MPIOV: Scaling Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using Non Transparent Bridges to (Multi-Core) Multi-Processor Systems

Daniel Münch, Michael Paulitsch, Oliver Hanka
Airbus Group Innovations
Munich, Germany
{Daniel.Muench | Michael.Paulitsch | Oliver.Hanka}
@airbus.com

Andreas Herkersdorf
TU München
Institute for Integrated Systems
Munich, Germany
herkersdorf@tum.de

***Abstract*—Safety-critical systems consolidating multiple functionalities of different criticality (so-called mixed-criticality systems) require separation between these functionalities to assure safety and security properties. Performance-hungry and safety-critical applications (like a radar processing system steering an autonomous flying aircraft) may demand an embedded high-performance computing cluster of more than one (multi-core) processor. This paper presents the *Multi-Processor I/O Virtualization (MPIOV) concept* to enable hardware-based Input/Output (I/O) virtualization or sharing with separation among multiple (multi-core) processors in (mixed-criticality) embedded real-time systems, which usually do not have means for separation like an Input/Output Memory Management Unit (IOMMU). The concept uses a Non-Transparent Bridge (NTB) to connect each processing host to the management host, while checking the target address and source / origin ID to decide whether or not to block a transaction. It is a standardized, portable and non-proprietary platform-independent spatial separation solution that does not require an IOMMU in the processor. Furthermore, the concept sketches an approach for PCI Express (PCIe)-based systems to enable sharing of up to 2048 (virtual) functions per endpoint, while still being compatible to the plain PCIe standard. A practical evaluation demonstrates that the impact to performance degradation (transfer time, transfer rate) is negligible (about 0.01%) compared to a system without separation.**

–Keywords: spatial separation, hardware-based I/O virtualization, non-transparent bridge (NTB), real-time embedded systems, mixed-criticality systems, IOMMU, IOMPU, multi-core, multi-processor

I. Introduction

Like in other fields of electronics, there is a demand in avionics to implement more and more functionality. This is accompanied by the trend of higher integration of functionality onto one (multi-core) computing platform to save space, weight and/or power consumption. The trend goes even further by integrating functionality of different criticality levels onto the same platform resulting in so called mixed-criticality systems.

The coexistence of functionalities of different criticality levels on one shared (multi-core) platform requires that these functionalities cannot interfere with other functionalities or with the entire system in case of a failure.
To handle this interference issue, temporal separation and spatial separation is essential to enable safety and security properties.
One central architectural system part is the Input/Output (I/O) subsystem, because almost every function needs I/O for its operation. We focus on I/O because it is an often underestimated problem.
Temporal separation means having an enforced separation in the time domain. For example, it is guaranteed that a device gets a certain transfer rate or limited transfer time over the interconnect. Spatial separation means having isolation in the address space domain. For example, it is enforced that an I/O device can only write into a distinct address range or memory of a distinct functionality or application. This prevents an I/O device from unwanted overwriting of application data associated with another I/O device. Such a situation could lead to a complete system failure.
Furthermore, the paper focuses on hardware-based I/O virtualization (cf. Section II and [1], [2]). This is the hardware-accelerated sharing of I/O subsystems in virtualized embedded systems. It uses memory-mapped I/O like PCI Express (PCIe) and offload the virtualization and sharing handling to hardware.
The vision of a common platform does not mean that such a platform is limited to one (multi-core) processor. Moreover, a performance hungry, but still critical application, may require for an embedded high-performance computing cluster with more than one (multi-core) processor, while still sharing the remaining processing power with other applications. An example is a collision avoidance system consisting of a radar processing system controlling or steering an autonomous flying aircraft. Such a system is highly critical and requires separation of shared interfaces from other applications.
Further constraints are the usage of Commercial Off–The–Shelf (COTS) components and the compatibility to standards. This is essential to keep costs low for products with low piece numbers / volume and long life cycles like aircraft.
[2] has already discussed temporal separation, whereas [3] has described spatial separation for a single processor system. The current paper focuses on scaling hardware-based I/O virtualization to (multi-core) multi-processor systems.

