Proc. of the 57th ACM/IEEE Design Automation Conference (DAC 2020), San Francisco, CA, USA, July 19-23, 2020.

AXI HyperConnect: A Predictable, Hypervisor-level Interconnect for Hardware Accelerators in FPGA SoC

Francesco Restuccia*†, Alessandro Biondi *†, Mauro Marinoni *†, Giorgiomaria Cicero *, and Giorgio Buttazzo*†
*TeCIP Insitute, Scuola Superiore Sant'Anna, Pisa, Italy
†Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy

***Abstract*—FPGA-based system-on-chips (SoC) are powerful computing platforms to implement mixed-criticality systems that require both multiprocessing and hardware acceleration. Virtualization via hypervisor technologies is, de-facto, an effective technique to allow the co-existence of multiple execution domains with different criticality levels in isolation upon the same platform. Implementing such technologies on FPGA-based SoC poses new challenges: one of such is the isolation of hardware accelerators deployed on the FPGA fabric that belong to different domains but share common resources such as a memory bus. This paper proposes AXI HyperConnect, a hypervisor-level hardware component that allows interconnecting hardware accelerators to the same bus while ensuring isolation and predictability features. AXI HyperConnect has been implemented on modern FPGA-SoC by Xilinx and tested with real-world accelerators, including one for Deep Neural Network inference.**

I. Introduction

Next-generation cyber-physical systems (CPS) such as autonomous vehicles and advanced robots are characterized by very complex computing workloads belonging to multiple subsystems with different criticality levels. Furthermore, they require hardware acceleration to accomplish heavy computations in real-time, such as those demanded by machine learning algorithms. To match these needs, heterogeneous platforms that integrate both multiprocessors and components that can implement hardware acceleration are emerging. Among such platforms, system-on-chips (SoC) that couple multiprocessors with field-programmable gate arrays (FPGA) are particularly promising as they can provide great flexibility in building CPS. In particular, they allow deploying reconfigurable energy-efficient, yet high-performance hardware accelerators (HAs) on the FPGA fabric while retaining all the advantages of classical software-programmable multiprocessors.

Many CPS are safety-critical and must undertake a *certification* process, at least for their critical subsystems. A key requirement for certifying safety-critical systems is *timing predictability*: critical functionality must complete within predetermined deadlines, and it is necessary to cope with *worst-case* behaviors of the system at design time to ensure so. This is particularly challenging for heterogeneous platforms, especially when considering that HAs can experience bus/memory contention that can severely affect their timing performance [1] — as a matter of fact, many HAs are memory-intensive (e.g., think of accelerators for machine learning in real-time video processing). Furthermore, *isolation* capabilities are required to avoid propagating faults across subsystems and to bound contention delays generated by low-criticality untrusted subsystems. Virtualization via hypervisor technologies is an established industrial practice for the co-existence of subsystems with mixed-criticality on the same platform while enforcing isolation [2]. However, new challenges arise when virtualization is applied to FPGA SoC. Indeed, isolation must not only be ensured for software components running on the processors (as done by most hypervisors) but also for HAs belonging to different subsystems that are jointly deployed on the same FPGA fabric. Just to name a possible issue, consider a HA of a low-criticality subsystem sharing a memory bus with another HA of a high-criticality subsystem. If no proper supervision is employed, the former HA is free to delay the latter in an unpredictable manner via bus contention [3]. Furthermore, bus arbitration and the corresponding latency must be known and predictable.

This paper. In this work, a new hypervisor-level hardware component named AXI HyperConnect is proposed. It allows interconnecting hardware accelerators to the same bus while ensuring isolation and predictability. The AXI HyperConnect has been developed in HDL language, allowing to achieve high performance in latency, while keeping comparable throughput and resource consumption with respect to state-of-the-art AXI interconnects for FPGA SoC. The AXI HyperConnect has been exported following the IP-XACT standard, which makes it compatible with other several commercial platforms, such as Intel FPGA [4]. Furthermore, it has been implemented on two modern FPGA SoC by Xilinx belonging to the Zynq-7000 and Zynq-Ultrascale+ families, and has been tested with real-world accelerators, including the one provided with the CHaiDNN framework by Xilinx to handle real-time inference of deep neural networks. The AXI HyperConnect has been integrated within a type-1 hypervisor under development in our laboratory.

II. Background

A typical FPGA SoC platform combines a processing system (PS), which includes one or more processors, with a Field Programmable Gate Array (FPGA) subsystem. Both the subsystems access a DRAM controller to reach a shared DRAM memory as illustrated in Figure 1. The de-facto standard interface for interconnections is the ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI) — also referred to as just AXI.

