2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS)

IOMPU: Spatial Separation for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using Non-Transparent Bridges

Daniel Münch*, Michael Paulitsch†, Andreas Herkersdorf‡
*Airbus Group Innovations, Munich, Germany, email: Daniel.Muench@airbus.com
†Thales Group (Thales Austria GmbH), Vienna, Austria, email: Michael.Paulitsch@thalesgroup.com
‡Technische Universität München, Munich, Germany, email: herkersdorf@tum.de

***Abstract*—Safety-critical systems and in particular mixed-criticality systems require spatial and temporal separation for their hosted applications and functionalities. Additional constraints are using Commercial Off–The–Shelf (COTS) components, portability and determinism. These items are required for economic success for products with low piece numbers and long life-cycles like aircraft. Available embedded processors lack means for spatial separation of Input/Output (I/O) components like an Input/Output Memory Management Unit (IOMMU). The objective of this paper is to provide spatial separation for I/O in COTS mixed-criticality embedded real-time systems like avionics with minimum possible impact on performance (transfer time, transfer rate, Central Processing Unit (CPU) usage).**

The three main contributions of this paper are: (1) The presented Input/Output Memory Protection Unit (IOMPU) enables to upgrade spatial separation for I/O to a system by using COTS components and Non-Transparent Bridge (NTB) technology. In addition, the IOMPU concept is compatible with existing temporal separation solutions. (2) The paper shows a prototype implementation and a potential use case in context of hardware-based I/O virtualization. (3) The evaluation in this paper demonstrates that the IOMPU concept is practically applicable. The performance overhead (transfer time, transfer rate) is below 0.88%, which is almost negligible, particularly compared to state-of-the-art software-based solutions.

I. Introduction

Safety-critical systems, such as an aircraft, require certification otherwise no operation is allowed and no business is possible. Certification is the process of proving that the system and its subsystems work safely without unacceptable hazards. A manufacturer has to provide evidence for defined safety targets and design assurance activities that an independent authority assesses and accepts. The cost of the development of an avionic system depends on its criticality-level classification, which is determined by the severity of its failure. For example, a classification at the highest level (DAL-A, catastrophic failure severity [1] [2]) adds additional 65% to the development costs [3]. The demand for more and more functionality combined with the demand to save space, weight and power forces higher functional integration onto a shared platform [4]–[6]. To fulfill the targets of certification, robust partitioning encompassing temporal separation based on spatial separation is mandatory [1], [2], [7]. Spatial separation means that a functionality cannot change private configuration or data of another functionality [1], [2], [7]. Temporal separation means that a functionality gets a guaranteed (data) rate or latency and that its timing behavior cannot be affected by another functionality [1], [2], [7]. Separation is especially important if functionalities of different criticality levels are integrated on a shared platform resulting in so called mixed-criticality systems [8]–[10]. Although the research focus of mixed-criticality systems has been shifted to temporal separation (cf. [8], [11], [12]), it must be noted that temporal separation relies on a working spatial separation. Without separation, the integrated functionalities have to be certified at the highest level of all integrated functionalities [1], [2], [7]. This means that without spatial separation, the development for certification of a mixed-criticality system is not economically viable. In addition to the separation requirements, further constraints are using Commercial Off–The–Shelf (COTS) components, portability and maintainability [5], [6], [13]. These items are required for economic success for products with low piece numbers / volume and long life-cycles like aircraft. Other important constraints are a static system configuration, low complexity, determinism and predictability [5], [13]–[15]. The reason for these items are that they ease the effort for the certification process of a development project.

This paper focuses on spatial separation of Input/Output (I/O), since every system needs I/O and I/O is an often underestimated issue [4], [16]. In particular, Direct Memory Access (DMA) is a challenging issue. Temporal separation for I/O has already been addressed by [17]–[20]. Spatial separation for I/O seems to be solved by Input/Output Memory Management Units (IOMMUs) in IT-server systems. However, spatial separation is a particular issue in embedded systems [21]–[23], since embedded processors or embedded processing system-on-chips have no adequate means for spatial separation such as an adequate IOMMU. Existing work [22] [23] address this issue using software-based solutions, but these solutions have non-negligible performance implications. Since temporal separation is based on spatial separation, solving the lack of spatial separation of I/O in embedded processors is of significant importance.

The challenge and objective of this paper is to provide a spatial separation solution for I/O in mixed-criticality COTS embedded real-time systems with minimum impact on performance (transfer time, transfer rate, Central Processing Unit (CPU) usage) and considering the above mentioned additional constraints.

The three major contributions of this paper are: (1) The

978-1-4799-8937-9/15 $31.00 © 2015 IEEE
DOI 10.1109/HPCC-CSS-ICESS.2015.221

paper presents the Input/Output Memory Protection Unit (IOMPU) concept providing hardware-based spatial separation for I/O based on Non-Transparent Bridge (NTB) technology (cf. Section II-B). The IOMPU concept enables spatial separation in COTS systems without an IOMMU. It offers a platform independent, reusable and standardized solution. In addition, the IOMPU concept is compatible with existing temporal separation concepts [17], [19], [20] and adaptable to existing temporal separation concepts [18]. (2) The paper shows a prototype implementation and an use case of the IOMPU concept in context of hardware-based I/O virtualization in embedded systems (cf. Section II-A). (3) Finally, the evaluation of the paper shows that the spatial separation is practically applicable and has almost negligible performance overhead, particularly compared to a state-of-the-art software-based solution.