978-3-9815370-4-8/DATE15/©2015 EDAA

The challenge and objective of this paper is to provide an I/O sharing solution for an embedded high-performance cluster of (multi-core) multi-processors. Spatial separation should be granted with minimal impact on performance. This has to be applied to mixed-criticality embedded real-time systems, which typically do not provide adequate means for spatial separation like an Input/Output Memory Management Unit (IOMMU).

The contribution of the Multi-Processor I/O Virtualization (MPIOV) concept of this paper enables sharing of I/O devices (e.g. PCIe Single Root I/O Virtualization (SR-IOV)) in multi-processor systems providing spatial separation without requiring an IOMMU. It is a platform architecture independent, reusable and standardized I/O sharing solution, which is scalable from a single processor to a multi-core multiprocessor system. Apart from this, this paper sketches a solution to further improve scalability by circumventing the common limit in embedded architectures of having maximum 8 shareable PCIe functions per PCIe I/O device up to 2048.
To our best knowledge, we are the first trying to discuss an I/O sharing solution for (multi-core) multi-processors in mixed-criticality embedded real-time systems that do not provide adequate means for separation.

While this paper focuses on avionics, the results are equally applicable to adjacent industries that have similar stringent security and safety requirements like automotive, railway and industrial control.

II. Related Work

The paper is based on hardware-based I/O virtualization (cf. [1], [2] and [3]). This is the hardware-accelerated sharing of I/O subsystems in virtualized embedded systems. Virtualized or partitioned systems consist of several virtual machines or application partitions running on a physical computing platform managed by a virtual machine manager or Hypervisor (HV). The big advantage of the hardware-based I/O virtualization approach is an I/O sharing concept providing near native performance with low overhead while still granting separation. The idea is to use memory-mapped I/O like PCIe and offload the virtualization handling to hardware [4]. In a second step, the offloaded virtualization provides a Physical Function (PF) (management interface) and several Virtual Functions (VFs) interfaces (application interfaces). The PF is directly mapped to the control partition or the HV. The VFs are directly mapped to the corresponding application partition. As a third step, already available means for memory management like Memory Management Unit (MMU) and IOMMU ensure the spatial separation (cf. Figure 1). Transactions from Central Processing Unit (CPU) to I/O are also called Programmed I/O (PIO). An MMU separates these PIO transactions like in case of conventional memory. In contrast to this, transactions from I/O to CPU are also called Direct Memory Access (DMA). An IOMMU separates DMA transactions.

[5], [6] and [7] describe the Non-Transparent Bridge (NTB) technology in context of PCIe. Standard PCIe has a tree topology with only one root - also called (single-root) PCIe hierarchy. This means, there is only one master, root or CPU per PCIe hierarchy at a maximum. The intention of NTB is to present a solution for a multi-processor system having

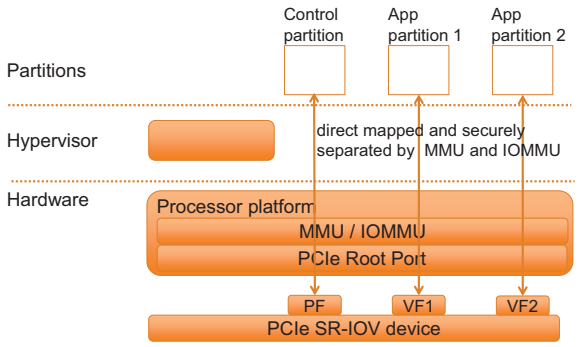

Fig. 1. Hardware-based I/O virtualization concept based on PCIe SR-IOV; Legend: PF= Physical Function (management interface); VF= Virtual Function (application interface); MMU= Memory Management Unit; IOMMU= I/O MMU; DMA= Direct Memory Access; PCIe= PCI express; SR-IOV= Single Root I/O Virtualization

multiple CPUs or roots to communicate or exchange data via memory-mapped I/O like PCIe. An NTB looks to a (single-root) PCIe hierarchy like an endpoint device (and not as a root). An NTB can be considered as two endpoints back to back with an address translation functionality. An NTB connects two separated PCIe hierarchies. Each of these two end-points is part of one PCIe hierarchy. NTBs open an address window (aperture) from one PCIe single root hierarchy to the other PCIe single root hierarchy. This way, a CPU or PCIe root is able to communicate with another CPU or PCIe root through an NTB. The current concept is based on NTB technology. However, it extends the NTB technology and uses it in a different way than formerly intended to enable multi-processor communication. The current concept uses NTBs to enable sharing of PCIe SR-IOV devices between multi-processors.