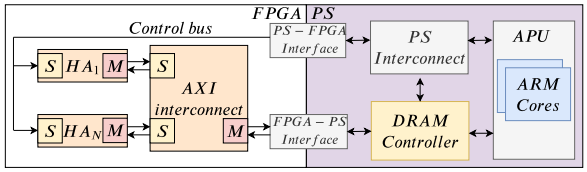

Fig. 1: Illustration of a typical FPGA SoC architecture and a sample multi-master architecture in the FPGA fabric.

The AXI bus. The AXI standard defines a master-slave interface to allow for simultaneous, bi-directional data exchange. All the transactions are initiated by a master, which requests to read/write data from/to a slave interface through AR or AW channels, respectively. Following address requests, data are transmitted back to the master on the R channel (data read) or provided to the W channel (data write) in the same order as requests have been routed to the corresponding address channel. Hence, data channels depend on address channels, and the access to the output data channels R and W depends on the routing order to the address channels. Even though the AXI standard

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

does not define this feature, many commercial devices [5], [6] follow this implementation choice, as reported in their documentation. The B channel is used by the destination device of a write request to acknowledge the master that its request has been correctly served.

FPGA-PS and PS-FPGA interfaces. The communication between the FPGA and the PS is allowed by two different types of interfaces: the PS-FPGA interface and the FPGA-PS interface (Figure 1). The first one offers a slave interface to the FPGA and is used by the processors to control the hardware devices or access data in the FPGA. In a dual manner, the second one offers a slave interface to the PS and is used by devices deployed on the FPGA to access the central DRAM memory or the on-chip memory in the PS.

Hardware accelerators. A hardware accelerator (HA) is a component developed in programmable logic and deployed on the FPGA fabric to perform a specific task. Each HA is associated to a software tasks (SW-task) running in the PS that controls it by issuing a request for acceleration. In typical FPGA SoC designsbased on shared-memory communication, HAs communicate with the PS via two AXI interfaces: an AXI master interface and an AXI slave interface. When a request for acceleration has to be issued, SW-tasks use AXI slave interfaces to setup the configuration of HAs (acting on memory-mapped registers). When running, HAs leverage AXI master ports to read/write data from/to the central DRAM memory in the PS. Following this paradigm, once activated, the execution of HAs is asynchronous with respect to the execution of SW-tasks running on the PS. HAs signal their completion to the PS by means of interrupts.

Multi-Master architecture. Whenever multiple AXI masters want to access the same output port, an AXI *interconnect* is in charge of arbitrating conflicting requests. This work is focused on AXI interconnects that have a set of slave input ports, to which HAs can be connected to, and one output master port, connected to a slave port in the FPGA-PS Interface. In FPGA SoCs by Xilinx, the state-of-the art AXI interconnect is called *AXI SmartConnect*. Even though the AXI standard defines specific signals to implement mechanisms for controlling the quality-of-service (QoS) of transactions, the AXI SmartConnect ignores such signals and implements a round-robin arbitration to solve the conflicts among the hardware accelerators in the access of the output master port (see [7], p.6 and p.8). A sample multi-master architecture is illustrated in Figure 1.

III. Related Work

Virtualization is becoming a crucial enabling technology in a growing number of application scenarios, including those related to safety-critical systems. Several open-source and commercial solutions support FPGA SoC but are mostly concerned with virtualization and isolation for software systems running on the processors in the PS. Virtuosity Xen Zynq Distribution (XZD)is a version of the popular Xen hypervisor customized for the Zynq SoC family by Xilinx. Its support for FPGA virtualization and isolation is limited to the management of address translations to handle data communication between the FPGA and software domains on the PS. Other hypervisor-based solutions have been proposed to virtualize the FPGA area by exploiting dynamic partial reconfiguration [8], [9], but still not supervising the data traffic generated on shared buses deployed on the FPGA fabric. To the best of our records, and according to public information, no hypervisor-level support is available to control AXI transactions on the FPGA. Concerning the AXI bus, a few hardware components have been recently proposed to increase isolation and predictability of AXI transactions while relying on state-of-the-art interconnects such as the AXI SmartConnect by Xilinx. Pagani et al. [10] proposed a solution to reserve a given bus bandwidth to each AXI master, while Restuccia et al. [11] presented a solution to guarantee a fair bandwidth distribution among HAs that issue transactions with heterogeneous burst sizes. However, both solutions require a dedicated unit to be placed between each AXI port of HAs and a port of the AXI SmartConnect, hence increasing the design complexity and the overall resource consumption on the FPGA fabric. It is worth mentioning that modern FPGA SoC platforms integrate specific blocks to manage the QoS in AXI, such as the ARM QoS-400 block implemented in the Xilinx UltraScale+ platform (see [6], p.375). These modules are implemented in the PS of the SoC, hence allowing to manage the QoS of transactions in the PS only. Note that, after request for transactions issued by different HAs in the FPGA enter the PS through the FPGA-PS interface, there are no signals to distinguish them. Therefore, the QoS-400 does not allow controlling the bus bandwidth provided to each individual HA. However, this is essential in a virtualized mixed-criticality system in which isolation between different applications must be ensured.

