-art

A. Industry State-of-the-Art

Currently, most commonly used safety-critical systems in industry are not able to support a MCS execution context due to lack of isolation support. Taking the automotive area as an example, only a single-criticality system can be supported by the latest SoCs targeting safety-critical systems from the world's top 3 (in terms of market share) automotive semiconductor suppliers [13].

NXP Semiconductors [11] have almost all current products with safety considerations. For example, the S32V234 processor, focusing on computer vision and machine learning, supports up to ASIL-B (ISO 26262) applications [9]. In these products, FS-SBCs are proposed specifically for safety [8]. They contain S32x series MCUs (general purpose) and MPC574x series MCUs (body control and gateway) [9], [11]. However, FS-SBCs only support compliance with the third critical level in ISO 26262 (i.e., ASIL-C). They are expected to support the highest criticality level (ASIL-D) within the next two years.

Infineon Technologies [10], are involved in different areas of automotive, such as wireless control, power and security. Among their products, the PRO-SIL family [7] is mainly designed for safe-critical systems, including hardware, software and other supporting features (e.g. for the development process). The PRO-SIL family SoCs provide different methodologies to satisfy a high criticality level (SIL-3 in IEC 61508), e.g., hardware redundancies (symmetric and asymmetric architecture redundancies) [66], software redundancies, external watchdog, core checks, etc. As shown in [66], the introduction of these methodologies brings increased scalability, reduced complexity and decreased power consumption compared to traditional safety-related methodologies. However, a mixed-criticality context sill can not be supported, due to lack support on isolation.

Renesas Electronics [12] essentially classifies their automotive products into three main categories: low power (e.g. sound generator), power efficiency (e.g. engineering control) and high performance (e.g. 3D graphics accelerators) [14]. Without constraints on power and complexity of functionalities, the RHx series (power efficiency category) achieves the best safety features (ASIL C in ISO 26262). At the current stage, their main concern is improving methodologies to satisfy a higher criticality level with less power consumption and better hardware efficiency (e.g. CRC operation, frequency detection, RAM guard, etc.) [14], [15]. However, MCS is not supported [12], [14] currently.

Instead of satisfying the essential requirements of a MCS, the systems proposed from industry mainly focus on standardised safety and security processes [57]. This issue has been recognised and discussed by Graydon and Bate [36], as well as Paulitsch et al. [57].

B. Prototypes for MCS I/O Systems

In order to satisfy isolation requirements in MCS I/O systems, different methodologies have been applied by prototype systems developed in academia.

TDMA-based methodologies. *TDMA* provides temporal isolation. For example, Carvajal and Fischmeister [28] proposed an open-source framework (Atacama) for real-time Ethernet for MCS. Cilku et al. [29] introduced a TDMA-based bus arbitration scheme. Goossen et al. [35] used a TDMA-based approach to schedule concurrent memory requests of the same physical memory. TDMA is efficient in achieving temporal isolation, not only in the MCS I/O system, but in the whole MCS system. However, TDMA-based approaches cannot satisfy the requirements on spatial and faults separation and usually leads to resource waste.

TrustZone-based methodologies. ARM TrustZone was firstly introduced into ARM application processor (Cortex-A) in 2004; recently, ARM released TrustZone-M for the new generation of ARM MCUs (Cortex-M) [59]. This technology is centred 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 [60] and TZDKS [31] proposed MCS architectures using TrustZone technologies, which achieve better system performance compared to traditional TDMA-based approaches. Pinto et al. then proposed Virtual-IO [56] based on LTZVisor, which achieves a MCS I/O system with predictable I/O timing. The main limitations of the TrustZone-based approach are twofold: 1) only two system modes can be supported; and 2) no adaptive I/O resource management upon mode switches at runtime supported.

Virtualisation-based methodologies. In virtualised systems, the virtual machines (*VM*s) are logically isolated, 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. For example, Groesbrink et al. [38] utilised hypervisor-based virtualisation to separate system to independant partitions (i.e. VMs). MultiPARTES [69] and Airbus' MP-IOV [55] used para-virtualisation [64] to establish an I/O virtualization system for a MCS; A ready-built separation kernel (i.e., Quest-V [51]) was extended by Missimer et al. [53] to support I/O virtualisation. 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 I/O requests can be served earlier (i.e., in [37], [38], partitioned RM scheduling of VMs based on periodic servers with fixed period are implemented; in [53], sporadic servers and priority inheritance bandwidth-preserving servers are built to schedule I/O requests among the VMs).

However these approaches involve complicated resource management and complex paths of instructions. For a more detailed analysis and discussion of virtualisation technologies, please see our previous papers [44], [47].

Different from virtualization-based methodologies, Kim et al. [48] proposed a methodology for MCS I/O systems from a different perspective — i.e., operating system (*OS*) level. As highlighted by Kim et al., OS is a largely ignored potential interference of the I/O system in MCS. From OS level, I/O accesses can be abstracted as memory accesses. Hence, they proposed optimised memory management (i.e., isolating dynamic-memory allocations) and improved IPC-related mechanisms (i.e., selective LLC bypass, concurrency elimination, and LLC Locking) to improve I/O performance in MCS and system performance. Drawbacks of this work are the same as the virtualization-based approach. Moreover, isolation provided from OS level is even weaker than the virtualization-based approach [62].

Virtualisation relies on hardware [71], therefore different hardware assistances have been promoted by today's chip manufacturers, in order to alleviate the issues virtualization causes. For example, Intel's Virtualisation Technology for Directed I/O (*VT-D*) provides direct I/O access from VMs [40]. Munch et al. [54] proposed a MCS I/O virtualisation system based on PCIe SR-IOV. The research was also extended in [55] to use other hardware assistance (I/O MMU [25]). However, even with hardware assistance, the I/O performance from the criticality worlds (VMs) cannot reach the I/O performance in a native system. Additionally, commonly used hardware assistance in I/O virtualisation cannot solve the requirement on predictability.

These lead to new challenges of the design of a MCS I/O system. In this next section, we will have an overview of the proposed MCS-IOV system, which could satisfy these requirements. We will later come back to this discussion in a case study in Section V-D.

III. MCS-IOV System Overview

The proposed mixed-criticality I/O system (MCS-IOV) relies on virtualisation and hardware assistance in its design and implementation. In MCS-IOV, the isolation of independent execution environment (i.e., criticality world) is achieved using Virtual Machines [64]. Due to the introduction of virtualisation, the criticality worlds are logically isolated in time and space and for faults. At the same time, each criticality world is assigned a system mode, which can be changed by the system integrator [1] during run-time. Moreover, the introduction of hardware assistance offloads the majority of virtualisation and low-layer I/O drivers onto the hardware and simplifies the I/O access path. Therefore, the software overhead and system performance (e.g. the I/O performance) can be improved, compared to a native system. Meantime, MCS-IOV integrates a ready-built real-time I/O system with customised interfaces for theoretical analysis, which guarantees the predictability of I/O operations (i.e., MCS-IOV hypervisor). The combined design of hardware and software leads to the timely system

[1] System integrator is the person responsible for the integration of the system, who pre-defines the condition of system mode change.

