Bounding Memory Access Times in Multi-Accelerator Architectures on FPGA SoCs

Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo

Abstract—Modern FPGA System-on-Chips (SoCs) embed large FPGA logics capable of hosting multiple hardware accelerators. Typically, hardware accelerators require direct access to the shared DRAM memory for reaching the high performance demanded by modern applications. In commercial FPGA SoCs, this goal is achieved by interconnecting the hardware accelerators on an interconnect based on AMBA AXI, which is the de-facto industrial standard for on-chip communications. The AXI standard provides great flexibility in the definition of the network topology. Nevertheless, such flexibility generates a significant unpredictability when attempting to bound the hardware accelerators' response time when executing under contention. This work focus on bounding the worst-case memory access time of hardware accelerators deployed on commercial FPGA SoCs. We propose a modeling and analysis technique to bound the response time of the hardware accelerators and evaluate the schedulability of a system applicable to arbitrary AXI-based bus structures deployed on FPGA SoCs. Our results are validated on real execution traces collected on two popular FPGA SoCs belonging to the Xilinx ZYNQ-7000 and Zynq-Ultrascale+ families and by simulated results.

Index Terms—on-chip communications, cyber-physical systems, timing analysis, real-time systems, safety-critical systems.

1 Introduction

Embedded computing platforms evolved toward heterogeneous architectures to support the increasing computational workload generated by modern *cyber-physical systems* (CPS), as self-driving cars, autonomous robots, smart production plants. Such systems must process a huge amount of sensory data in real-time to meet stringent timing constraints imposed by the interaction with the physical environment. A significant amount of processing is performed by deep learning algorithms, which can be efficiently accelerated in Field Programmable Gate Arrays platforms (FPGAs) or General Purpose Graphical Processing Unit platforms (GPGPUs). Today, such *hardware accelerators* (HAs) are available in commercial heterogeneous computing platforms, as the Zynq Ultrascale+ by Xilinx, which integrates in the same chip different types of multicore processors and a large FPGA fabric, or the Xavier from Nvidia, which includes a GPGPU and specialized accelerators for machine learning algorithms.

When developing safety-critical software for CPS, a crucial issue is to guarantee *timing constraints* for the application tasks. This problem is particularly challenging when hardware acceleration is involved, especially when no internal architecture details are publicly available, as for Nvidia GPU platforms. This is particularly relevant when multiple HAs perform memory-intensive operations that cause several contentions in accessing shared resources, as buses and memory controllers.

FPGA-based acceleration represents a promising solution for coping with these problems since it provides a powerful and energy-efficient computation with a very regular clock-level timing behavior [3], [20]. As a result, the execution time of a HA that runs in isolation has very low fluctuations and hence is quite predictable. One of the major threats to predictability for hardware accelerators deployed on FPGA SoC platforms is due to contentions that may occur while accessing the bus and the memory controller. This issue can properly be addressed since FPGAs expose a precise control on the bus structure to the system integrator designers, which can organize the bus hierarchy to match timing constraints and deploy custom arbitration modules to dispatch memory transactions to the memory controller [1].

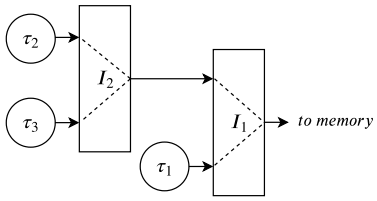

Fig. 1: A typical bus architecture with three HAs connected by two interconnects.

A common approach used for FPGA accelerators in COTS SoCs consists of having a set of HAs that act as managers in accessing the bus during transactions to the principal DRAM off-chip memory shared with the multiple processors [27] [10] [33]. Since the number of ports to access the shared memory is limited, a common solution is to multiplex multiple managers on a single port using the *interconnect* available in the IP library offered by the FPGA vendor. Multiple interconnects can be interconnected to create a hierarchical bus network. Figure 1 illustrates a sample network of interconnects including three hardware accelerators (τ_1, τ_2, and τ_3) and two interconnects (I_1 and I_2).