[3] depicts the Input/Output Memory Protection Unit (IOMPU) concept using 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. The current paper extends this approach to provide spatial separation for a (multi-core) multi-processor system, whose processors lack IOMMUs. This is still the case in embedded processors.

The Multi Root I/O Virtualization (MR-IOV) PCIe standard [8] suggests a concept for sharing PCIe devices in a multi-processor environment. MR-IOV uses far-ranging changes through the PCIe protocol layers. These changes require MR-IOV-aware endpoints and MR-IOV-aware switches, which are not available yet and are not likely to be available within the next years. In contrast to MR-IOV, the current concept is compatible with available PCIe and SR-IOV devices and PCIe switches.

Johnson et al. [9] depict system architectures sharing PCIe multi-function devices among multiple single PCIe hierarchies or processors using NTBs. [10] shortly sketches the idea to use NTB to interconnect multiple PCIe single-root hierarchies or processors sharing a PCIe SR-IOV device. Sharing PCIe SR-IOV devices in multi-processor systems is also described in [11]. However, the core area is to improve fault tolerance by

enabling VF migration with NTBs of SR-IOV devices between single PCIe hierarchies or processors. In contrast to the current paper, none of these sources targets sharing with safe and secure spatial separation of the devices or functions.

The closest related work are [12] and [13]. Both use PCIe interconnect, NTB and IOMMU to share a PCIe SR-IOV device among multiple processors considering spatial separation. In particular, [13] focuses on sharing a PCIe SR-IOV network card among multiple Intel Xeon hosts in the IT-server domain. In contrast to this, the current paper uses PCIe interconnect and NTB technology without an IOMMU to share a PCIe SR-IOV or PCIe multifunction device, while still providing spatial separation in a mixed-criticality real-time embedded system. The current concept presents a more general, platform-independent and portable solution, which can be applied in a variety of computing platforms (Intel, ARM, PowerPC, ...) and does not rely on special features of a distinct computing platform (Intel IOMMU / Intel VT-d).

III. Multi-Processor I/O Virtualization (MPIOV)

A. Concept

Basic constraints for the MPIOV concept are the usage of COTS components, portability and maintainability (using a uniform and standardized interface). This is needed for products with low piece numbers / volume and long life cycles like aircraft. Further assumptions are a static system configuration. The reason for this is the common understanding in industry that a safety-oriented and security-oriented development (like in avionics) priorities determinism, predictability and reduced complexity over dynamic flexibility [14]. This development approach of focusing on determinism is important, because it eases the effort of the required assurance or certification process of a development project [14]. Having an IOMMU is essential if DMA is used. One reason for using DMA is to achieve high transfer rates and low transfer times [1]. Another reason is that temporal separation requires DMA [1] [2]. As also stated in [1], hardware-based I/O virtualization is well-known in the IT server domain, but it is nearly unknown in the embedded mixed-criticality real-time domain. Embedded systems typically lack an IOMMU or have an IOMMU with limited functionality that is not capable of fulfilling the required separation of management interface and application interfaces [1].