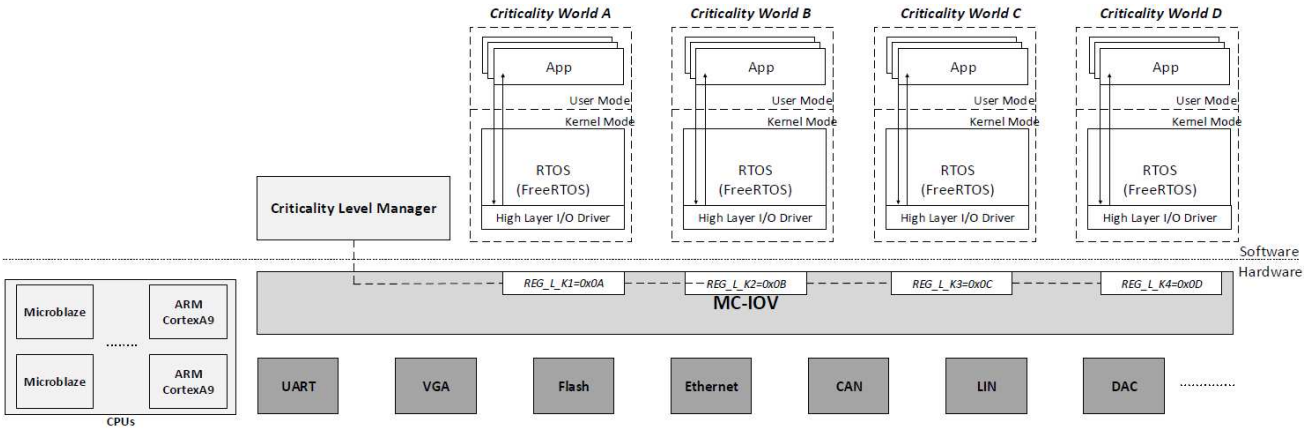


Fig. 1. MCS-IOV Architecture

mode changes being applied to both hardware and software — avoiding criticality level inversion.

A. Context

In industry, a system architecture is often required to comply with safety standards, which is a huge project that may require large numbers of engineers to follow complicated procedures, including safety concept analysis, safety mechanism implementation, verification, quantity analysis and system assessment. Although MCS-IOV is a research prototype, we partially follow ISO 26262 [41]. In the context of ISO 26262, MCS-IOV is able to support four criticality levels, from ASIL A (lowest) to ASIL D (highest). MCS-IOV has a general-purpose design which can be applied to other safety standards and more criticality levels.

B. General Architecture

The proposed architecture of MCS-IOV is shown in Figure 1. In the software layer, the RTOS kernel (e.g. FreeRTOS [6]) in each criticality world is able to executed in kernel mode to achieve full functionality. At the same time, it can provide a real-time environment for applications that need to guarantee deadlines.

In the hardware layer, the MCS-IOV hypervisor is responsible for system virtualisation (including resource allocation), physical isolation between criticality worlds, and providing high-layer access interfaces for user applications in the criticality worlds. In the hypervisor, the of the criticality worlds are stored in dedicated registers, which can be modified by the system mode manager involved in the software layer during run-time. In MCS-IOV hypervisor, customised interfaces are enabled to support scheduling polices coming from new research and analysis.

The architecture uses existing work in BlueIO [45] and BlueTree [34], which are open-source and can be publicly accessed. In our proposed system, BlueIO is used for I/O virtualisation and BlueTree is used for memory accessing.

C. BlueIO

BlueIO is a scalable hardware-implemented real-time I/O virtualisation system [45]. It follows the work of the Virtualised Complicated Device Controller (VCDC) [44], which scalably integrates I/O virtualisation and I/O drivers into the hardware, with additional features of predictability and related analysis. The BlueIO virtualises a physical I/O device to multiple virtual I/O devices, provides access interfaces for software applications and enables predictable I/O operations. BlueIO can be physically connected to a multi-core or a many-core system and is composed of three main parts (see Figure 2):

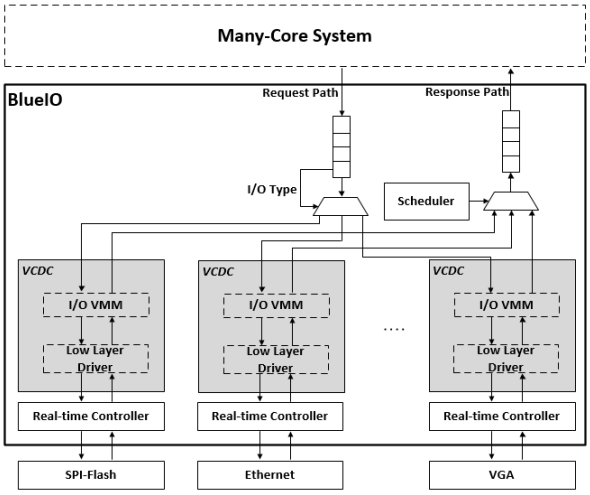


Fig. 2. Structure of BlueIO (simplified and adopted from [45])

- I/O VMM – Maintains the virtualisation of I/O devices. The main responsibility of I/O VMM is translating I/O requests sent from each VM into actual I/O instructions to operate a physical I/O. Considering that the functionalities and features of I/O devices are different, it is difficult to build a general purpose module to achieve

virtualisation for all types of I/O devices. In [45], they propose a specific-purpose I/O VMM for commonly used I/O devices, including UART, VGA, DMA, Ethernet, etc. Further customised I/O VMM can be included in the VCDC via documented interfaces [44].

- Low Layer I/O Drivers – Encapsulate specific I/O drivers for a specific I/O device (e.g. read 16-bit data from a particular address of the SPI flash.)
- Real-time I/O Controller – Binds I/O operations with timing constraints, which allows real-time and timing-accurate I/O operations [46].

D. BlueTree

BlueTree is a tree-like memory interconnect for many-core systems that enables time-predictable memory read/write from a number of CPUs [32], [33]. In this paper, BlueTree is extended for real-time memory access of I/O devices, in order to achieve particular functionalities (e.g. DMA-based Ethernet communication [17]). We will not introduce the implementation of the memory access module in this paper, for more details please see [21], [32], [33].

IV. MCS-IOV Design

MCS-IOV is included in a 2D mesh type open source NoC [61]. The use of a NoC is not required by MCS-IOV, as it is a general-purpose I/O system which is agnostic to the type of bus and the CPUs. To support a complete MCS-IOV system, the platform requires:

- Communication channels between MCS-IOV hypervisor and CPUs;
- A global synchronisation timer;
- Memory access interface. In the proposed design, BlueTree [32] is adopted as the memory access interface (see Section III-D).