In the remainder of this paper, Section II overviews fundamental technologies, while Section III reviews related work. Section IV discusses the IOMPU concept, its prototype implementation and its relevance. Section V evaluates the enforcement of spatial separation, its performance overhead and its importance. The conclusions are drawn in Section VI.

II. Background

At first, this section overviews Hardware-based I/O Virtualization. Thereafter, NTB technology is introduced.

A. *Hardware-based I/O Virtualization*

The context of the presented IOMPU concept is hardware-based I/O virtualization [23], [20]. Hardware-based I/O virtualization is the hardware-supported sharing of I/O devices in virtualized or partitioned embedded systems. Virtualized or partitioned systems are systems where several virtual machines or multiple application partitions operate on a shared computing platform supervised by a virtual machine manager or hypervisor. Advantages of hardware-based I/O virtualization are I/O sharing reaching a near native performance and guaranteed separation. The idea is to use memory-mapped I/O like PCI Express (PCIe) and to offload the operation of the sharing to hardware. The hardware interface is divided into a Physical Function (PF) (management interface) and several Virtual Functions (VFs) interfaces (application interfaces) (cf. PCIe [24] Single Root I/O Virtualization (SR-IOV) [25]). The PF is directly mapped to the control partition or the hypervisor, whereas a VF is directly mapped to the corresponding application partition. Already available means for memory management like a Memory Management Unit (MMU) and an IOMMU provide the spatial separation. MMUs separate the transactions from CPU to I/O (Programmed I/O (PIO) transactions). The opposite direction from I/O to CPU (DMA transactions) is separated by IOMMUs (cf. Figure 1). However, nowadays available embedded processors lack an IOMMU or an IOMMU, which is capable of separating I/O devices or management interfaces from application interfaces or application interfaces from other application interfaces. Therefore, a safe and secure I/O sharing cannot be accomplished in such systems.

B. *Non-transparent bridge(ing) (NTB) technology*

In general, NTB technology or non-transparent bridging means connecting two divided elements non-transparently (cf. Figure 2). Connecting two elements non-transparently implies connecting in a way that is easily visible and obviously detectable for the system and its parts. This means that a connection is not usable by default and needs to be set up and configured before it is usable. The PCIe standard uses a tree topology with maximally one root, master or CPU. This tree is also called (single-root) PCIe hierarchy. The issue of such a topology is that communication between two roots, masters or CPUs is not possible per default. NTB technology in context of PCIe [26], [27] is the non-transparent solution for a multiprocessor system to allow multiple CPUs to communicate via PCIe. An NTB is a device consisting of two endpoint devices back-to-back with an additional address translation functionality in-between (cf. Figure 2). If an NTB connects two (single-root) PCIe hierarchies, each side of an NTB is part of one PCIe hierarchy. For each PCIe hierarchy, each side of an NTB looks like a normal endpoint. The address translation functionality of the NTB establishes an address window (aperture) from one PCIe hierarchy to the other PCIe hierarchy. This address window or aperture allows a CPU to communicate with another CPU through an NTB.

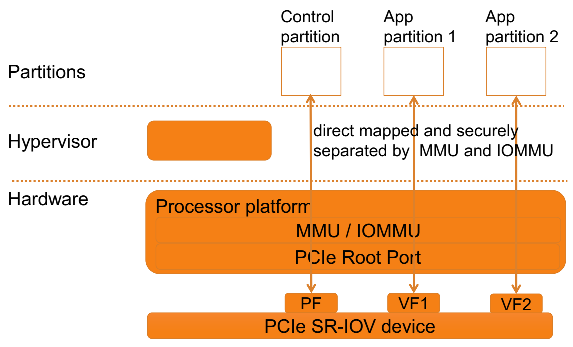

Fig. 1. Hardware-based I/O virtualization based on PCIe SR-IOV

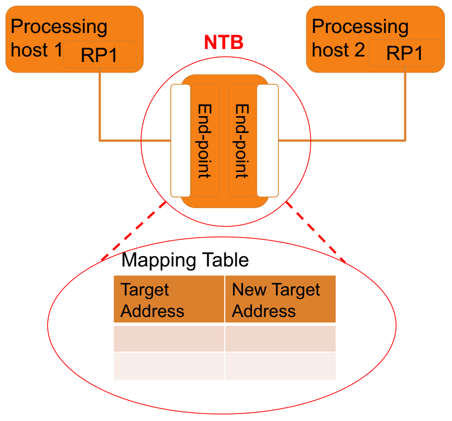

Fig. 2. Non-transparent bridge(ing) (NTB) technology

III. Related Work

Some research papers and patents have been published about IOMMUs in the recent years. [28] depicts an architecture of an IOMMU as part of a processor. A processor-internal (proprietary) interconnect connects the IOMMU with system memory and I/O devices via bridges. The architecture encompasses a multi-level hierarchical storage structure for translation data with multiple domain identifiers in the main system memory. [29] describes another architecture of an IOMMU located in a processor. The IOMMU is connected by a processor-internal interconnect (with a proprietary protocol). The focus is a method to enable a trusted initialization over a shared link for data and control. The method is based on a special protection mode blocking all I/O requests. In contrast, the current paper presents a protection unit at the I/O interconnect bus level, which uses the standardized PCIe protocol, a safe blocking feature by default, an encapsulated storage for the access rules within the protection unit and a dedicated connection for control.

[30] and [31] suggest a quota-based on-demand DMA or IOMMU mapping strategy enabling run-time flexibility and overprovisioning of system memory. Flexible configuration and dynamic memory allocation aim to improve IT-server systems and come with additional complexity and non-deterministic behavior. This contradicts the prerequisites of determinism, predictability and low complexity for mixed-criticality embedded real-time systems.