The MPIOV concept enables the scaling of the hardware-based I/O virtualization for mixed-criticality embedded real-time systems using non-transparent bridges to (multi-core) multi-processor systems. Each processing host or CPU has its dedicated own (bus) address space or (single-root) PCIe hierarchy. The management host controls the main (bus) address space. Each of the dedicated (bus) address spaces of a processing host is connected to the main (bus) address spaces by an NTB (cf. Figure 2). One NTB is used per processing host. An NTB has the property that it blocks all transactions by default. An NTB does not look like a usual bridge to the two address spaces (cf. also Section II). Each side of the bridge looks rather as an endpoint (I/O device) to both address spaces. Without any further configuration or means no transaction can pass the bridge (cf. Figure 2). The main address space or part is exclusively used for management and control without admitting DMA memory transactions. The concept explicitly allows DMA write and DMA reads initiated by the I/O device(s) over the NTBs to a defined target address space behind an NTB (cf. Figure 2). An NTB opens an address window (aperture) from one (bus) address space to the other (bus) address space. If an application interface initiates a transaction to the NTB, 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 translate the target address to the defined target address in the (bus) address space on the other side of the NTB. This rule set is accessible and configured by the management host. An example setting is to allow sharing of the I/O device between processing host 1 and processing host 2 (cf. Figure 2). NTB1 allows access from application interface 1 or VF1 to a DMA buffer within the (bus) address space of processing host 1. All other target addresses in the bus address space of processing host 1 are blocked. NTB2 allows access from application interface 2 or VF2 to a DMA buffer within the (bus) address space of processing host 2. All other target addresses in the bus address space of processing host 2 are blocked. In the opposite direction, NTB1 allows access from processing host 1 only to the address space of application interface 1. Similarly, NTB2 allows access from processing host 2 only to the address space of application interface 2. In addition to this, the MMU of the processor (core) of processing host 1 further restricts the access window of NTB1 to its corresponding application. For processing host 2, the MMU of the processor (core) further restricts the access window in the NTB2 to its corresponding application.

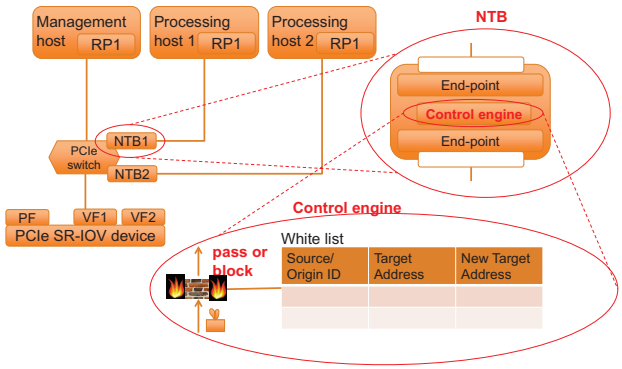

Fig. 2. Concept - I/O sharing between multiple processors

A further important benefit of the presented approach is that it circumvents the limit to address maximum 8 functions per end-point in standard PCIe systems. The standard PCIe (routing) ID consists of 16 Bits, 8 Bits for bus number, 5 Bits for device number and 3 Bits for function number. The device number is always ”zero” in PCIe and only present for Peripheral Component Interconnect (PCI) software legacy compliance. The PCIe standard [15] suggests for settings needing more than 8 functions using the optional Alternative Routing-ID Interpretation (ARI) extension. ARI reuses the in PCIe ”unused” device number and uses 8 Bit instead of 3 Bit to address up to 256 functions. Since this reinterpretation is a protocol change, it is required that all PCIe interconnect

participants (processor or root complex, switch, end-point) support this optional ARI extension. In addition to this, if more than 256 functions are needed in an end-point, [4] suggests using additionally bus numbers. In contrast to Intel server systems, embedded real-time systems do not support the ARI extension. A solution for avoiding this limitation and still complying with standard PCIe is to use multiple bus numbers instead of device numbers for an end-point device to address more than 8 functions. The configuration during boot-up (also called enumeration) adjusts in the PCIe bridge (root port or switch port) in upstream direction of the device the quantity of bus numbers, which the end-point needs in downstream direction. The end-point can then make use of the assigned bus numbers. For example, functions 0-7 have the PCIe ID 16h'0000 - 16h'0007. Functions 8-15 have the PCIE ID 16h'0100 - 16h'0107. This enables maximum $256 * 8 = 2048$ functions in a single end-point.