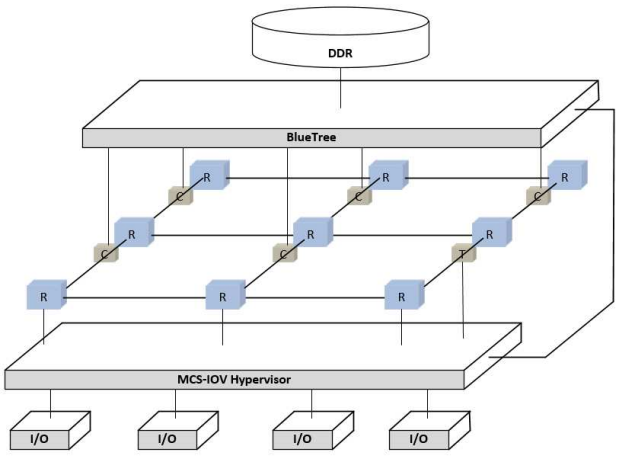


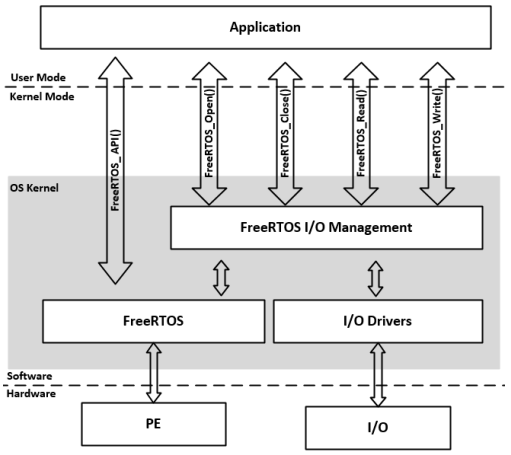
Fig. 3. Platform Overview (C - Processor Core; R - Router/Abiter; T - Global Timer)

The use of MCS-IOV within the NoC is shown in Figure 3. MCS-IOV hypervisor is physically connected to the home port (via the physical link) of a router, the global timer T, and the memory access interface (BlueTree). A complete MCS-IOV system contains both software and hardware layers. In the software layer, the independent criticality worlds and the system mode manager are introduced. The MCS-IOV hypervisor is introduced in the hardware layer.

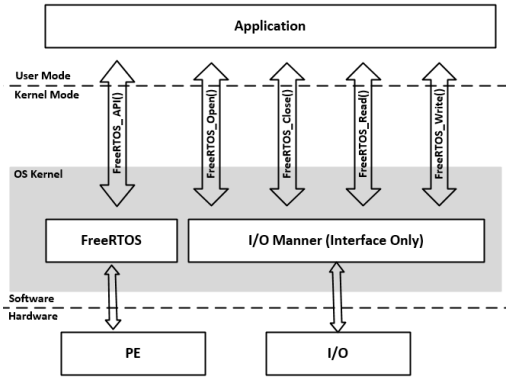
A. Criticality Worlds and Guest OS

In the system, four independent criticality worlds are proposed as virtual machines. The virtualisation has the following features:

- Bare-metal virtualisation [64] – the guest OS in the criticality world can be executed on processors directly, without the host OS. Therefore, a guest OS is able to execute in kernel mode to achieve full privileges.
- Para-virtualisation [50] – the kernel of the guest OS requires modifying in the I/O management, which has to be replaced by high level I/O drivers. This allows a smaller OS software footprint and simplified I/O access paths.



(a) Native System



(b) MCS-IOV

Fig. 4. FreeRTOS in Native System and MCS-IOV

Currently, in our proposed design, three OS kernels have been modified to support I/O virtualisation: FreeRTOS [6], μCosII [16] and Xilkernel [19]. In Figure 4, we use the FreeRTOS kernel as an example to demonstrate the modifications in an OS kernel.

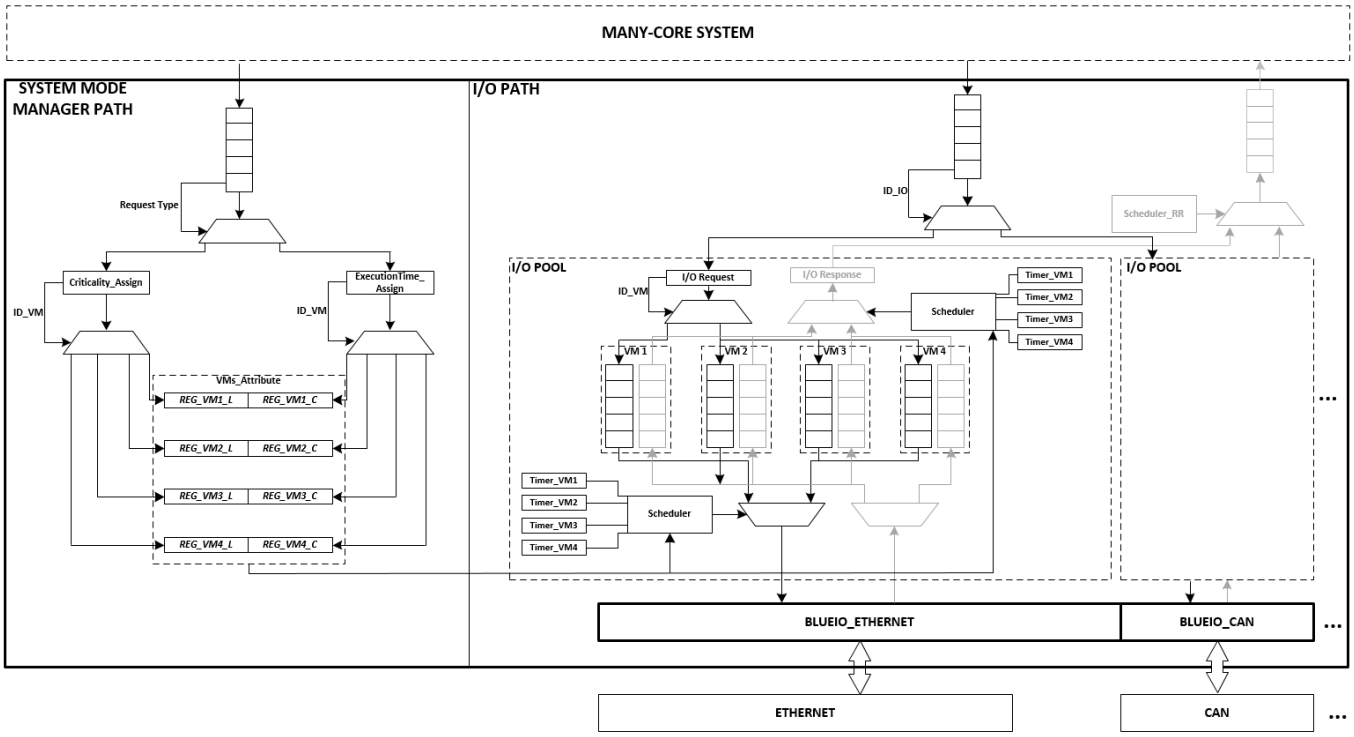


Fig. 5. Structure of MCS-IOV Hypervisor (grey shading indicates the response path)

1) Isolation: In our proposed design, each criticality world (guest OS) independently runs in kernel mode with full privileges and without recognising the existence of other criticality worlds or the hypervisor (MCS-IOV). The OS kernels running independently with full privileges may result in illegal accesses, including illegal address accesses (e.g. criticality world K_1 tries to access an address space belonged to criticality world K_2) and illegal requests (e.g. criticality world K_1 tries to release an I/O being used by criticality world K_2). MCS-IOV hypervisor will detect and block these malicious requests by checking a resource allocation table inside a dedicated module (see Section IV-C). With the assistance of MCS-IOV hypervisor, isolation in space and faults can be achieved. Note, the allocation of hardware resources is decided by the system integrator.