Research efforts have also been spent to propose arbitration policies for on-chip interconnects to improve throughput and predictability [12]–[15] but not focusing on the AXI standard. Application-dependent reconfigurable arbiters have been presented by Yuan et al. [16] and Sousa et al. [17] to exploit different arbitration schemes. We do not exclude that our research findings may benefit of them. However, we believe that their integration requires a far more complex logic than the one required by the approach proposed in this paper, also increasing area consumption and possibly latency.

IV. Considered framework

This work is focused on supporting mixed-criticality applications upon the same FPGA SoC. Each application comprises a software system running on the PS and a set of HAs. Applications can be independently developed and are finally subject to an *integration* phase before deployment, which is performed by a *system integrator*. Isolation between the applications must be ensured. Standard hypervisor technology is considered to support the isolation of the software parts of the applications (which can even rely on different operating systems). A new component, named AXI HyperConnect and conceived as a hypervisor-level hardware component (i.e., a hardware extension of the hypervisor), is proposed in this paper to supervise the bus traffic generated by all the HAs.

For the sake of completeness, it is crucial to discuss how the integration phase is performed. Although HAs can be provided to the system integrator in different forms (e.g., open-source, purchased with closed-source), standardized solutions are available to simplify integration: we assume that the IP description is provided in an XML format such as the popular IP-XACT. Following several standard practices, each HA implements an interface composed of an AXI control slave interface and an AXI master interface, as discussed in the previous section. The system integrator is then in charge of embedding all the HAs provided by the various applications into an FPGA design. Note that this also requires connecting them to the PS using a system integration tool (e.g., Xilinx Vivado, Intel Platform Designer). Each AXI master port provided by the HAs is connected to an input slave port of a AXI HyperConnect. The master port of the AXI HyperConnect is connected to the FPGA-PS interface, while the AXI slave ports of the HAs are connected to the PS-FPGA interface. Once all the HAs have been connected, the system integrator uses a synthesis tool (e.g., Xilinx Vivado, Intel Quartus Prime) to synthesize

the overall design for the target platform. Finally, the synthesis tool produces a bitstream file that contains the configuration of the FPGA fabric. The bitstream file can be programmed on the FPGA fabric by either the boot loader or the hypervisor, i.e., it is not under the control of the applications, which are also denied in configuring the FPGA. The hypervisor is in charge of granting access from each application to the corresponding HAs only (via standard memory virtualization), as well as routing their interrupts. Furthermore, the hypervisor is in charge of configuring the AXI HyperConnect — this aspect is novel and discussed next.

V. The AXI HyperConnect

This section presents the AXI HyperConnect. Its key features and the corresponding motivations are first presented in Sec. V-A. Then, its internal hardware architecture is discussed in Sec. V-B.

A. Key features

Openness. Typically, state-of-the-art AXI interconnects for commercial-off-the-shelf (COTS) platforms are provided as closed-source IPs. For instance, this is the case of the AXI SmartConnect by Xilinx. The documentation provided with such interconnects usually defines their functionality but not their internal hardware structure and the corresponding logic with a high level of detail. Unfortunately, this information is generally not sufficient to understand the worst-case timing behavior of the AXI interconnect, which can only be inferred with profiling techniques. As a matter of fact, their adoption is not advisable in safety-critical applications that require strict temporal guarantees, where predictability is mandatory to certify the system. The AXI HyperConnect addresses this issue by coming with a slim and open architecture (see Section V-B), making it amenable to low-level inspection to extract worst-case timing bounds, as well as to the corresponding validation. Furthermore, the AXI HyperConnect comes with an open-source driver to control it.

Low latency and resource consumption. A large worst-case propagation latency can be hazardous for safety-critical applications. Furthermore, if an interconnect leaves room for rare pathological cases in which the maximum latency occurs, the pessimism in worst-case analysis for critical systems is increased. One of the significant advantages of developing AXI HyperConnect from scratch has been the possibility of defining its internal architecture with a high degree of freedom. Indeed, a key design principle for the AXI HyperConnect has been to keep a low fixed latency. The experimental results reported in Section VI show that it improves the propagation latency with respect to the state-of-the-art AXI interconnects while maintaining the same throughput. Its development in the HDL language also allows obtaining low resource consumption on the FPGA fabric with respect to the state-of-the-art AXI interconnects (see Section VI).