It is worth observing that the frequency at which the

Francesco Restuccia is with the University of California San Diego; Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo are with the Scuola Superiore Sant'Anna Pisa.
E-mail: frestuccia@ucsd.edu, {marco.pagani, alessandro.biondi, mauro.marinoni, giorgio.buttazzo}@sssup.it

FPGA fabric operates in commercial FPGA SoCs is much less than the one used by the on-chip memory controller (this latter is realized in hard silicon and placed outside of the FPGA fabric) and the memory itself. To make an example, the default operating frequency of the FPGA in a Xilinx Zynq-7000 is 100 MHz. Differently, the frequency of the Processing System (including the memory controller) is 650 MHz. As a consequence, the delays originating from the bus infrastructure implemented on the FPGA are comparable to the ones required by memory accesses, and thus cannot be neglected when computing the response times of HAs.

Contributions. This work investigates the major sources of delay in HAs memory access time and presents a worst-case response time analysis for hardware accelerators deployed on FPGA SoC platforms. We considered the AXI standard [2] for three main reasons: **(1)** AXI is the most widely adopted standard for communication on commercial FPGA SoCs [31] [13]; **(2)** AXI is the default option in well-established design tools for FPGA platforms, as Xilinx Vivado [30] and Intel Quartus Prime [14]; **(3)** AXI is the default interface leveraged by several commercial hardware accelerator modules for bus communications. The rest of the paper is organized as following described: Section 2 proposes a detailed model of the AXI bus and AXI interconnects. The proposed model accounts for the behavior of commercial AXI interconnects and the delays experienced by bus transactions. Section 3 proposes a worst-case response-time analysis bounding the response time of recurrent hardware accelerators concurrently accessing the shared memory in PS through hierarchical networks of AXI interconnects. Finally, Section 4 reports our experimental validation, split into three sets of experiments. The first set of experiments validates the proposed model on real hardware waveform tracks obtained from two modern FPGA SoC platforms from Xilinx. The second set presents a case study deployed and running on the same platforms and compares real execution measurements with two bounds built out of the proposed analysis. At last, the third set presents the experimental results obtained with a synthetic workload.

This work extends the results of [25] by providing the following new contributions: **(i)** An extended system model considering the pipelining structure of AXI interconnects and hierarchical interconnections; **(ii)** A reformulated worst-case analysis including new lemmas that copes with the limits of realistic HW-tasks and the pipelining of AXI interconnects; **(iii)** An updated overall recursive bound on the maximum amount of interfering transactions, computed by combining the results of the new proposed lemmas; **(iv)** A less pessimistic algorithm for bounding the response times of the HW-tasks and checking the system schedulability, considering all the proposed improvements; **(v)** An extended experimental evaluation, showing the benefits of the proposed improvements. The comparison shows a considerable reduction of pessimism with respect to [25] thanks to the novel contributions provided in this paper.

2 System model

This work considers systems involving multiple manager hardware accelerators implemented on an FPGA SoC platform and leveraging the AXI standard for communication. These hardware accelerators can autonomously access a shared DRAM memory through a memory controller located on the PS side of the FPGA SoC platform.

2.1 HW-task model

Each hardware accelerator is supposed to implement a distinct functionality. In the rest of the paper, the name *hardware tasks* (or HW-tasks for short) will also be used to indicate them. Each HW-task relies on an AXI manager interface to autonomously read and write from/to the shared DRAM memory. The generic HW-task τ_i is periodically executed every T_i clock cycles. Thus, it generates a periodic sequence of instances referred to as *jobs*. For each job, τ_i: **(1)** issues N_i^R read transactions and N_i^W write transactions. The transactions have burst length B; **(2)** has at most ϕ_i outstanding transactions per channel. In other words, at any time τ_i have at most ϕ_i pending read transactions and ϕ_i pending write transactions; **(3)** computes for at most C_i clock cycles; and **(4)** has a relative deadline equal to T_i. Thus, each job of τ_i must complete before the release of the next one. We assume that read and write transactions are independent of each other. According to the AXI standard, read transactions and write transactions are propagated through separated channels. Thus, they do not influence each other in data propagation. It is important to note that no assumptions are made on memory access patterns for the HW-tasks to keep the paper general and robust regarding the behavior of HW-tasks. Hence, an arbitrary temporal distribution of memory transactions across the jobs must be considered. At the same time, this assumption limits the exploitation of the parallelism provided by the AXI standard. In the worst-case scenario, HW-tasks cannot fully exploit parallelism since transactions can be sufficiently spread far apart.