[23] describes a software-based solution mitigating the limitations of the IOMMU of the Freescale QorIQ Platform Series (e.g. P4080). This IOMMU is not capable of separating similar I/O devices (e.g. PCIe devices or PCIe functions). The general idea of the software-based Input/Output Memory Management Unit Handshake Protocol (IOMMUHP) concept is that a specific PCIe function of the I/O device must request and release DMA accesses via the hypervisor, which activates and deactivates the correct IOMMU settings. [22] presents another software-based approach trapping each DMA transaction in the hypervisor. This privileged software part manages and checks access rights and performs the DMA accesses on behalf of the application partition. The main problem is that a software-based approach comes with a non-negligible performance overhead for transfer rate, transfer time and CPU usage. In contrast, the current paper presents a hardware-based solution with negligible performance overhead. In addition, the IOMPU concept offers a more general and portable spatial separation solution, since it is applicable to multiple platform architectures.

[26], [27] discuss using NTB technology to enable inter-processor communication with the PCIe protocol (cf. Section II-B). In contrast, the IOMPU concept uses NTB technology (in a formerly unintended way) to provide spatial separation like an IOMMU. The IOMPU concept divides the usual single connection of an I/O device to the host into two connections. A (non-transparent) reconnection of these two connections with an NTB enforces one connection for management and control and the other connection for controlled data transactions. An additional control engine within an NTB decides whether or not to block transactions with an encapsulated rule set in dependency of the source / origin ID and target address of the transactions.

[32] uses PCIe interconnect, NTB and IOMMU to share a PCIe SR-IOV network card among multiple Intel Xeon hosts in the IT-server domain. In contrast, the current concept uses NTB technology to emulate an external IOMMU to provide spatial separation for I/O devices like the separation feature of an IOMMU for a single (multi-core) computing host lacking an IOMMU.

[33] uses PCIe interconnect and multiple NTBs to share a PCIe I/O device in a multi-processor system. An NTB checks the source and destination to decide whether to block or pass a data transaction from or to the PCIe I/O device. In contrast, the presented concept uses a single NTB with examining the origin and target address of the transaction in a formerly unintended way to upgrade the spatial separation feature of an IOMMU to a single-processor system. The multi-processor sharing idea is transferred and adapted from multi-processor systems to single-processor systems using two isolated PCIe roots in a single-processor system. Purpose is to enable spatial separation functionality for single processor systems or system-on-chips that do not have adequate means, which is the case for available embedded architectures.

[34] uses NTB and isolated interrupt trigger registers in the processor system to safeguard PCIe device interrupts and inter-processor interrupts in the PCIe interconnect. In contrast, the current concept focuses on protecting data transactions of PCIe devices in a single-processor system. Intention is to upgrade an IOMMU-like spatial separation functionality to embedded single-processor systems or system-on-chips that do not have adequate separation means.

IV. I/O Memory Protection Unit (IOMPU)

First, this section describes the central ideas of the IOMPU concept. Then, the prototype implementation and the impact of the IOMPU concept is discussed.

A. Concept

The key idea of the IOMPU concept is to use a *dedicated COTS component to perform the spatial separation*. We call this component an IOMPU. Since the typical computing system today is a system-on-chip, internal buses and networks-on-chip cannot be accessed directly. However, the spatial separation does not need to be performed shortly before the system memory, it can also be performed much earlier in the I/O interconnect. Therefore, the memory-mapped I/O interconnect like PCIe is disconnected and intercepted by the IOMPU. The presented IOMPU concept uses and extends the previously mentioned NTB technology by employing four steps.

The first step of the IOMPU concept is to divide the single link and the single address space between the I/O device and the host system into two links and two separated (bus) address spaces or parts (cf. Figure 3).

The second step of the IOMPU concept is to connect these two separated links and address spaces by an NTB again (cf. Figure 4). The effect of a bridge acting as an endpoint is a non-transparent behavior. That means that no transaction can pass the bridge by default without any further configuration or means. This forms one important part of separation, *the separation by default*. The reason for this is

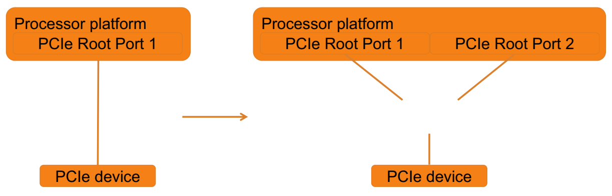

Fig. 3. Step 1: Dividing link / address space into two parts

that an PCIe endpoint terminates a transaction by default and is not able to route and forward a transaction further [24]. For IT-server systems this non-transparent behavior is considered cumbersome. However, for safety and security critical systems, this behavior is in particular valuable to ease a safe and secure system boot up. For example, state-of-the art IOMMUs do not have this blocking feature by default. [29] is also aware of the importance and suggests a special IOMMU protection mode and a special boot-up configuration sequence to achieve this behavior.