Bandwidth reservation. State-of-the-art AXI interconnects lack of mechanisms to reserve a portion of the bus/memory bandwidth to a given HA *independently* of the behavior of the others. However, bandwidth reservation for each individual HA is a crucial feature for mixed-criticality applications, as it is essential to guarantee isolation. For instance, if not present, a misbehaving HA of a low-criticality application is free to flood the bus and inject large delays on a HA of a high-criticality application. AXI HyperConnect implements the bandwidth reservation mechanism proposed in [10], which works by limiting the number of transactions to a given budget within periodic time windows. This mechanism is configured by the hypervisor, which allows reserving a given bus bandwidth to each HA and also controlling the overall memory traffic coming from the FPGA fabric directed to the shared memory subsystem (which can delay the execution of software running on the processors of the PS).

Fair bandwidth distribution. As introduced in Section II, state-of-the-art AXI interconnects solve conflicts via round-robin arbitration. However, recent research [11] demonstrated that round-robin arbitration applied to AXI may lead to a complete unfair bandwidth distribution among HAs in the presence of heterogeneous burst sizes (the Xilinx SmartConnect is affected by this issue). This problem is particularly relevant in the considered framework as bandwidth-stealer HA could be deployed to jeopardize the entire FPGA subsystem (including HAs of critical applications). To overcome this issue, AXI HyperConnect implements the mechanisms proposed in [11] to equalize bus transactions to a nominal burst size and liming the number of outstanding transactions. Combined with bandwidth reservation, this mechanism guarantees a very predictable bus access as both the number of transactions (i.e., the reservation budget) and the corresponding data size (i.e., the nominal burst) are implicitly bounded in any time window under analysis.

Runtime reconfiguration. State-of-the-art AXI interconnects are not conceived to change their configuration at run-time. Indeed, they are designed for being configured at system integration time and then synthesized and programmed on the FPGA fabric with a static configuration. This fact determines a strong limitation when the bus and shared memory subsystems require some form of run-time management, e.g., to control the bus interference or to dynamically adapt the bandwidth reserved to a bus master. This limitation has even a much higher impact in applications that leverage dynamic partial reconfiguration (in which the implemented HAs can change at run-time). To overcome this limitation, AXI HyperConnect exports a control AXI slave interface that allows changing its configuration from the PS as a standard memory-mapped device. In the considered framework, this control interface is managed by the hypervisor.

Decoupling from the memory subsystem. AXI HyperConnect allows to individually enable/disable at runtime the access to the PS (and hence to the memory subsystem) for each HA connected to its slave ports. This feature allows isolating misbehaving/malicious HAs (even due to faulty silicon) detected in the system. It can be controlled by just acting on a memory-mapped register of the AXI HyperConnect's control interface. Note that this feature provides different capabilities with respect to the use of an I/O MMU because faulty HAs may generate transactions to any memory address and the source of such transactions is indistinguishable when the I/O MMU is reached, i.e., after the FPGA-PS interface. Furthermore, even if assuming that faulty HAs access some given memory addresses only, the mechanism offered by the AXI HyperConnect is far more simpler and lightweight as it does not require navigating page tables nor to invalidate translation buffers. Finally, it is worth mentioning that the AXI HyperConnect decouples all the signals of a disabled slave port, which is an useful feature under dynamic partial reconfiguration.

Compatibility. All the features offered by the AXI HyperConnect have been developed to be compliant with the AXI standard. This means that the AXI HyperConnect is completely transparent to both the HAs and the memory subsystem and can hence be installed in place of state-of-the-art interconnects without any extra effort. The AXI HyperConnect is compatible with both AXI3 and AXI4 devices. As today's FPGA SoC platforms do not implement out-of-order transactions at the memory controller (i.e. the transactions are served in-order by the memory controller [5], [6]), AXI HyperConnect does

not currently support out-of-order completion. The implementation of this feature is left as a future work to make the AXI HyperConnect compatible with future platforms.

B. Internal architecture

This section describes the hardware architecture of the AXI HyperConnect and its internal components. The architecture has been conceived to be as slim as possible and to introduce a *predictable* propagation delay on each transaction. The proposed architecture makes AXI HyperConnect prone to worst-case timing analysis, which is not addressed here due to lack of space. Figure 2 shows the hardware architecture of the AXI HyperConnect with N input ports.