2.2 AXI interconnect model

The system includes multiple AXI interconnects joined one on top of the other in a hierarchical fashion, creating a network of interconnects. The generic interconnect in the network I_j exports S_j subordinate ports and a single manager port. Since each interconnect exports a single manager port, the incoming traffic (at the manager port and) directed to the subordinate ports does not experience any conflict. Differently, conflicts can be encountered by requests for transactions of the same kind (read or write) issued by distinct HW-tasks. Such conflicts are typically solved by independent, per-channel, arbiters (see [32], [35]). Round-robin arbiters have a granularity of ϕ_I requests, meaning that the manager port grants at most ϕ_I read requests (or respectively, write requests) to each HW-task at each round-robin cycle. In this work, all interconnects have the same parameter ϕ_I. This allows easing the notation in the analysis presented in Section 3 (the case of distinct round-robin granularities for each interconnect can be easily incorporated in the analysis).

Each interconnect delays address and data propagation introducing a propagation delay, denoted as: **(i)** $d_{\text{Int}}^{\text{addr}}$ as the propagation delay on address requests, **(ii)** $d_{\text{Int}}^{\text{data}}$ as the propagation delay of a data word (read or write), and **(iii)** $d_{\text{Int}}^{\text{bresp}}$ as the propagation delay of write response. Such delays are inferred from the documentation of the AXI interconnect

under analysis (whenever such documentation is provided by the vendor) or by means of specific experimental profiling. Propagation delays are the consequence of multiple operations operated by the AXI interconnect on requests, data, and write responses. For each channel, such operations are performed by a corresponding pipeline composed of a series of internal stages, such as input buffering, decoding, optional resizing, routing, output buffering, etc. The sum of the maximum execution times of all of the stages traversed by an address request while propagating through the AXI interconnect is equal to $d_{\text{Int}}^{\text{addr}}$. In a similar way, the sum of all of the maximum execution times of the stages traversed by data and write responses are equal to $d_{\text{Int}}^{\text{data}}$ and $d_{\text{Int}}^{\text{bresp}}$, respectively. Each pipeline stage in a chain is managed in parallel by the AXI interconnect. It is important to note that this implies that multiple address requests, data, and write responses can be propagated in parallel along a network of interconnects. This observation is leveraged in Section 3 to reduce the pessimism of our worst-case analysis. In the following, we assume that the pipeline of the interconnects never gets full. Note that the main cause for the pipeline to get full is the presence of one or multiple HAs stalling the bus (i.e., hanging on the data phase). In this work, this situation is considered a misbehavior of HAs and hence not addressed in the following. Our results can however be extended to cope with misbehaving HAs by considering the results of [22]: this is left as future work.

The hold times are the numbers of clock cycles for which some information must remain on the bus to be correctly sampled by a module connected to the bus (i.e., interconnect or HW-task). They are denoted by the following terms: **(i)** t_{addr} is the hold time of an address request, **(ii)** t_{data} is the hold time of a word of data, and **(iii)** t_{bresp} is the hold time of a write response. The hold times are assumed not to be lower than the maximum execution time of the stages of the corresponding interconnect pipeline. According to the AXI standard, requests, data, and write responses are propagated on separated channels. Thus, they do not interfere with each other in propagation.

2.3 Processing System and Memory Controller model

In a typical FPGA SoC architecture, the FPGA fabric is combined with a *Processing System* (PS), which generally includes multiple processors and peripherals. The DRAM memory controller is placed in PS and is shared among the HW-tasks deployed in the FPGA fabric and the devices embedded in the PS. The HW-tasks deployed in the FPGA fabric access the DRAM memory through the FPGA-PS interface. In modern FPGA SoC platforms, the FPGA-PS interface exports a set of secondary ports based on the ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI standard). Each HW-task is an active entity exporting an AXI manager port through which it can generate requests for memory transactions. Such requests are submitted to the shared DRAM memory controller through the FPGA-PS interface. We assume that each HW-task has a private memory buffer in DRAM to load and store data. This is a typical setting for applications leveraging hardware acceleration.