2) Resource Efficiency: Unlike the native OS kernel (Figure 4(a)), user applications in the modified kernel (Figure 4(b)) are able to directly access and operate virtualised I/O devices through the provided high-layer interfaces, without the interposition of the OS kernel. The simplification of I/O access paths and the hardware implementation of I/O drivers significantly improves the resource efficiency with increased system performance and decreased software overhead compared to a traditional virtualised system, and even the native system.

Simultaneously, user applications running on a native OS kernel can be directly ported to the modified kernel in the MCS-IOV system, since the original interfaces on both OS kernels and the (virtualised) I/O devices are kept.

B. System Mode Manager

In the software layer, parallel to the criticality worlds, the system mode manager is also introduced. Different from the criticality worlds, the system mode manager is allocated to a dedicated processor and never uses shared resources such as I/O resources. Its main responsibility is assigning system modes to different criticality worlds during run time and allocating the execution time of each criticality world (see Section IV-C).

1) Run-time System Mode Switch: The system mode of each criticality world is mapped to a corresponding dedicated 32-bit register inside MCS-IOV hypervisor. In the system mode manager, there is only one process running continuously, which is used by the system integrator to modify the system modes of the criticality worlds by updating the corresponding registers. For example, system mode manager can be configured to switch the system mode of a criticality world(s) while detecting the overload of I/O buffers.

2) Limitations: Due to blocked I/O operations by MCS-IOV hypervisor [58], the mode switches of the criticality world can only be issued after each complete I/O operation. Hence, a delay in the mode switches will be introduced. The maximum delay equals the worst-case execution time of the corresponding I/O operation.

C. MCS-IOV Hypervisor

The MSC-IOV hypervisor consists of the following main parts (see Figure 5):

- *System Mode Manager Path*: Provides the interface from the system mode manager (see Section IV-B) to the MCS-IOV hypervisor via the NoC mesh. Inside the system mode manager path, the attributes of each criticality world are stored in independent registers. Specifically, the system modes can be accessed at addresses *0x00*, *0x04*, *0x08* and *0x0C* respectively offset to the MSC-IOV's $BaseAddress$. Moreover, the time server[2] allocated to each criticality world can be configured at addresses $0x10$, $0x14$, $0x18$ and $0x1C$ respectively offset to the MSC-IOV's $BaseAddress$. These two types of attributes can be used as input parameters to the scheduler in I/O pools.
- *I/O Path*: provides the interface between criticality worlds and I/Os. As shown on the right of Figure 5, the I/O path mainly contains the I/O pools and the BlueIO. Specifically, the I/O pool schedules I/O requests and I/O responses to different criticality worlds, whilst BlueIO provides the functionality of I/O virtualisation and encapsulates the corresponding I/O drivers.

1) Configurability: As shown in Figure 5, the I/O pool can be divided into two paths:

- The *Request Path* is responsible for transmitting I/O requests from different criticality worlds to I/O devices.
- The *Response Path* (shown in grey) is responsible for transmitting I/O responses from I/O devices back to the corresponding criticality worlds.

In both paths, an independent scheduler is introduced to schedule the I/O requests/responses to the different criticality worlds. The scheduling policies of the schedulers are configurable, and can be switched between the basic scheduling polices (e.g. Fixed-Priority, Round-Robin) and customised scheduling policies (e.g. dynamic time servers). An interface to the system integrators to implement the scheduling polices based on state-of-art theoretical analysis is also proposed.

2) Predictability: As described in Section IV-C, the I/O path is composed of the I/O pools and BlueIO. As introduced in [44] and [45], BlueIO enables I/O virtualisation with improved predictability compared to a native system. At the same time, the schedulers in each I/O pool support real-time scheduling. Therefore, MCS-IOV significantly increases the predictability of the I/O operations compared to a traditional virtualised system.

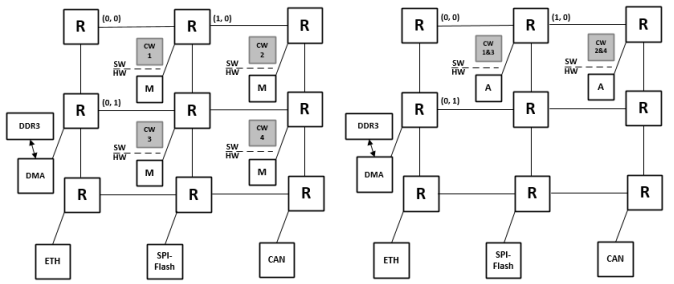
3) Scalability: The design of the I/O pool is generic. Hence, MSC-IOV can be easily expanded to support different I/Os (e.g. CAN, LAN, etc.) by duplicating the I/O pool and then connecting the new I/O pool between the many-core system and the BlueIO.

V. Evaluation

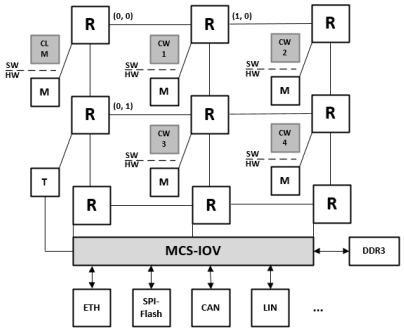
The MCS-IOV hypervisor was implemented using Bluespec System Verilog [3] and synthesised on a Xilinx ZC706 development board [20]. The following evaluation focuses on I/O devices and I/O systems. NB We do not consider the effects caused by the NoC and routing protocols.

In the evaluation, the MCS-IOV hypervisor was connected to a 3×3 2D mesh type open source real-time NoC [61], which contains 4 MicroBlaze processors running independent criticality worlds. The OS kernel running in each criticality world is a modified guest OS kernel (FreeRTOS v9.0.0) described in Section IV-A. The software on MicroBlaze processors is compiled using the Xilinx MicroBlaze GNU tool chain [18]. The architecture is shown in Figure 6(c). To enable a comparison, two baseline systems are introduced (with all architectures running at 100 MHz):

- ***Baseline System – Native (BS-N)*** (Figure 6(a)): The NoC system without hardware assistance. Scheduling related to shared resources is undertaken by the routers/arbiter. Each processor is deemed to be an independent criticality world [26], [70].
- ***Baseline System – TZDKS [31] (BS-T)*** (Figure 6(b)): The MCS built on ARM trust-zone technology. In the system, four independent criticality worlds are implemented in the secure and non-secure worlds of processor (1,0) and processor (2,0) respectively.