B. Evaluation Setup of the Concept

The evaluation setup examines the presented MPIOV concept in the context of sharing a DMA-capable multi-function PCIe I/O card in a mixed-criticality embedded processing platform with multiple multi-core processors.
Figure 3 depicts the system setup of the evaluation. A Xilinx

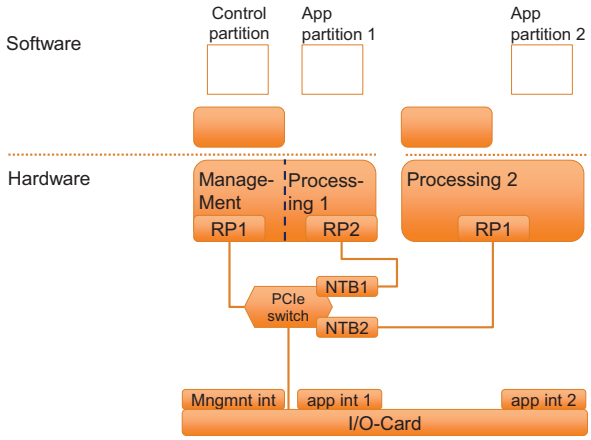

Fig. 3. Evaluation Setup for the Concept

VC709 FPGA evaluation board in PCIe form factor serves as I/O card. A PLX 8749 chip is used as PCIe switch and provides also the two non-transparent bridges. Two Freescale QorIQ P4080 Development Systems (P4080DS) serve as two system hosts. The P4080 platform is a representative of the Freescale QorIQ series - a PowerPC multi-core embedded processing platform. We use Freescale's Software Development Kit (SDK) Version 1.2 as software foundation. The PowerPC architecture-based P4080 platform is currently considered as a platform candidate for embedded avionics systems [16], [1]–[3], [14].

For the ease of explanation, the evaluation setup considers two multi-core processors and one DMA-capable and bus-mastering capable PCIe card with two physical PCIe functions. PF0 serves as management interface and application interface 1 and PF1 servers as application interface 2. A further reason for using two physical functions instead of the SR-IOV capability is that the SR-IOV capability of the Xilinx VC709 FPGA evaluation board is not fully compatible to the P4080DS. Xilinx requires the optional PCIe ARI extension to address VFs which is not supported by the P4080DS [2]. Xilinx has confirmed this issue and we are in discussion with Xilinx to remove this artificial limitation in the next generation of Xilinx FPGAs.

As mentioned above, the system encompasses two multi-core processors. The third management processor is spared to keep the costs and the effort for the evaluation system as low as possible. One core and one dedicated PCIe hierarchy or root port of the first multi-core processor takes over the tasks of the management host, so that only two processors are needed instead of three (cf. Figure 2). This management control partition is in charge of setting up the system, controlling the NTBs and the management interface of the I/O card. Application partition 1 running on the first multi-core processor is directly mapped to application interface 1 of the I/O card, whereas application partition 2 running on the second multi-core processor is mapped to application interface 2 of the I/O card. The spatial separation of the two application interfaces are performed by the NTB checking the target address and source / origin ID without an IOMMU.

The evaluation of the enforcement of the source / origin ID check in NTBs can be similarly conducted to the analysis already performed in [3]. For further details confer [3].

The evaluation of the performance overhead (transfer time, transfer rate) of the MPIOV concept is measured with the following procedure:
The control partition sets up the NTB. The two application partitions are target of repetitive read or write transactions. The performance of the MPIOV concept is evaluated by measuring the transfer time and transfer rate of transactions including the low-level software overhead like synchronization interrupts.
The *Timebase Register* of the e500mc processor core determines the time resolution and measurement precision. In this case the time resolution is configured to 20.41 ns.
The hardware scheduler in the I/O card is configured to provide 50% of the transfer rate for application interface 1 and 50% of the transfer rate for application interface 2. The size of the DMA transaction is stepwise increased by changing the number of 128 Byte-sized DMA packets sent consecutively by the device between arbitrated transactions. The quantity of the consecutively sent packets range from 1 to 255. The measurements are repeated 100 times for each packet count.
The described measurement procedure is conducted twice. One time it is performed using the presented MPIOV concept with separation. The other time it is executed using the state-of-the-art NTB configuration without separation (cf. [5] and [13]). Then both results are compared.