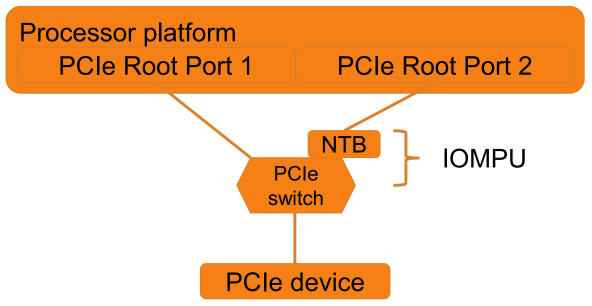

Fig. 4. Step 2: (Re)connecting by non-transparent bridges

The third step is to use one address space or part exclusively for management and control without admitting DMA memory transactions (cf. Figure 5). This establishes another part of spatial separation, *the separation between management or control and data*. Even if a partition has a safety or security issue, this partition cannot misuse the I/O device (or an application interface of an I/O device) in any way to change the rule set configuration, since management or control and application data are explicitly separated. The exclusive usage for control of one link is achieved by deactivating memory space and the bus mastering property of an (uplink) switch port. In addition, the ingoing direction window from the I/O interconnect to the internal system-on-chip interconnect is deactivated and the memory space and the bus mastering property in the root port is deactivated. This way, a PCIe device (or PCIe function or application interface) is not able to use this control link for its initiated data transactions.

The fourth step is to explicitly allow DMA write and DMA reads initiated by the I/O device(s) over the NTB to a defined target address space behind the NTB (cf. Figure 6). The NTB opens an address window from one address space (or single-root PCIe hierarchy) to the other address space.

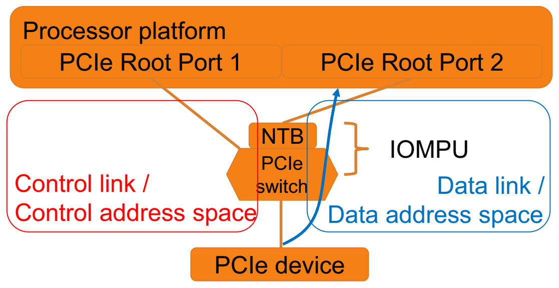

Fig. 5. Step 3: Exclusive management connection / address space

If a PCIe device (or PCIe function or application interface) initiates a transaction to the NTB, the control engine in the NTB checks the target address and source / origin ID (e.g. PCIe ID) according to a defined rule set (e.g. white list). This rule set decides whether to block the transaction or pass the transaction and translates the target address to the defined target address in the address space on the other side of the NTB. This source / origin ID check establishes another part of separation, *the separation by source / origin and target address*. The occurrence of such a blocking event can be monitored and reported back to the host system to take further actions. This rule set is located within the NTB. It is only accessible and configurable through the management and control link.

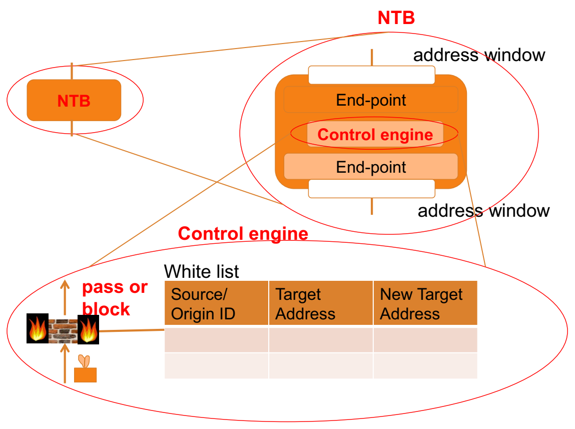

Fig. 6. Step 4: Data transactions over non-transparent bridge with source / origin ID check

Together these four steps form a fault containment zone or a sandbox for PCIe device (or PCIe function or application interface) since a PCIe device cannot break out its address limits and cannot change the rule set.

B. Implementation of the Concept

An application for the presented spatial separation IOMPU concept is a common mixed-criticality avionics computing platform, which is deployable from small unmanned aerial vehicles over helicopters to big aircraft. Each of these aircraft requires different and specialized I/O, but has the need for safely and securely shareable I/O interfaces with low performance overhead. Since certification requires separation but embedded processors do not provide sufficient means, this gap is filled by the IOMPU concept. We implemented the IOMPU concept in context of sharing a DMA-capable multi-function PCIe I/O card in a mixed-criticality multi-core embedded processing platform.

Figure 7 depicts the implemented prototype. A Xilinx VC709 FPGA evaluation board is used as PCIe-based I/O card. The PLX 8749 serves as PCIe switch and non-transparent bridge. A Freescale QorIQ P4080 Development System (P4080DS) is used as system host. The Freescale P4080 is an embedded multi-core processor and a representative of the Freescale QorIQ processor family. Freescale's Software Development Kit (SDK) Version 1.2 is used as software foundation. The avionic industry currently treats the PowerPC-based P4080 as a platform candidate for avionics systems [13], [16], [35]. This decision is mainly driven by certification reasons. The certification process requires the cooperation of the semiconductor vendor to provide detailed design documentation or detailed process information (e.g. about processor production). These points are at the moment given for the Freescale PowerPC architecture and not given for other architectures like ARM, AMD, Intel. In addition, the avionics industry has most (in-service) experience with the PowerPC architecture, which is very relevant to handle the effort for certification [36] [2]. This P4080 platform has an IOMMU which uses only a single PCIe ID for a whole PCIe tree and is therefore not able to distinguish multiple PCIe devices or PCIe devices with multiple functions [23].