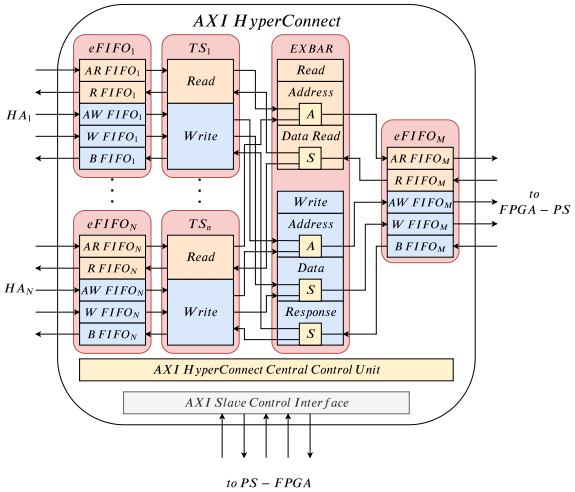

Fig. 2: Architecture of the AXI HyperConnect.

Each of the AXI slave input port of the AXI HyperConnect is handled by a module named *efficient first-in-first-out queuing* (eFIFO). Each eFIFO module is in turn connected to another module named *transaction supervisor* (TS), which monitors and acts on the transactions issued by the HAs to the end of enforcing bandwidth reservation and fair bandwidth distribution, as well as implementing the decoupling from the memory subsystem as discussed in the previous section. All the TS modules are connected to a single module named *efficient crossbar* (EXBAR). The EXBAR is connected to a buffered AXI master interface implemented by another eFIFO module. The architecture is pipelined. The AXI HyperConnect also includes a central unit to manage common configurations to multiple components (details are provided in the following) and resets signals. A description of the main internal components is provided next.

eFIFO. eFIFO modules represent the external interfaces of the AXI HyperConnect for the HAs and the FPGA-PS interface. The eFIFO module is a buffered AXI interface that comes in two variants: a slave and a master. Precisely, each HA is connected to an eFIFO module (AXI slave port) while the eFIFO at the AXI master port of the AXI HyperConnect is connected to the FPGA-PS interface. Each eFIFO module defines five independent FIFO queues, one for each AXI channel (see Figure 2), and does not apply any modification on the incoming addresses or data. Each of such queues is implemented as a proactive (i.e., always ready to receive) circular buffer. This allows each FIFO queue to introduce a latency of just one clock cycle, as well as to improve in terms of latency upon the AXI Stream data FIFOs available for COTS platforms by 66% (see [18], p. 8). The eFIFO module also implements a decoupling mechanism. When a HA is decoupled, the AXI handshake signals on all the AXI channels are kept low, not allowing the HA connected to them to exchange data with the AXI HyperConnect. Also, the other signals are set to ground, hence completely disconnecting the HA from the system.

Transaction supervisor. The TS is the core module of the AXI HyperConnect concerning bandwidth and memory access management. As read and write transactions are independently managed in AXI (thanks to parallel channels), the TS can be conceptually split into two independent subsystems named *read management* and *write management*. The TS implements the logic proposed in [11] to equalize transactions to a nominal burst size (configurable via the control interface of the AXI HyperConnect). Specifically, the read management subsystem splits read requests into sub-requests with nominal burst size and merges incoming data. The write subsystem splits write requests and, accordingly, splits data and merges write responses. Furthermore, both the subsystems equalize the number of outstanding transactions that can be issued by each input port to a programmable value. Please refer to [11] for further details. The TS also integrates a reservation mechanism similar to the one proposed in [10]. Via its control interface, the AXI HyperConnect allows configuring a budget of transactions reserved to each input port that is periodically recharged every T time units, with T being another configurable parameter common to all input ports. The former are referred to as *reservation budgets*, while the latter to as *reservation period*. The TS is in charge of counting the number of transactions at run-time and ensuring that the reservation budget is never exceeded. The reservation period is recharged for all the TS modules by the central unit in a synchronous manner. The propagation latency introduced by the TS module is just one clock cycle on each address request (both on read and write transactions), independently of the burst size. Read, write, and write response channels are managed proactively such that no extra latency is introduced on these channels (the information they carry can be predicted by the corresponding address request channels).

EXBAR. The EXBAR is a low-latency crossbar in charge of solving the conflicts experienced by read/write address requests propagated by the TS modules. To this end, it implements round-robin arbitration with a fixed granularity of one transaction per TS module in each round-cycle. Note that we experimentally found that state-of-the-art interconnects, such as the AXI SmartConnect, adopts a variable round-robin granularity that introduces pessimism in terms of predictability (in the worst case, a TS module can be interfered by $g \times (N-1)$ transactions, where g is the *maximum* round-robin granularity). The EXBAR is also in charge of keeping track of the order in which the address requests are granted, referred to as *routing information*. Routing information is stored in a temporary internal memory of the EXBAR implemented as a circular buffer. The EXBAR introduces a propagation latency of just one clock cycle per address request. The EXBAR proactively routes the read, write, and write response channels according to the stored routing information and hence does not introduce any extra latency on these channels.