C. Evaluation Results of the Concept

Figure 4 shows transfer time results and Figure 5 depicts transfer rate results using the MPIOV concept. For the cases of the MPIOV concept with and without separation, the write transactions and the read transactions – regardless of the application interface – show virtually the same transfer time values and transfer rate values.

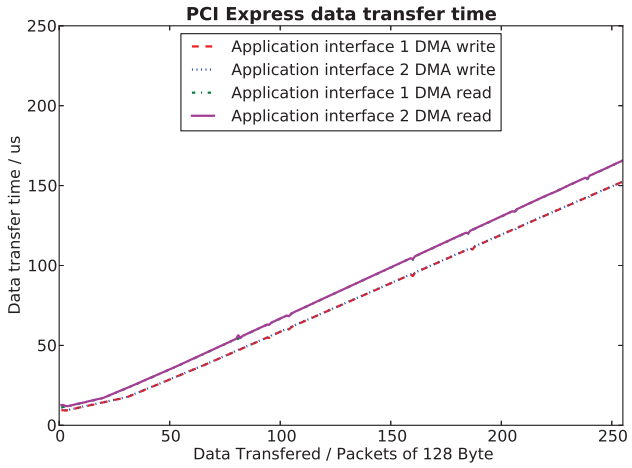

Fig. 4. Transfer time results using the MPIOV concept

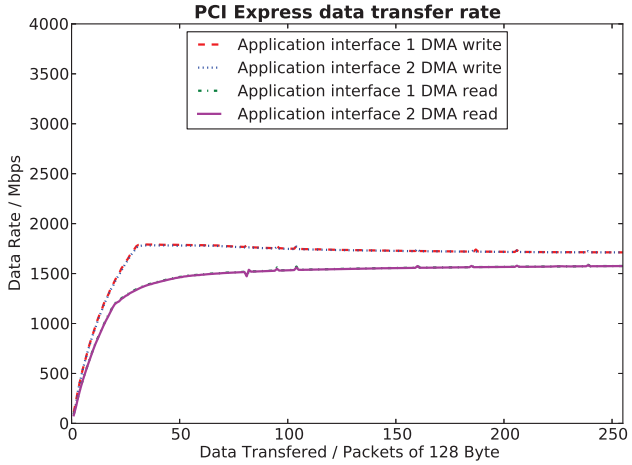

Fig. 5. Transfer rate results using the MPIOV concept

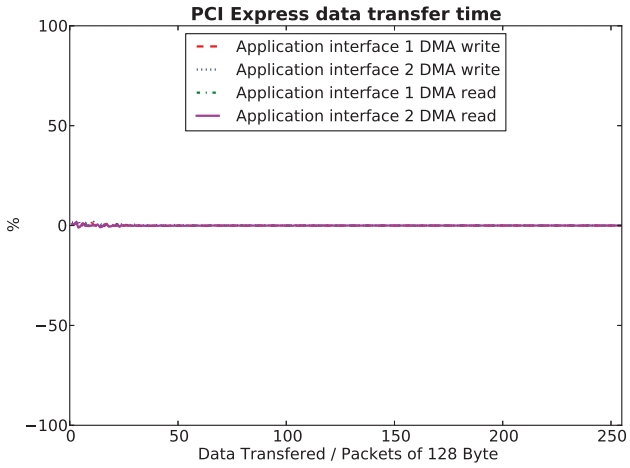

Fig. 6. Difference between the transfer time results of the case using separation with the MPIOV concept and the case without separation