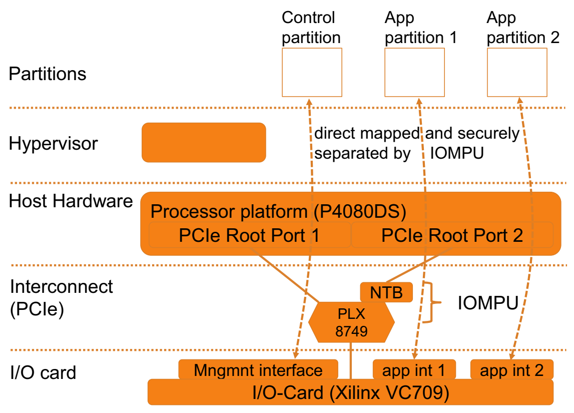

Fig. 7. Implemented prototype of the concept

To keep the complexity as low as possible, the prototype uses a DMA-capable and shared PCIe-based I/O card with two application interfaces. However, the IOMPU concept is scalable and can provide separation for multiple devices and multiple devices with multiple functions or virtual functions, like SR-IOV devices. Physical function (PF) 0 is used as management interface and application interface 1 and PF 1 is used as application interface 2. Another rationale for using two physical functions is that the PCIe SR-IOV capability of the Xilinx VC709 FPGA evaluation board is not compatible with the Freescale PowerPC-based QorIQ processor series. The Freescale processors do not support the optional PCIe Alternative Routing-ID Interpretation (ARI) extension. However, the Xilinx PCIe SR-IOV capability requires PCIe ARI to enable the addressing of VFs [20]. Xilinx has confirmed this issue. At the moment, we are discussing with Xilinx to remove this limitation in future Xilinx FPGAs.

The prototype comprises one control partition directly mapped to the management interface and two application partitions directly mapped to application interface 1 and application interface 2 of the I/O card, respectively. The PCIe connection over root port 2 is used for DMA write and DMA read transactions initiated by the two application interfaces of the I/O card. The origin of the DMA write and DMA read transactions are checked. Then it is decided if a transaction is blocked or allowed to pass by checking the white list.

C. *Discussion and Impact of the Concept*

The IOMPU concept is reusable and achieves the spatial separation by using standard-compatible components. The prototype of the IOMPU concept uses only available COTS components for system host and interconnect. The IOMPU concept and evaluation setup do not require to change the standardized interconnect protocol like PCIe. Since the standardized interconnect protocol is not modified, the solution is platform independent. The prototype can hence be reused in any computing platform supporting the standardized interconnect regardless of its architecture (Intel, ARM, PowerPC, etc.). The interoperability with PCIe is also the reason that the IOMPU concept is scalable to separate multiple I/O devices and multiple I/O devices with multiple PCIe functions or virtual functions like PCIe SR-IOV devices. The spatial separation granularity of the IOMPU concept is sufficient for PCIe functions. Each device or part of a device just requires a dedicated entry in the rule set in the NTB. The compatibility to PCIe is also the rationale that the IOMPU concept is interoperable with already existing temporal separation concepts described in [17], [19], [20]. Since [18] is based on the predecessor technology Peripheral Component Interconnect (PCI), it would be made compatible with the IOMPU concept with little effort. Portability, scalability and compatibility are important prerequisites to provide maintainability for products with a long life-cycle like aircraft.

Compared to an IOMMU, the IOMPU concept has additional advantages. Placing the IOMPU in a standardized I/O interconnect offers a more general and portable solution than an IOMMU. Since IOMMUs are usually located further away from the I/O device and closer to the system memory, the IOMPU concept located in the I/O interconnect provides an earlier spatial separation. Another point is the blocking feature by default that eases a safe and secure boot up. In addition to this, the presented approach uses a physically dedicated control link and keeps the controlling rule set within the protection unit instead of in the system memory. This encapsulated storage of the rule set is especially interesting for safety and security-relevant systems. Advantages are a higher system safety and security as well as a more predictable and deterministic behavior, since no data need to be fetched from main memory and can cause interference (cf. [37]).

The physically dedicated control link assumes that the computing system has at least two dedicated control links or address spaces. In context of PCIe, this means two root ports or two PCIe hierarchies. This is an accomplishable prerequisite, since available computing systems fulfill this, regardless of their platform architecture. Nowadays, most systems offer more than one root port or PCIe hierarchy (cf. [38]–[40]). Spending one root port is hence an acceptable price for achieving spatial separation.

The IOMPU concept has described monitoring and controlling transactions from the PCIe device(s) to the host system to achieve spatial separation. Moreover, the IOMPU concept can also be used in the opposite direction from the system host over the NTB to the I/O device(s) and enforce spatial separation in this direction. Furthermore, the ability to filter in both directions also allows to realize a filtering gateway. The IOMPU concept can provide even more spatial separation features than an IOMMU. However, this is not the focus here.

The IOMPU concept can be applied to any memory-mapped I/O standard, for example also to serial Rapid I/O (sRIO). In addition to safety certification the IOMPU concept can also be utilized for security certification.

V. Evaluation

At first, this section describes the setup to evaluate the enforcement of spatial separation and the setup to evaluate the performance overhead. Subsequently, the evaluation results of both are depicted and discussed.

A. *Setup for the enforcement of the source / origin ID check*

The enforcement of the source / origin ID check of the IOMPU concept is verified in the following way:
The control partition configures the NTB and clears and rearms the PCIe advanced error reporting feature (AER). A DMA write transaction that is compliant with the rule set of the NTB is transmitted. The control partition reads the content of the AER registers. These registers contain the complete header and the first 32 data bits of the first PCIe packet causing an error. Application partition 1 prints out the receiving DMA buffer. In this case, the print out of the receiving DMA buffer is expected to contain the previous sent data and the AER registers are expected to report no blocked packet. In the other case, a write transaction is triggered that is not covered by the allowed rule set of the NTB. In this case, the print out of the receiving DMA buffer is expected not to contain the previous sent data and the AER registers are expected to report the violating packet. To get confidence, this procedure is repeated 1 million times covering various addresses and rule sets.