VI. Experimental Evaluation

This section presents an experimental evaluation that has been conducted to assess the performance of the AXI HyperConnect on modern FPGA SoC platforms. Section VI-B reports the propagation latency and the throughput of the AXI HyperConnect, while Section VI-C presents a case study focused on the CHaiDNN deep neural network (DNN) accelerator by Xilinx [19]. Finally, Section VI-D

reports the resource consumption of the AXI HyperConnect. In all such sections, the performance of the AXI HyperConnect is compared against the AXI SmartConnect, i.e., the state-of-the-art AXI interconnect by Xilinx [7] (note that the AXI Interconnect [20] has been deprecated by Xilinx). Being the AXI SmartConnect closed-source, a comparison against its internals is unfortunately not possible.

A. Experimental setup

The experiments have been conducted on both a Xilinx ZYNQ Z-7020 platform and a Xilinx ZCU102 ZYNQ Ultrascale+ platform, obtaining similar results. Due to lack of space, we report just the results for the ZYNQ Ultrascale+ platform. The considered architecture is composed of two HAs connected to the PS (and hence to the memory subsystem) through an AXI HyperConnect or an AXI SmartConnect — please refer to Figure 1 assuming $N = 2$. The implementation and synthesis of all the considered components has been performed using Xilinx Vivado 2018.2. In all the experiments, the configuration of the AXI SmartConnect is left as the default one auto-tuned by the Xilinx Vivado tool.

B. Propagation latency and response times in isolation

This experiment aims at comparing **(i)** the propagation latency introduced on each AXI channel and **(ii)** the memory access time for different amount of data, both between the case in which the AXI HyperConnect is used and the one in which the AXI SmartConnect is used. Two Xilinx AXI DMAs [21] have been chosen as representative HAs. This choice has been made because they can mimic the behavior on the bus of many HAs and because their are capable of saturating the maximum memory bandwidth provided by both the considered platforms. As they can stress the bus, note that this choice allows highlighting possible differences in latency and throughput between AXI Smartconnect and AXI HyperConnect. To achieve accurate measurements, latency and throughput have been measured using a custom-developed timer implemented in the FPGA fabric. Figure 3(a)

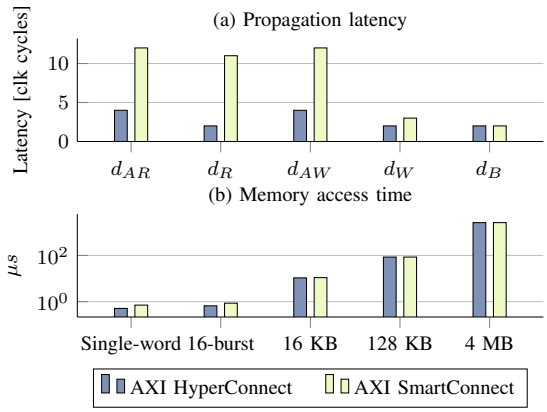

Fig. 3: Timing performance of the AXI HyperConnect.

reports the measured propagation latency on each AXI channel. The latency in the propagation of an address request on address read and address write channels (namely d_{AR} and d_{AW}, respectively) in AXI HyperConnect is equal to four clock cycles. More specifically, one clock cycle is spent on the slave interface of the eFIFO, one clock cycle is spent on the TS module, one clock cycle on the EXBAR, and one clock cycle in the master interface eFIFO at the output port of the AXI HyperConnect. The propagation latency on data read, data write, and write response channels (namely, d_R, d_W, and d_B, respectively) is two clock cycles (one for traversing the slave eFIFO and one for traversing the master eFIFO). Figure 3(a) shows that AXI HyperConnect improves the propagation latency of the 66% on AR and AW channels, 82% on R channel, 33% on W channel with respect to the ones introduced by the AXI SmartConnect, while keeping the same latency on the B channel. Considering that the overall propagation latency introduced on a read transaction is equal to $d_{AR} + d_R$ and that the overall propagation latency on a write transaction is equal to $d_{AW} + d_W + d_B$, the AXI HyperConnect improves the propagation latency by 74% on each read transaction and by 41% on each write transaction, with respect to the AXI SmartConnect. Figure 3(b) shows the maximum memory access time measured to access different amounts of data. Average times differ by less than 5% with respect to maximum times and are not reported to improve the readability of the figure. The results show that, thanks to the improvement in propagation latency of AXI HyperConnect, the response times for single-word transactions and 16-word burst transactions are improved by 28% and 25%, respectively, against the use of the AXI SmartConnect. Note that this improvement in latency does not come at cost of a reduced throughput. Indeed, the AXI HyperConnect shows a comparable throughput with respect to the AXI SmartConnect in accessing 16 KB of data (corresponding to 256 bursts of 16-word transactions) and 4 MB of data (corresponding to 65536 bursts of 16-word transactions).