We consider the scenario in which all hardware accelerators share a single AXI port at the FPGA-PS interface for accessing their memory buffers in DRAM–this is the scenario keeping all the contention of the HW-tasks at the FPGA interconnect. We made this choice as leveraging multiple ports at the FPGA-PS interface moves the contention generated by the HW-tasks from the FPGA fabric to the PS interconnect and DRAM memory controller. Our investigation in this paper is mainly focused on the worst-case effect generated at the FPGA interconnect rather than the contention generated at the PS interconnect and DRAM memory controller (which depends on strategic information typically not released by the vendor). Investigating the contention in PS goes beyond the purpose of this paper—we are planning for such an investigation in future work, aspiring for the provisioning of more detailed information on the internals of the PS interconnect and DRAM memory controller from the vendor.

In commercial FPGA SoC platforms, the shared DRAM memory controller included in the PS is split in two modules: **(1)** an AXI interface module and **(2)** a physical core module, directly accessing the physical DDR memory [28], [31]. The AXI interface block receives and arbitrates the AXI transactions from the AXI subordinate ports of the memory controller. The DDR physical core schedules and issues the requests to the physical layer and generates control and data signals for the DRAM memory.

Commonly, the internal architecture of the DDR physical core is based on multi-level queues, where throughput and efficiency are maximized by applying dedicated scheduling policies to reorder transactions [11]. The internals of the DDR physical core block on many commercial platforms are not publicly revealed or not well documented, including the queues structure and the scheduling policies. Hence, a fine-grained model of the DDR physical core block is beyond the scope of this paper. Being our focus on analyzing the system interconnect, a coarse-grained modeling of the DRAM-related delays is adopted. It is worth mentioning that our results could be refined if the internals of the DDR controller are known (e.g., by adopting the results from [11]).

The DRAM Memory Controller AXI Interface block guarantees that the requests directed to the shared DRAM are served *in order* (see [28], p. 297, and [31], p. 440). This means that, from the point of view of the HW-tasks, the order of the data read responses follows the order of address read requests granted at the FPGA-PS interface. Likewise, write address requests are handled in order. It is worth mentioning that this feature does not depend on the scheduling policies implemented by the DDR Physical core block, which may affect the worst-case service time of a request due to internal reordering.

From previous considerations, this paper takes into account the following (cumulative) delays introduced by the PS and the memory controller:

- $d_{\text{PS}}^{\text{read}}$ is the maximum time elapsed between the sample of a read transaction at the FPGA-PS interface and the availability of the first word of the corresponding data at the FPGA-PS interface; and
- $d_{\text{PS}}^{\text{write}}$ is the maximum time elapsed between the sample of the last word of data of a write transaction at the FPGA-PS interface and the availability of the corresponding write response at the FPGA-PS interface.

By definition, these delays incorporate the propagation times due to the PS internal logic and the overall service time at the memory controller. Such parameters derive from the internals of the PS and can be obtained from the official documentation furnished by the vendor (if publicly available) or quantified through experimental profiling and over-provisioning. Accurate bounds on the delays introduced by the memory controller can be computed with state-of-the-art techniques [4], [11] provided that its internal architectural details are available.

2.4 Overall architecture

The system under analysis is formally composed of a set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of n HW-tasks, a set $\mathcal{H} = \{I_1, \ldots, I_s\}$ of s AXI interconnects, and a shared DRAM memory controller $\mathcal{M}$ in PS. The HW-tasks $\in \Gamma$ are deployed on a network of AXI interconnects (in the set $\mathcal{H}$) and organized as follows. Each subordinate port of an interconnect is connected to the manager port of a HW-task or to the manager port of another interconnect (in a hierarchical manner). The set of the HW-tasks connected to the interconnect I_j is denoted by $\Gamma(I_j)$. In a similar way, the set of interconnects connected to the subordinate ports of I_j (i.e., in input) is denoted with $\mathcal{H}(I_j)$. The set of HW-tasks that are directly or transitively connected to I_j is denoted by $\Gamma^+(I_j)$ (i.e., whose transactions traverse I_j). The manager port of the interconnect placed at the very bottom of such a hierarchy is connected to the subordinate port of the FPGA-PS interface. We referred to this latter interconnect as the *root* interconnect I_{root} – the transactions released by all of the HW-tasks pass through this interconnect to reach the memory controller in PS. Each interconnect has one manager port. The manager port of the generic interconnect $I_j \neq I_{\text{root}}$ is connected to a subordinate port of another interconnect, denoted by $\beta(I_j)$. For consistency, $\beta(I_{\text{root}}) = \emptyset$. Overall, the system topology is a tree where: **(i)** the root node is represented by I_{root}, **(ii)** the leaves are represented by the HW-tasks in Γ, and **(iii)** the interconnects in $\mathcal{H} \setminus \{I_{\text{root}}\}$ represents the intermediate nodes (see Figure 2(b)). We say that an interconnect I is placed at the hierarchical level L_I when a HW-task connected to I must traverse L_I interconnects to reach the FPGA-PS interface (I_{root} is at first level, i.e., $L_{I_{\text{root}}} = 1$). Table 1 summarizes the symbols used in this paper.