B. *Setup for the performance overhead*

The performance overhead (transfer time, transfer rate, and CPU usage) of the IOMPU concept is measured by applying the following setup: The control partition configures the NTB. Two application partitions are targeted by continuous DMA read or write transactions. The hardware scheduler of the I/O card is set up to grant each of both application interfaces a 50% share of the available transfer rate of the PCIe link. The size of the DMA transaction is stepwise increased by increasing the number of consecutively sent packets. Each packet is 128 Bytes long. The number of packets is varied from 1 to 255. For each packet count the measurements are repeated 100 times.

The measurements of the transfer time, transfer rate and CPU-usage contain the low-level software overhead including the handling of synchronization interrupts. The measurement precision and time resolution is regulated by the *Timebase Register* of the e500mc processor core. In the current case the time resolution is determined to 20.41 ns. The transfer time is determined by sampling the timebase register for the starting time and end time. The transfer time is calculated by $\text{Ttime} = (\text{ValTimeBaseEnd} - \text{ValTimeBaseStart}) * \text{Tresolution}$. The transfer rate is calculated by $\text{Trate} = \text{Transfersize}/\text{Transfertime}$. The CPU usage is calculated by $\text{CPUusage} = \text{Tactive}/\text{Ttransfer} * 100\%$. Tactive is the time the CPU is occupied with handling I/O. Tactive is measured the same way than the transfer time using the timebase register.

The evaluation of the measurements of performance overhead is conducted with following procedure:
At first, the evaluation measures the absolute values for the transfer time, transfer rate and CPU-usage using an I/O device with an IOMPU (cf. Figure 7). Then the evaluation measures these absolute values again using an I/O device with an uncontrolled and unseparated connection without an IOMPU (cf. Figure 1). Thereafter, the relative difference of the transfer time for the IOMPU separation and no separation is calculated by $\text{diff\_ttime\_iompu\_}[\text{wr}|\text{rd}] = (\text{ttime\_iompu\_}[\text{wr}|\text{rd}] - \text{ttime\_no\_iompu\_}[\text{wr}|\text{rd}])\ /\ \text{ttime\_no\_iompu\_}[\text{wr}|\text{rd}] * 100\%$. The relative difference of the transfer rate is calculated correspondingly to the calculation of the relative difference of the transfer time. The relative difference of the CPU-usage is calculated by $\text{diff\_CPUusage\_iompu} = \text{CPUusage\_iompu} - \text{CPUusage\_no\_iompu}$. At second, the relative difference of the transfer time for the IOMMUHP separation and no separation (cf. Section III and [23]) is determined. This accomplished the same way than the relative difference values of the IOMPU concept. Finally, the relative difference values of the IOMPU concept and the IOMMUHP concept are compared.

C. *Evaluating the enforcement of the source / origin ID check*

Example output for the evaluation result of the enforcement of the source / origin ID check of the IOMPU concept is given by the following:

```
Test case 1: ID ok -> pass
//setup NTB
  (source ID=0 is allowed to target address=E0408000)
//trigger write 32bit to allowed combination of
  source ID and target address
  (packet with
  source ID=0,target address=E0408000,data=AFFEBEE0)
 PLXV_AER_HEADER0 480FDFD0: 00000000
 PLXV_AER_HEADER1 480FDFD4: 00000000
 PLXV_AER_HEADER2 480FDFD8: 00000000
 PLXV_AER_HEADER3 480FDFDC: 00000000
//printout targeted DMA buffer
 Print DMA buffer data: 0xAFFEBEE0

Test case 2: ID violation -> block
//setup NTB
  (source ID=1 is allowed to target address=E0408000)
//trigger write 32bit to not allowed combination of
```

```
   source ID and target address
   (packet with
    source ID=0,target address=E0408000,data=AFFEBEE0)
 PLXV_AER_HEADER0 480FDFD0: 40000001
 PLXV_AER_HEADER1 480FDFD4: 0000000F //ID=0
 PLXV_AER_HEADER2 480FDFD8: E0408000
 PLXV_AER_HEADER3 480FDFDC: E0BEFEAF
                             //-> AFFEBEE0
//printout targeted DMA buffer
 Print DMA buffer data: 0x0
```

Test case 1 of the source / origin ID check shows the event that a packet satisfies the rule set. The packet is allowed to pass. The receiving DMA buffer contains the sent data. Test case 2 of the source / origin ID check demonstrates the state that a packet violets the rule set. The receiving DMA buffer does not encompass the sent data. The packet is blocked and disclosed by the AER registers. Summarizing, the origin ID check of the IOMPU concept shows that the spatial separation based on the origin ID is enforced at PCIe function level. This means that the IOMPU concept is an applicable spatial separation solution for processor systems, which do not have an IOMMU at all or have an IOMMU, which is not able to distinguish multiple PCIe devices or PCIe devices with multiple functions.

D. Evaluating the performance overhead

Write transactions and read transactions for the cases of IOMPU separation and no separation show virtually the same absolute transfer time values and transfer rate values, independently of the considered application interface.

Since the values of application interface 1 and application interface 2 are also almost equal and do not differ, the evaluation of the relative difference values uses only the data of application interface 1. Figures 8 and 9 describe the relative difference in percent between the transfer time and transfer rate using the IOMPU concept and no separation, as well as the relative difference between using the software-based solution (IOMMUHP) and no separation (cf. Section III and cf. [23]).