C. Case study: CHaiDNN + one interfering HA

The performance of the AXI HyperConnect has also been evaluated with a state-of-the-art acceleration framework for DNNs on FPGA SoCs, namely CHaiDNN by Xilinx, which provides functionalities to accelerate on FPGAs the most common operations required during the inference phase of DNNs. The inference is managed by a software task running in the PS, which performs acceleration requests to the CHaiDNN hardware accelerator programmed on the FPGA fabric, referred to as HA_{CHaiDNN}. CHaiDNN works with a shared-memory communication paradigm with the PS. Once activated, HA_{CHaiDNN} uses its master port to read/write the data from/to the shared DRAM memory in the PS. In our case study, HA_{CHaiDNN} executes together with a high-throughput HA, named HA_{DMA}, implemented by a Xilinx AXI DMA. HA_{DMA} is set to read 4 MB of data from the memory subsystem and write back other 4 MB of data. For instance, HA_{DMA} may mimic the bus traffic generated by a video or audio processing engine. The CHaiDNN framework has been configured to accelerate the quantized GoogleNet neural network provided with the standard CHaiDNN distribution. The architecture of the case study is still the one reported in Figure 1 with $N = 2$, where HA_1 is the entire CHaiDNN accelerator subsystem (HA_{CHaiDNN}), and HA_2 is HA_{DMA}. For the ChaiDNN accelerator, it is useful to measure the frames per second it is able to process as a performance index. For HA_{DMA}, as a performance index we measured the number of times the DMA is capable of completing its work (move 4MB of data) in a second. Note that the two performance indexes are never compared between each other. To keep a compact notation, they are both referred to *rate per second* or just *performance* in the following. The performance in isolation (i.e., when one HA at a time is executing) of the two HAs has been measured by both using the AXI HyperConnect and the AXI SmartConnect. The results are reported in Figure 4: as it can be noted from the plot, no performance degradation is experienced when using the AXI HyperConnect with respect to the use of the AXI SmartConnect.

Subsequently, we evaluated the performance of the system in the presence of contention (both CHaiDNN and HA_{DMA} are active at

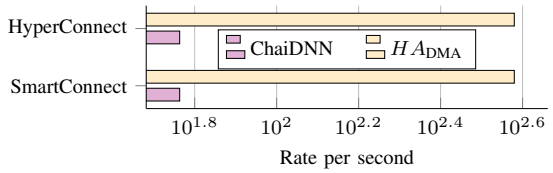

Fig. 4: Comparison on AXI HyperConnect and AXI SmartConnect for CHaiDNN and HA_{DMA} in isolation.

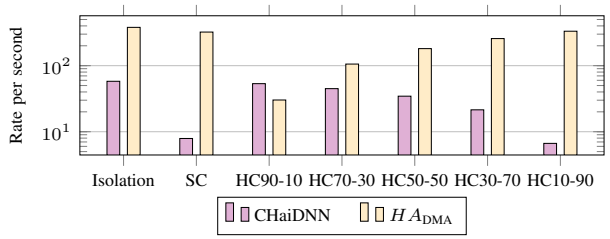

Fig. 5: Performance evaluation.

the same time). The results are reported in Figure 5. The first pair of bars represent the results in isolation and are reported for comparison purposes. The second pair of bars reports the performance of the CHaiDNN and HA_{DMA} when using the AXI SmartConnect. Since HA_{DMA} is more greedy in accessing the bus than HA_{CHaiDNN}, it can take most of the bandwidth while HA_{ChaiDNN} can dispose of just a little portion of the bus bandwidth. It is worth noting that, using the state-of-the-art AXI SmartConnect, there is no way to redistribute the bandwidth between HA_{DMA} and HA_{CHaiDNN} in a different way. However, this is not the case for the AXI HyperConnect. The last five pairs of bars in Figure 5 report the performance of CHaiDNN and HA_{DMA} when the AXI HyperConnect is used. The bandwidth reservation mechanism is enabled to distribute a percentage X of bus bandwidth to HA_{ChaiDNN}, and the remaining percentage Y = 100-X to HA_{DMA} (referred to as HC-X-Y) in Figure 5). In the third group of bars in the figure, the AXI HyperConnect has been configured to assign 90% of the bandwidth to HA_{ChaiDNN} (HC-90-10): this scenario shows a performance for CHaiDNN that is close to the one in isolation. A similar consideration can be made when assigning 70% of the bandwidth to HA_{CHaiDNN} and 30% to HA_{DMA} (HC-70-30), and the other configurations.