(a) Baseline System: Native (***BS-N***) (b) Baseline System: T (***BS-T***)



(c) MCS-IOV System

Fig. 6. Experimental Platforms. R - Router/ Arbiter; M - MicroBlaze Processor; A - ARM Processor; T - Global Timer; CW: Criticality World; SMM: System Mode Manager.

The evaluation of MCS-IOV should mainly relate to the essential requirements of separation, resource efficiency, predictability and run-time system mode switch. Specifically, the separation requirement belongs to functionalities which were introduced in Sections IV-A1 – temporal, spatial and fault isolation can be simultaneously supported.

[2]Time servers are implemented to support some theoretical models, and can be disabled by the system integrator if not required.

TABLE I
HARDWARE OVERHEAD (IMPLEMENTED IN FPGA)

Involved I/O	Supported Criticality Levels	Hardware Consumption							
		LUTs	% of ZC706	Registers	% of ZC706	BRAMs	DSP	Power (mW)	Maxium Frequency (Mhz)
LIN	4	831	0.38%	874	0.20%	0	0	13	324
	8	962	0.44%	1,093	0.25%	0	0	15	318
	16	1,159	0.53%	1,339	0.32%	0	0	16	311
LIN+CAN	4	2,164	0.99%	1,967	0.45%	0	0	23	288
	8	2,426	1.11%	2,317	0.53%	0	0	25	277
	16	2,820	1.29%	2,842	0.65%	0	0	29	260
LIN+CAN +ETH	4	4,416	2.02%	4,416	1.01%	0	0	43	249
	8	4,787	2.19%	4,984	1.14%	0	0	47	236
	16	5,334	2.44%	6,007	1.39%	0	0	55	204

TABLE II
HARDWARE OVERHEAD (IMPLEMENTED IN VLSI)

Involved I/O	Supported Criticality Levels	Hardware Consumption												
		AND	AIO	DEEPOS	HA	INV	MUX	NAND	NOR	OAI	OR	XNOR	XOR	Total
LINX	4	513	2,381	2,188	39	2,782	43	1,929	1,442	1,849	232	18	22	13,438
	8	601	2,649	2,344	40	3,238	45	1,984	1,501	1,887	249	19	24	14,581
	16	694	3,010	2,683	52	3,586	54	2,618	1,822	2,314	314	23	30	17,200
LIN+CAN	4	1,080	5,271	5,281	83	6,112	89	4,401	3,168	4,034	519	37	51	30,126
	8	1,258	6,502	5,808	96	6,793	118	4,731	3,493	4,566	627	44	52	34,088
	16	1,435	6,642	6,360	108	8,028	135	5,782	4,236	5,880	699	57	63	39,425
LIN+CAN +ETH	4	2,594	11,604	9,704	178	13,171	208	8,747	7,074	9,349	1,034	79	114	63,856
	8	2,719	11,984	10,949	217	16,049	214	9,897	7,463	10,419	1,249	103	114	71,377
	16	2,735	12,884	13,328	238	16,767	236	11,959	8,815	10,242	1,319	103	129	78,755

Hence, the evaluated metrics mainly focus on resource efficiency, predictability and run-time system mode switch.

A. Resource Efficiency

The resource efficiency of an evaluated system is classified as both software and hardware overhead and performance, which are evaluated in this section.

1) Software Overhead: In this section, we evaluate the memory footprint of MSC-IOV and the different versions of FreeRTOS running in the three evaluated systems (memory size tools: the Xilinx Microblaze GNU tool chain [18] and the ARM GNU tool chain [1]). In the evaluation, the native version of FreeRTOS (*FreeRTOS_BS-N*) was fully-featured [5], and it is the foundation of the other FreeRTOS kernels. Table 7 presents the collected measurements.

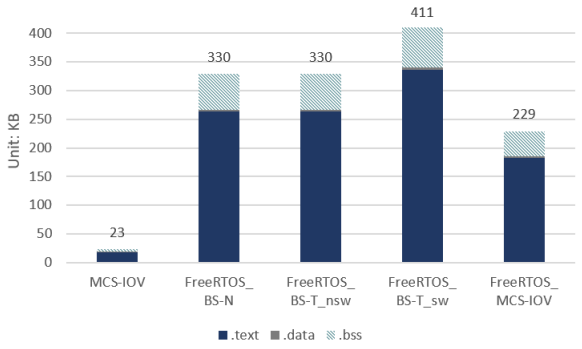


Fig. 7. Memory Footprint of MCS-IOV and RTOS kernels in the Evaluated Systems (nsw: non-secure world; sw: secure world)

As can be seen, an extra 23 KB memory footprint is introduced in MCS-IOV, resulting from the introduction of the system mode manager, which is 7.12% of the native freeRTOS. The native full-featured FreeRTOS (*FreeRTOS_BS-N*) requires 337,616 bytes (≈330KB). The higher memory footprint compared to the official version [6] results from introducing the drivers for the Ethernet, CAN and LIN. When it comes to the BS-T, there is no increment on the OS kernel running in the non-secure worlds (*FreeRTOS_BS-T_nsw*). However, the OS kernel running in the secure worlds (*FreeRTOS_BS-T_sw*) suffers from a 26.77% increment of software overhead, which is mainly introduced by virtualisation and the corresponding monitor [31]. In the MCS-IOV system, the *FreeRTOS_MCS-IOV* only consumes 234,798 bytes (≈229KB) memory footprint, which is 69.4% and 57.8% of the *FreeRTOS_BS-N* and *FreeRTOS_BS-T_sw*, respectively. The main reason behind such a low memory footprint is the implementation of para-virtualisation (see Section IV-A), which significantly removes the software overhead.

2) Hardware Overhead: The BS-N (Figure 6(a)) does not require any extra hardware assistance. However, the BS-T (Figure 6(a)) requires the support of ARM TrustZone technology, which is integrated into all ARM Cortex-A and the latest Cortex-M33 processors [2]. In this section, we evaluate the hardware overhead of MCS-IOV (i.e., MCS-IOV hypervisor) in both FPGA and VLSI.

In the FPGA implementation (Table I), Vivado (v2018.3) is used to synthesise and implement MCS-IOV hypervisor on the Xilinx ZC706 FPGA board [20]. The hardware consumption is mainly summarised as the use of LUTs, registers, BRAMs and power consumption. For the VLSI (see Table II), a Cadence RTL encounter compiler (v11.20) [4] is used to synthesise the MCS-IOV hypervisor into the gate level using the open source MOSIS SCMOS TSMC 0.18μm library (*OSU_SOC_v2.7*).