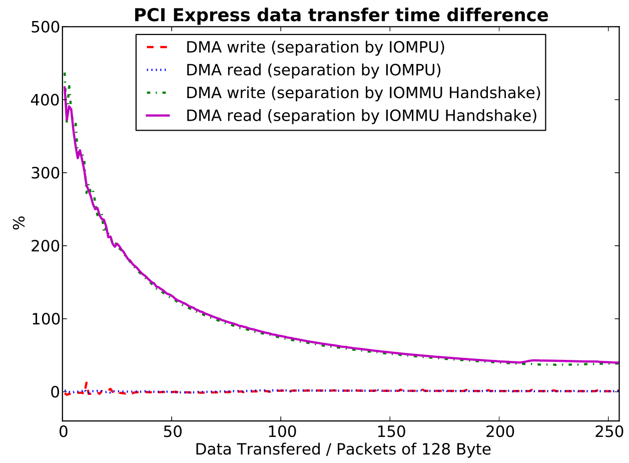

Fig. 8. Transfer time for read and write transactions: relative difference between using the IOMPU concept and no separation as well as relative difference between the software-based solution (IOMMU handshake protocol (IOMMUHP)) and no separation

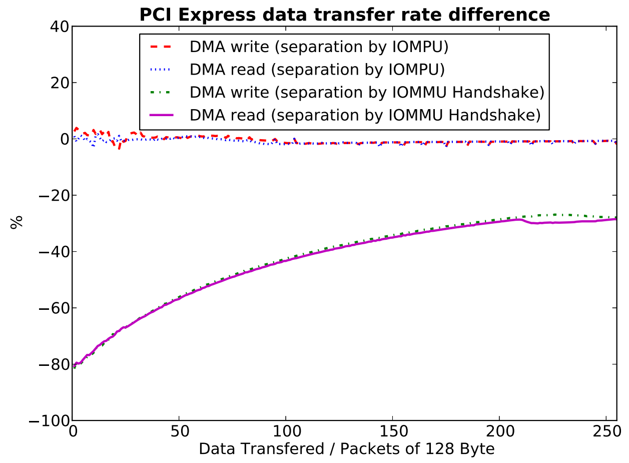

Fig. 9. Transfer rate for read and write transactions: relative difference between using the IOMPU concept and no separation as well as relative difference between the software-based solution (IOMMU handshake protocol (IOMMUHP)) and no separation

Figure 8 shows that the transfer time of the IOMPU concept is about 0.49% higher for write transactions and 0.86% higher for read transactions than the transfer time without separation. In case of the IOMMUHP, the transfer time is about 36.80% to 437.2% higher for write transactions or about 39.88% to 416.8% higher for read transactions than the transfer time without separation.

Figure 9 demonstrates that the transfer rate of the IOMPU concept is about 0.56% lower for write transactions and about 0.88% lower for read transactions lower than the transfer rate without separation. In case of the IOMMUHP, the transfer rate is about 26.87% to 81.57% lower for write transactions or about 28.50% to 80.63% lower for read transactions than the transfer rate without separation.

The transfer time figure (cf. Figure 8) and the transfer rate figure (cf. Figure 9) make clear that the IOMPU concept has a better transfer time value and transfer rate value than the software-based IOMMUHP concept under all circumstances. The IOMPU concept achieves up to 4-5 times lower and better transfer time values than the IOMMUHP concept. The transfer rate is by about the same factor higher and better. Furthermore, 0.86% higher transfer time and the 0.88% lower transfer rate of IOMPU concept compared to no separation is only a very low impact on performance, in most cases negligible. The slightly inferior transfer time and transfer rate for read transactions can be explained by the nature of the PCIe protocol. Write transactions are single packets without additional acknowledgment. In contrast, read transactions consist of a request packet answered by a completion packet containing the requested data. Therefore, the IOMPU concept influences read transactions more than write transactions since read transactions are influenced one more time.

The CPU-usage values in the cases of IOMPU separation and no separation do not differ. In case of the IOMMUHP, the CPU-usage is about 4.74% to 11.52% higher for write transaction or 4.73% to 10.45% higher for read transactions than the CPU-usage of transactions without separation.

From a CPU-usage and offloading perspective, the IOMPU concept has lower and therefore better CPU-usage than the IOMMUHP concept. The 0% CPU-usage difference of the IOMPU concept can be explained by the fact that the spatial separation can be totally performed in hardware. Hence, there is no significant CPU-usage difference between the case providing separation and the case providing no separation. Therefore, the IOMPU concept makes the CPU-usage impact of the software-based IOMMUHP concept obsolete.

VI. Summary and Conclusion

Regarding performance (transfer time, transfer rate and CPU usage), the IOMPU concept achieves nearly native performance without spatial separation. In contrast to software-based solutions like the software-based IOMMUHP, there is no significant performance degrading impact.

The advantage of the IOMPU concept is that it is a general, standardized, portable and platform-independent spatial separation solution using available COTS components. In addition to the aspect of wider applicability, the IOMPU concept provides an earlier spatial separation in the I/O interconnect, a safe blocking feature by default and a more encapsulated storage of the rule set compared to an IOMMU. Summarizing, the IOMPU concept is a practically applicable solution to upgrade spatial separation to mixed-criticality embedded real-time systems which do not have an IOMMU.

While this paper focuses on avionics, the results are applicable to adjacent markets which have similar stringent security and safety requirements, such as automotive, railway and industrial control.