D. Resource consumption

Table I reports the measured resource consumption for the two-input AXI HyperConnect used in the case study discussed above on the Xilinx ZCU102 platform. The results confirm that the design of a slim architecture and the development in VHDL language allows containing the resource consumption with respect to the state-of-the-art AXI SmartConnect. Note that, despite being characterized by lower resource consumption, the AXI HyperConnect also implements functionalities that are not present in the AXI SmartConnect.

TABLE I: Resource consumption

ZCU102	Resources			
	LUT (274080)	**FF** (548160)	**BRAM**	**DSP**
HyperConnect	3020 (11%)	1289 (0.3%)	0	0
SmartConnect	3785 (14%)	7137 (1.3%)	0	0

VII. Conclusions

FPGA-based SoCs are attracting growing interest in the development of mixed-criticality systems. However, current virtualization support for the FPGA subsystem is not providing sufficient isolation for critical applications. This paper introduced the AXI HyperConnect: a hypervisor-level, predictable AXI interconnect for FPGA SoCs. Its features have been presented and motivated, together with its slim internal architecture. The AXI HyperConnect has been implemented for two commercial FPGA SoC platforms, and its performance has been compared against the ones of the state-of-the-art AXI interconnect from Xilinx, both on synthetic traffic and on a popular framework for deep neural network acceleration. The results show an improvement in propagation latency, while keeping a comparable throughput and resource consumption.

Acknowledgment

This work has been partially supported by the Italian Ministry of Education, University and Research (MIUR) under the project SPHERE: Software Architecture for Predictable Heterogeneous Real-time systems, grant no. 20172NNB4T.

References

[1] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Modeling and Analysis of Bus Contention for Hardware Accelerators in FPGA SoCs," in *32st Euromicro Conference on Real-Time Systems (ECRTS 2020)*, 2020.

[2] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *11th Real-Time Linux Workshop*. Citeseer, 2009, pp. 263–272.

[3] F. Restuccia, A. Biondi, M. Marinoni, and G. Buttazzo, "Safely Preventing Unbounded Delays During Bus Transactions in FPGA-based SoC," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020.

[4] *Platform Designer System Design Tutorial*, Intel Corp., aN 812.

[5] *Zynq-7000 - Technical Reference Manual, UG585*, Xilinx.

[6] *Zynq UltraScale+ - Technical Reference Manual, UG1085*, Xilinx.

[7] *SmartConnect, LogiCORE IP Product Guide*, Xilinx, 2018, pG247.

[8] A. K. Jain, K. D. Pham, J. Cui, S. A. Fahmy, and D. L. Maskell, "Virtualized execution and management of hardware tasks on a hybrid ARM-FPGA platform," *Journal of Signal Processing Systems*, vol. 77, pp. 61–76, 2014.

[9] T. Xia, Y. Tian, J.-C. Prévotet, and F. NOUVEL, "Ker-one: A new hypervisor managing FPGA reconfigurable accelerators," *Journal of Systems Architecture*, 2019.

[10] M. Pagani, E. Rossi, A. Biondi, M. Marinoni, G. Lipari, and G. Buttazzo, "A bandwidth reservation mechanism for AXI-based hardware accelerators on FPGAs," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[11] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? restoring fairness in AXI interconnects for FPGA SoCs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 51, 2019.

[12] T. D. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das, and V. Degalahal, "A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks," in *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*. IEEE, 2006, pp. 8–pp.

[13] P. Burgio, M. Ruggiero, F. Esposito, M. Marinoni, G. Buttazzo, and L. Benini, "Adaptive tdma bus allocation and elastic scheduling: A unified approach for enhancing robustness in multi-core rt systems," in *Computer Design (ICCD), 2010 IEEE International Conference on*. IEEE, 2010, pp. 187–194.

[14] H. Shah, A. Raabe, and A. Knoll, "Priority division: A high-speed shared-memory bus arbitration with bounded latency," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–4.

[15] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "The lotterybus on-chip communication architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 6, pp. 596–608, 2006.

[16] C.-C. Yuan, Y.-J. Huang, S.-J. Lin, and K.-h. Huang, "A reconfigurable arbiter for soc applications," in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*. IEEE, 2008, pp. 713–716.

[17] É. Sousa, D. Gangadharan, F. Hannig, and J. Teich, "Runtime reconfigurable bus arbitration for concurrent applications on heterogeneous mpsoc architectures," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 74–81.

[18] *AXI4-Stream FIFO v4.1 LogiCORE IP Product Guide*, Xilinx, pG080.

[19] *CHaiDNN official github.*, Xilinx, https://github.com/Xilinx/chaidnn.

[20] *AXI Interconnect, LogiCORE IP Product Guide, PG059*, Xilinx.

[21] *AXI Central Direct Memory Access, LogiCORE IP Product Guide, PG034*, Xilinx.