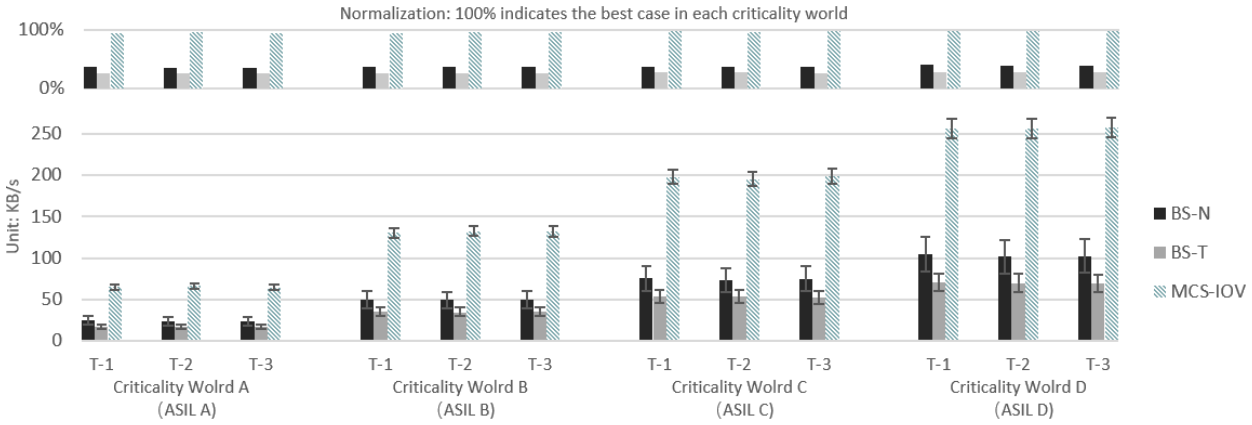


Fig. 8. I/O Throughput (T-1: Task 1; T-2: Task 2; T-3: Task 3)

The hardware consumption is summarised in the use of the essential logic gates.[3] In both evaluations, we increased the number of I/O devices involved and the supported criticality levels respectively in order to show the scaled hardware consumption.

The resource efficiency of MCS-IOV hypervisor is shown in Tables 1 and 2, note that even the MCS-IOV supporting LIN, CAN and Ethernet with 16 criticality levels only requires 2.44% LUTS and 1.39% registers of the ZC706 FPGA board.

Moreover, as shown in Tables 1 and 2, hardware consumption linearly increases with the number of I/Os and CPUs. At the same time, the number of supported I/Os dominates hardware consumption. For example, compared to a 4-criticality system with LIN and CAN (FPGA: 2,164 LUT and 1,967 registers; VLSI: 30,126 gates), an extra 30.30% LUTs and 44.44% registers, or 30.87% gates are required to support an extra 8 criticality levels. However, to support an extra I/O device (Ethernet), an additional 104.04% LUTs, 124.44% registers or 111.96% gates are consumed.

3) Performance: In this section, we evaluate the performance of three architectures via the I/O throughput. The I/O device used in the experiments is an SPI nor-flash (model: S25FL128S). In the criticality worlds of each architecture, three software applications are set to run and continuously read data (32 bits) from the flash. The bytes written from each application per second is recorded as the I/O throughput (KB/s). In each criticality world, Round-Robin is used as the scheduling policy for the software applications. Meantime, the execution time of each criticality world is allocated as 10%, 20%, 30% and 40% (from ASIL A to ASIL D). All experiments were implemented 1,000 times. NB, other theoretical models can be also applied.

As shown in Figure 8, the behaviours of I/O throughput in each architecture are subject to the scheduling policies and time allocations of the MCS context. In the criticality worlds of BS-T, due to the introduction of virtualisation, I/O throughput suffers from approximately 30% reductions compared to BS-N (low resource efficiency). Thanks to the hardware implementation of virtualisation and I/O drivers, in each criticality world of the MCS-IOV, better performance on I/O throughput is always achieved, which is about 300%, and 425%, respectively compared to BS-N and BS-T. Moreover, as shown in the normalised comparison, the average I/O performance in each criticality world of MCS-IOV is always close to the best case, which is more than 95%.

[3]In order to make the numbers readable, the names of the gates are summarised. For example, the 2-entries OR gate and 4-entries OR gate are summarised to the OR gate in the table.

B. Predictability

As shown in the evaluation results of Section V-A3 (see Figure 8), in each criticality world, there is less variance in the I/O performance in the MCS-IOV than the other baseline systems, showing better predictability.

Furthermore, the experiments in this section are designed to measure the I/O predictability by evaluating the Ethernet response time (loop back mode). Specifically, in the criticality worlds of each architecture, three software applications continuously send 1 KB Ethernet packets using the MAC protocol. Information about the I/O response time is recorded and summarised. A lower I/O response time (for both average and worst-case) indicates a higher I/O performance and better resource efficiency, and a lower variance means better predictability. In each criticality world, Round-Robin is used as the scheduling policy for software applications. The execution time of each criticality world is set as 10%, 20%, 30% and 40% (from ASIL A to ASIL D). All experiments were run 1,000 times. NB, other theoretical models can also be applied.

As shown in Table III, among criticality worlds with the same system modes, the I/O operations in the MCS-IOV system always have a shorter response time (in both average and worst cases). This indicates that MCS-IOV has a better resource efficiency than the baseline systems. Furthermore, when it comes to variation in the 1,000 experiments, the I/O response time in the MCS-IOV always has less variation. Specifically, when the criticality worlds are executed in highest critical system mode (ASIL D), the MCS-IOV only suffers from 16.8% and 18.6% variation respectively, compared to

TABLE III
I/O RESPONSE TIME OF LOOP BACK 1KB ETHERNET PACKETS IN EVALUATED SYSTEMS (UNIT: μs)

Citicality World	Baseline System: Native System			Baseline System: TZDK MCS			MCS-IOV		
	Average	Worst-case	Variation	Average	Worst-case	Variation	Average	Worst-case	Variation
VM1 (ASIL D)	72.85	77.43	8.35	78.25	83.51	7.51	22.33	23.22	1.41
VM2 (ASIL C)	164.32	181.82	25.33	180.14	193.32	21.21	51.25	53.52	3.83
VM3 (ASIL B)	250.99	278.49	42.21	282.17	299.02	31.29	77.49	80.21	4.28
VM4 (ASIL A)	351.09	393.63	70.21	378.32	402.03	45.38	103.93	107.33	6.60

BS-N and BS-T. When the lowest system mode (ASIL A) is assigned, these numbers are decreased to 9.4% and 14.5%. These experimental results demonstrate significant improvements in the predictability of the MCS-IOV system.

C. Delay of System Mode Switches

In this section, we evaluate the delay of system mode switches in MCS-IOV (this functionality is not supported in the baseline systems). The experiment setup is based on the experimental configurations in Section V-B. Additionally, instructions of system mode switch on each criticality world are requested randomly during runtime. The time from an instruction request until it can be completely issued was recorded as the delay of system mode switch. The experiments were run 1,000 times.