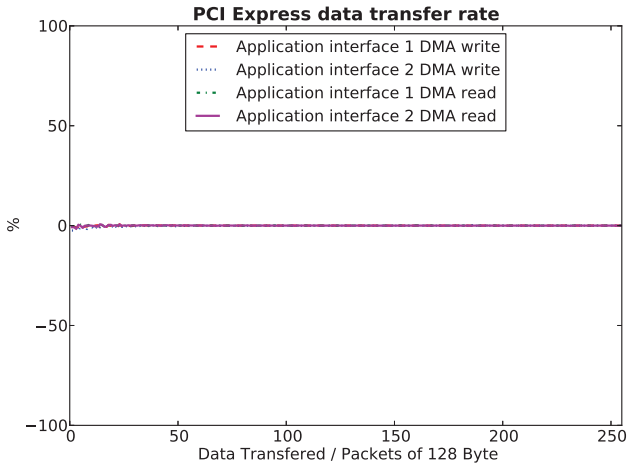

Fig. 7. Difference between the transfer rate results of the case using separation with the MPIOV concept and the case without separation

Since the values of application interface 1 and application interface 2 are almost equal, only the data of application interface 1 is used for further comparisons and evaluations. The Figures 6 and 7 describe the relative difference in percent between the transfer time and transfer rate of the MPIOV concept with and without separation.

Figure 6 shows the relative difference of the transfer time for the MPIOV concept with and without separation. The relative difference of the transfer time for the interrupt concept is calculated by:

$$\text{diff\_ttime\_}[\text{wr|rd}] = \frac{(\text{ttime\_sep\_}[\text{wr|rd}] - \text{ttime\_no\_sep\_}[\text{wr|rd}])}{\text{ttime\_no\_sep\_}[\text{wr|rd}] * 100}$$

The transfer time of write and read transactions of the MPIOV concept with separation differs 0.01% from the transfer time of transactions without separation.

Figure 7 shows the relative difference of the transfer rate for the MPIOV concept with and without separation. The relative difference of the transfer rate is calculated correspondingly to the calculation of the relative difference of the transfer time. The transfer rate of write and read transactions of the MPIOV concept with separation is about 0.01% lower than the transfer rate of transactions without separation.

D. Discussion and Impact of the Concept

The read and write data rate and transfer time figure (cf. Figure 6 and Section III-C) show that the MPIOV concept with separation differs in transfer time by 0.01% from state-of-the-art NTB configuration without separation. This is a very low impact to the transfer time, which can be neglected.
The figure for the transfer rate (cf. Figure 7 and Section III-C) confirms the statements of the transfer time.
For the ease of explanation, the evaluation setup considers two multi-core processors sharing one I/O card with two application interfaces. The MPIOV concept is scalable from one application interface per processing host to multiple applications interfaces per processing host with one NTB with multiple windows or multiple NTBs.
The MPIOV concept is reusable and a standardized solution providing spatial separation. The evaluation setup of the MPIOV concept uses only available COTS components for

system host and interconnect. The concept and evaluation setup do not require changing the standardized interconnect protocol like PCIe. Since the standardized interconnect protocol is not adapted, the solution is platform independent. The evaluation setup can be ported and reused in any computing platform supporting the standardized interconnect regardless its architecture (Intel, ARM, PowerPC, ...). Since the MPIOV concept itself provides spatial separation, there is no need that the applied processor systems have IOMMUs.

For embedded real-time systems, the Multi-BusNr-Device concept provides more than 8 PCIe functions up to maximum 2048 functions per PCIe end-point. In contrast to aggregating multiple end-points with 8 functions each, this approach offers not only also compatibility to the PCIe standard, but has the advantage of hardware resource usage of a single end-point. Being compatible with standard PCIe without the ARI extension has the advantage that such a multi-function device is usable in any system using PCIe (like ARM, Intel embedded, PowerPC, ...). It will not be restricted only to Intel server systems supporting the optional PCIe ARI extension. We are in dialog with Xilinx to implement this Multi-BusNr-Device concept in the Xilinx PCIe SR-IOV IP-core of future Xilinx FPGAs.

IV. Summary and Conclusion

The presented MPIOV concept scales the hardware-based I/O virtualization for mixed-criticality embedded real-time systems using non-transparent bridges to embedded high-performance computing cluster of (multi-core) multi-processor systems.