Acknowledgment

This work was supported by the projects ARAMiS (01IS11035R) and SIBASE (01IS13020B). We thank Freescale, PLX Technologies and Xilinx for their support. Special thanks also to our colleagues from Airbus Defence and Space, especially Peter Ganal.

References

[1] EUROCAE, "ED-12/DO-178(C) Software considerations in airborne systems and equipment certification," 2012.

[2] ——, "ED-80/DO-254 Design Assurance Guidance for Airborne Electronic Hardware," 2000.

[3] Highrely, "DO-178B Costs Versus Benefits," Highrely, Tech. Rep., 2009.

[4] J. Littlefield-Lawwill *et al.*, "System Considerations for Robust Time and Space Partitioning in Integrated Modular Avionics," in *DASC*, 2008.

[5] L. Kinnan, "Use of Multicore Processors in Avionics Systems and its Potential Impact on Implementation and Certification," in *DASC*, 2009.

[6] X. Jean *et al.*, "Ensuring Robust Partitioning In Multicore Platforms For IMA Systems," in *DASC*, 2012.

[7] J. Rushby, "Partitioning in Avionics Architectures Requirements, Mechanisms, and Assurance," FAA, Tech. Rep., 2000.

[8] A. Burns *et al.*, "Mixed Criticality Systems - A Review," University of York, Tech. Rep., 2014.

[9] EUROCAE, "ED-124/DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations," 2007.

[10] J. Barhorst *et al.*, "A Research Agenda for Mixed-Criticality Systems," AFRL, Tech. Rep., 2009.

[11] S. Vestal, "Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance," in *RTSS*, 2007.

[12] S. Baruah *et al.*, "Towards the design of certifiable mixed-criticality systems," in *RTAS*, 2010.

[13] H. Agrou *et al.*, "A Design Approach for Predictable and Efficient Multi-Core Processor for Avionics," in *DASC*, 2011.

[14] D. Andrews *et al.*, "Impact of Embedded Systems Evolution on RTOS Use and Design," in *OSPERT*, 2005.

[15] D. Muench *et al.*, "Iterative FPGA Implementation Easing Safety Certification for Mixed-Criticality Embedded Real-Time Systems," in *DSD*, 2014.

[16] X. Jean *et al.*, "MULCORS - Use of Multicore Processors in airborne systems," EASA, Tech. Rep., 2012.

[17] S. Bak *et al.*, "Real-Time Control of I/O COTS Peripherals for Embedded Systems," in *RTSS*, 2009.

[18] R. Pellizzoni *et al.*, "Coscheduling of CPU and I/O Transactions in COTS-based Embedded Systems," *RTSS*, 2008.

[19] O. Kotaba *et al.*, "Multicore In Real-Time Systems Temporal Isolation Challenges Due To Shared Resources," in *WICERT*, 2013.

[20] D. Muench *et al.*, "Temporal Separation for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using PCIe SR-IOV," in *ARCS*, 2014.

[21] A. Hattendorf *et al.*, "Shared Memory Protection for Spatial Separation in Multicore Architectures," in *SIES*, 2012.

[22] O. Schwarz *et al.*, "Securing DMA through Virtualization," in *COMPENG*, 2012.

[23] D. Muench *et al.*, "Hardware-Based I/O Virtualization for Mixed Criticality Real-Time Systems Using PCIe SR-IOV," in *ICESS*, 2013.

[24] PCI-SIG, "PCIe Base Specification 3.0," 2010.

[25] ——, "Single Root I/O Virtualization and Sharing Specification 1.1," 2010.

[26] J. Regula, "Using Non-transparent Bridging in PCI Express Systems," PLX, Tech. Rep., 2004.

[27] ——, "Using PCIe in a variety of multiprocessor system configurations," 2007.

[28] M. Hummel *et al.*, "Direct Memory Access (DMA) Address Translation in an Input/Output Memory Management Unit (IOMMU) (US7809923)," 2010.

[29] A. Kegel *et al.*, "Input/Output Memory Management Unit with Protection Mode for Preventing Memory Access by I/O Devices (US8631212)," 2014.

[30] P. Willmann *et al.*, "Protection Strategies for Direct Access to Virtualized I/O Devices," in *ATC*, 2008.

[31] B.-A. Yassour *et al.*, "On the DMA Mapping Problem in Direct Device Assignment Categories and Subject Descriptors," in *SYSTOR*, 2010.

[32] C.-C. Tu *et al.*, "Secure I/O Device Sharing among Virtual Machines on Multiple Hosts," in *ISCA*, 2013.

[33] D. Muench *et al.*, "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 *DATE*, 2015.

[34] ——, "SgInt: Safeguarding Interrupts for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using Non Transparent Bridges," in *ARCS*, 2015.

[35] J. Nowotsch *et al.*, "Leveraging Multi-Core Computing Architectures in Avionics," in *EDCC*, 2012.

[36] EASA, "Certification Memorandum - Subject Development Assurance of Airborne Electronic Hardware," 2011.

[37] X. Zhou *et al.*, "The Interval Page Table: Virtual Memory Support in Real-Time and Memory-Constrained Embedded Systems," in *SBCCI*, 2007.

[38] Freescale, "P4080 QorIQ Integrated Multicore Communication Processor Family Reference Manual," 2011.

[39] Intel, "Intel Atom Processor E6x5C Series (Revision 001US)," 2010.

[40] TI, "Multicore ARM KeyStone II System-on-Chip (SoC) AM5K2E04, AM5K2E02 - SPRS864B," 2014.