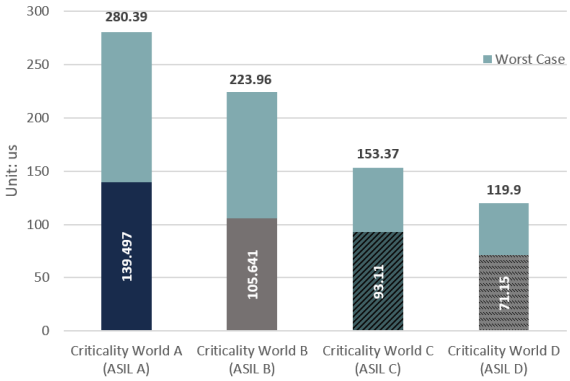


Fig. 9. Delay of System Mode Switches in MCS-IOV

As shown in Figure 9, the average delay of system mode switches among different criticality worlds (from ASIL A to ASIL D) are: 139.497 μs, 105.641 μs, 93.11 μs and 71.15 μs. In criticality worlds with lower system modes, significant variances were seen between the average delay and the worst case: 101% (ASIL A) and 112% (ASIL B). Conversely, in the criticality worlds with higher system modes, the variances decreased to 64.72% (ASIL C) and 68.52% (ASIL D), respectively. Although the delay to system mode switches is controlled at the microsecond level in MCS-IOV, the predictability still requires further improvement in the future.

D. Case Study

In this case study, we consider a 16 PE (Microblazes) real-time NoC, with a 5x5 2D mesh architecture, and a single Ethernet device. Specifically, PE 0 - 7 execute the main functions (i.e., task sets), and PE 8 - 15 generate I/O contentions to the system. The task sets contain 40 high-criticality tasks, and 40 low-criticality tasks. The release period of the tasks are all set to be 20 ms, with deadline equals to period; and the computational execution time of each task is between 0.5 ms and 1 ms. During each execution of the task, a 2 KB Ethernet packet is sent respectively at three time points: 0.25, 0.5 and 0.75 of the task execution time (See Figure 10). N.B., according to 1,000 times experimental results, the average I/O response time of sending a 2 KB Ethernet packets from one of the NoC nodes without any I/O contention is 27.10 us (0.0271 ms) — around 3% of the execution time of a task.

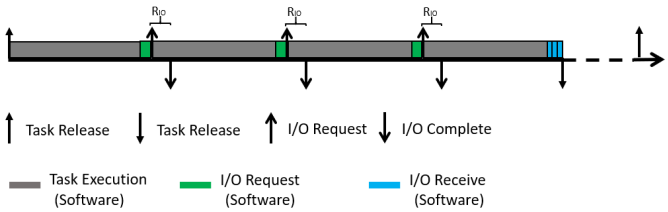


Fig. 10. Example of a Task Repetitively Requesting an Asynchronous I/O

In each experiment, 80 tasks are randomly allocated to PE0 - PE7 (10 tasks per PE). The initial utilisation of each PE is set to be 50% with Round-Robin scheduling policy. Three evaluated system models (i.e. non-MCS, MCS with AMC and MCS-IOV) are respectively executed with different number of I/O contentions (I). In AMC models, the WCET of each task (without I/O contention) is achieved via experimental measurements, and the optimistic estimation of the WCET is set to be 2.5 and 1.5 times of the experimental results respectively in AMC_2.5 and AMC_1.5 [4]. As only two system modes are evaluated, we mapped ASIL A and B as the low-criticality level, and ASIL C and D as the high-criticality level in the MCS-IOV system. Specifically, in the MCS-IOV system, PE 0 and 1 are configured to criticality world A; PE 2 and 3 are configured to criticality world B; PE 4 and 5 are configured to criticality world C; PE 6 and 7 are configured to criticality world D. Furthermore, the system mode manager of the MCS-IOV is configured to switch the system mode of all the criticality worlds by detecting the overall usage of I/O pool being greater than 80% (MCS-IOV_80) and 40% (MCS-

[4]In AMC-x, the WCET of each task is set to be x times of its measured maximum execution time (without I/O contention). Meantime, the system is designed to switched to the high-critical mode while any task being detected to exceed its WCET.

IOV_40) respectively. Each experiment executes 200 ms (10 periods), and all the experiments are performed 1,000 times.

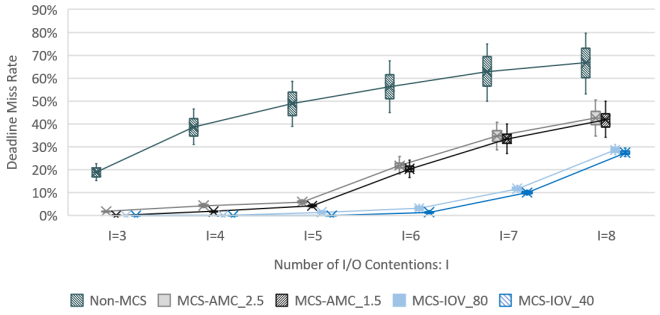


Fig. 11. Case Study: Deadline Miss Rate

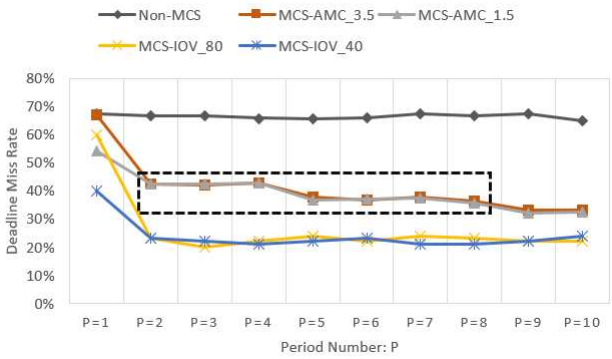


Fig. 12. Case Study: Deadline Miss Rates of Three Systems in Each Period Box: Criticality Level Inversion (Except non-MCSs, System Model Switches Occur in the First Period)

Figure 11 demonstrates the deadline miss rates of the three evaluated models (five systems) with different number of I/O contentions. The experiments shows:

- The 'short' I/O operation can significantly increase the execution time of tasks. For example, in a non-MCS system, the average deadline miss of the system increases to 66%, when $I = 8$.
- Compared to a non-MCS, conventional MCS (i.e., AMC) can somehow alleviate the issue. However, with the number of I/O contentions increasing, the deadline miss rate also increased dramatically. Specifically, when $I < 5$, the deadline miss rate is around 5% (mainly caused by criticality level inversion); when $I > 5$, the deadline miss rate increased dramatically (mainly caused by I/O contention).
- Schedulability of MCS-IOV is significantly improved compared to conventional methodologies, because of enhanced I/O performance and hardware support of run-time system mode switches. Specifically, enhanced I/O performance decreases the impact of I/O to task execution time; and the hardware support of run-time system mode switches effectively avoid the criticality level inversion.
- In MCS-IOV systems, as the number of I/O contention increases, the schedulability of the system will be significantly decreased in a certain situation ($I = 7$). This is the maximum capacity of the system.

In order to demonstrate the criticality level inversion clearly, Figure 12 specifies the average deadline miss rate of each evaluated system in each period while the number of I/O contention equalling to 8 ($I = 8$).

As shown in Figure 12, AMC-based systems and MCS-IOV systems are all switched to the high-critical mode in the first period. In AMC-based systems, the deadline miss rates can be only stabilised after 8 periods, due to criticality level inversion (i.e., black box in Figure 12). Conversely, With the support from hardware in MCS-IOV systems, criticality level inversion can be eliminated (i.e., the deadline miss rates can be stabilised immediately after system model switch).