The MPIOV concept uses an NTB to connect each dedicated bus address spaces of a processing host to the main bus address space of the management host. The NTB checks the target address and source / origin ID (e.g. PCIe ID) according to a defined rule set and decides whether to block the transaction or pass the transaction. The evaluation results of the MPIOV concept shows that the performance overhead (transfer time, transfer rate) compared to no separation is negligible (about 0.01%). The MPIOV concept is reusable, standardized and portable solution providing I/O sharing among multiple processors without significant performance overhead. Since the MPIOV concept itself provides spatial separation, there is no need that the applied processor systems have IOMMUs.

The Multi-BusNr-Device concept uses multiple bus numbers instead of device numbers for an end-point device to allow to address more than 8 functions per end-point in a standard PCIe compatible system without the optional ARI extension. The Multi-Bus-Nr device concept mitigates the 8 functions per end-point limit without any additional resource overhead and is compatible with the PCIe standard without any proprietary extensions. At the moment, we are in discussion with Xilinx to implement this Multi-BusNr-Device concept in the Xilinx PCIe SR-IOV IP-core of future Xilinx FPGAs.

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 project ARAMiS and SIBASE funded by the German Federal Ministry of Education and Research (BMBF) under the funding ID 01IS11035R and 01IS13020B. We also thank Xilinx and PLX Technologies for their support. Special thanks also to our colleagues from Airbus Defence and Space, especially Peter Ganal.

References

[1] 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 *International Conference on Embedded Software and Systems (ICESS)*, 2013.

[2] D. Muench, M. Paulitsch, and A. Herkersdorf, "Temporal Separation for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using PCIe SR-IOV," in *International Conference on Architecture of Computing Systems (ARCS)*, 2014.

[3] D. Muench, "IOMPU: Spatial Separation for Hardware-Based I/O Virtualization for Mixed-Criticality Embedded Real-Time Systems Using Non Transparent Bridges (TR-TX4-399)," Airbus Group, Tech. Rep., 2014.

[4] PCI-SIG, "Single Root I/O Virtualization and Sharing Specification 1.1," 2010.

[5] J. Regula, "Using Non-transparent Bridging in PCI Express Systems," PLX, Tech. Rep., 2004. [Online]. Available: http://www.plxtech.com/files/pdf/technical/expresslane/NontransparentBridging.pdf

[6] ——, "Using PCIe in a variety of multiprocessor system configurations," 2007. [Online]. Available: http://www.eetimes.com/design/embedded/4006788/Using-PCIe-in-a-variety-of-multiprocessor-system-configurations

[7] A. Kazmi, "PCI Express and Non Transparent Briding Support High Availability," *Embedded Comping Design*, 2004. [Online]. Available: http://embedded-computing.com/pdfs/PLXTech.Win04.pdf

[8] PCI-SIG, "Multi-Root I/O Virtualization and Sharing Specification 1.0," 2008.

[9] B. D. Johnson and O. Torudbakken, "Routing direct memory access requests using doorbell addresses," 2009. [Online]. Available: http://www.freepatentsonline.com/7574536.htm

[10] A. Aswadhati, "Scaling Data Center Interconnects with PCI Express," 2011. [Online]. Available: http://www.pcisig.com/developers/main/training_materials/get_document?doc_id=415a477bf2725a554d7903f9d8d499daa3e8e4bb

[11] K. Tanaka and M. Takada, "Storage apparatus and virtual port migration method for storage apparatus," 2012. [Online]. Available: http://www.freepatentsonline.com/y2012/0096192.html

[12] D. Muench and M. Paulitsch, "Rechnersystem (Patent pending)," 2012.

[13] C.-C. Tu, C.-T. Lee, and T.-C. Chiueh, "Secure I/O Device Sharing among Virtual Machines on Multiple Hosts," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[14] D. Muench, M. Paulitsch, M. Honold, W. Schlecker, and A. Herkersdorf, "Iterative FPGA Implementation Easing Safety Certification for Mixed-Criticality Embedded Real-Time Systems," in *Euromicro Conference on Digital System Design (DSD)*, 2014.

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

[16] X. Jean, M. Gatti, G. Berthon, and M. Fumey, "MULCORS - Use of Multicore Processors in airborne systems," EASA, Tech. Rep., 2012.