VI. Conclusion

In safety/life-critical systems, MCSs are proliferating at a rapid pace in both academia and industry, mainly due to the various functionalities required by modern safety-critical systems and fast evolution of execution platforms. In MCS, the I/O system is vitally important, since the system modes of the whole system are often determined by the use of I/O. In a MCS I/O system, multiple features are often simultaneously required – isolation/separation, performance/efficiency, and predictability. Currently, existing approaches cannot achieve all these requirements at the same time.

MCS-IOV is the first systematical solution that fulfils all the three common requirements of MCS as well as supports adaptive I/O resource management. Moreover, MCS-IOV supports I/O-driven mode switch — e.g., the mode switch can be triggered by detection of unexpected I/O behaviors (e.g., a higher I/O utilization than expected) to help the system to better satisfy the timing requirement in the high-criticality mode.

VII. Acknowledgements

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818). Meantime, Zhe Jiang would like to present his deepest appreciation to his new family members: Mrs. Yanting Dai and Mr. Akira Jiang for the support of a very tough time. Especially, Mr. Akira Jiang, you will be always alive in Zhe Jiang's heart.

References

[1] ARM gnu tool chain website. https://developer.arm.com/open-source/gnu-toolchain/gnu-rm. Accessed September 23, 2018.
[2] ARM official website. https://www.arm.com/. Accessed September 27, 2017.
[3] Bluespec inc. bluespec system verilog (bsv). http://www.bluespec.com/products/. Accessed September 27, 2017.
[4] Encounter rtl compiler. https://www.cadence.com/. Accessed Otc 16, 2018.
[5] Freertos i/o official website. http://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_IO/FreeRTOS_Plus_IO.shtml. Accessed September 27, 2016.
[6] Freertos official website. http://www.freertos.org/. Accessed September 27, 2017.

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

[8] NXP: Functional Safety System Basis Chips (FSSBC). https://www.nxp.com/docs/en/brochure/SBCAUTOBRA4.pdf. Accessed Jan 12, 2019.

[9] NXP: NXP S32x Series. https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/. Accessed Jan 12, 2019.

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

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

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

[13] 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.

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

[15] Renesas: Rh850 micro-controller familiy. https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rh850.html. Accessed Jan 12, 2019.

[16] uCos official website. https://www.micrium.com/rtos/kernels/. Accessed September 27, 2017.

[17] Xilinx 1g/2.5g ethernet subsystem manual. https://www.xilinx.com/support/documentation/ip_documentation/tri_mode_ethernet_mac/v9_0/pg051-tri-mode-eth-mac.pdf. Accessed August 27, 2016.

[18] Xilinx microblaze gnu tool chain. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug1043-embedded-system-tools.pdf. Accessed September 23, 2018.

[19] Xilinx official website. https://www.Xilinx.com. Accessed July 5, 2017.

[20] Xilinx zc706 official website. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html. Accessed April 12, 2017.

[21] N. Audsley. Memory architectures for noc-based real-time mixed criticality systems. *Proc. WMC, RTSS*, pages 37–42, 2013.

[22] S. Baruah and A. Burns. Implementing mixed criticality systems in ada. In *International Conference on Reliable Software Technologies*, pages 174–188. Springer, 2011.

[23] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22. IEEE, 2010.

[24] 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.

[25] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and L. Van Doorn. The price of safety: Evaluating iommu performance. In *The Ottawa Linux Symposium*, pages 9–20, 2007.

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

[27] R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, 1993.

[28] 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.

[29] 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.

[30] A. Crespo, A. Alonso, M. Marcos, A. Juan, and P. Balbastre. Mixed criticality in control systems. *IFAC Proceedings Volumes*, 47(3):12261–12271, 2014.

[31] 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.

[32] J. Garside and N. C. Audsley. Prefetching across a shared memory tree within a network-on-chip architecture. In *ISSoC*, pages 1–4, 2013.

[33] M. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens. A globally arbitrated memory tree for mixed-time-criticality systems. *IEEE Transactions on Computers*, 2016.

[34] M. D. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens. A generic, scalable and globally arbitrated memory tree for shared dram access in real-time systems. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 193–198. EDA Consortium, 2015.

[35] 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.

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

[37] S. Groesbrink, L. Almeida, M. de Sousa, and S. M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24. IEEE, 2014.

[38] 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.

[39] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 13–23. IEEE, 2011.

[40] R. Hiremane. Intel virtualization technology for directed i/o (intel vt-d). *Technology@ Intel Magazine*, 4(10), 2007.

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

[42] F. Jahanian and A. K.-L. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on software engineering*, (9):890–904, 1986.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] J. A. Landis, T. V. Powderly, R. Subrahmanian, A. Puthiyaparambil, and J. R. Hunter Jr. Computer system para-virtualization using a hypervisor that is implemented in a partition of the host system, July 19 2011. US Patent 7,984,108.

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

[52] A. Ma and M. Zhang. Computer system architecture. *Computer*, 6:L1–1, 2001.

[53] E. Missimer, K. Missimer, and R. West. Mixed-criticality scheduling with i/o. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 120–130. IEEE, 2016.

[54] D. Muench, O. Isfort, K. Mueller, M. Paulitsch, and A. Herkersdorf. Hardware-based i/o virtualization for mixed criticality real-time systems using pcie sr-iov. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 706–713. IEEE, 2013.

[55] D. Münch, M. Paulitsch, O. Hanka, and A. Herkersdorf. Mpiov: scaling hardware-based i/o virtualization for mixed-criticality embedded real-time systems using non transparent bridges to (multi-core) multi-processor systems. In *Proceedings of the 2015 Design, Automation*

& Test in Europe Conference & Exhibition, pages 579–584. EDA Consortium, 2015.

[56] A. Oliveira, J. Martins, J. Cabral, A. Tavares, and S. Pinto. Tz-virtio: enabling standardized inter-partition communication in a trustzone-assisted hypervisor. In *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, pages 708–713. IEEE, 2018.

[57] M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. Mixed-criticality embedded systems–a balance ensuring partitioning and performance. In *2015 Euromicro Conference on Digital System Design*, pages 453–461. IEEE, 2015.

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

[59] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares. Towards a lightweight embedded virtualization architecture exploiting arm trustzone. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–4. IEEE, 2014.

[60] 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.

[61] G. Plumbridge, J. Whitham, and N. Audsley. Blueshell: a platform for rapid prototyping of multiprocessor nocs and accelerators. *ACM SIGARCH Computer Architecture News*, 41(5):107–117, 2014.

[62] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

[63] O. Rozenfeld, R. Sacks, Y. Rosenfeld, and H. Baum. Construction job safety analysis. *Safety science*, 48(4):491–498, 2010.

[64] 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.

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

[66] J. W. Sheaffer, D. P. Luebke, and K. Skadron. A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors. In *Graphics Hardware*, volume 2007, pages 55–64. Citeseer, 2007.

[67] D. H. Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. ASQ Quality press, 2003.

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

[69] 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.

[70] 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.

[71] C. Waldspurger and M. Rosenblum. I/o virtualization. *Communications of the ACM*, 55(1):66–73, 2012.