TABLE 1: Symbols used in this paper.

N_i	Number of transactions issued by τ_i (can have superscript R or W)
ϕ_i	Number of maximum outstanding transactions for τ_i
ϕ_I	Transactions granted per round-robin cycle by interconnects
B	Burst length of a transaction
t_{addr}	Single address request hold time
t_{data}	Single data word hold time
t_{bresp}	Single write response hold time
$d_{\text{PS}}^{\text{read}}$	Maximum delay introduced by the PS on a read transaction
$d_{\text{PS}}^{\text{write}}$	Maximum delay introduced by the PS on a write transaction
$d_{\text{Int}}^{\text{data}}$	Data word interconnect propagation latency
$d_{\text{Int}}^{\text{addr}}$	Address request interconnect propagation latency
$d_{\text{Int}}^{\text{bresp}}$	Write response interconnect propagation latency
$\Gamma(I_i)$	Set of the HW-task connected to I_j
$\mathcal{H}(I_j)$	Set of the interconnects connected to subordinate ports of I_j
$\beta(I_j)$	Interconnect connected to the manager port of I_j
$\Gamma^+(I_j)$	Set of HW-tasks whose transactions pass through I_j

3 RESPONSE-TIME ANALYSIS

This section presents a worst-case analysis bounding the worst-case response times of HW-tasks deployed on a hierarchical interconnect network.

We structured the analysis in a set of incremental lemmas: at first, we bound the worst-case response time of one read or write transaction assuming no contention at the interconnects (Section 3.1). Following, we propose three methodologies for bounding the maximum number of interfering transactions affecting the execution of a job of a HW-task under analysis (Section 3.2, Section 3.3, and Section 3.4). The three proposed bounds are then combined in Section 3.5. Finally, Section 3.7 proposes an algorithm leveraging the results of the preceding sections to bound the maximum response time of the HW-tasks and check system schedulability.

The bounds derived in this section can be applied to both read and write transactions. To keep a compact notation, this section uses the simplified symbol N_i in place of N_i^R or N_i^W to represent the number of transactions issued by τ_i.

3.1 No contention at the interconnects

The following lemmas bound the memory access time of one transaction issued by a generic HW-task τ_i under evaluation that is connected to the interconnect I placed at an arbitrary hierarchical level L. The lemmas presented in this section consider the cases in which no bus contention is generated by the other HW-tasks in the system[1]. Two lemmas are provided, one for read and one for write transactions.

Lemma 1. *Let $\tau_i \in \Gamma$ be the HW-task under analysis and connected to interconnect $I_j \in \mathcal{H}$ placed at the L-th hierarchical level. If all the HW-tasks in $\Gamma \setminus \{\tau_i\}$ are not active, i.e., they do not generate interference to τ_i, the worst-case response time of a single read transaction R issued by τ_i is upper bounded by*

$$d^{NoCont,read}(I_j) = t_{addr} + L \cdot d_{Int}^{addr} + d_{PS}^{read} + \\ + L \cdot d_{Int}^{data} + B \cdot t_{data}.$$

Proof. As from the official AXI standard documentation [2], a read transaction R begins with the issue of the address read request R_{addr}, which is then sampled by I_j. The address time is constant and equal to t_{addr}. The latency cost for R_{addr} to traverse the interconnect I_j is bounded by $d_{\text{Int}}^{\text{addr}}$. At this point, R_{addr} goes through the interconnect network tree, traversing the remaining $L - 1$ interconnects. As argue for I_j, each of such interconnects introduces a latency bounded by $d_{\text{Int}}^{\text{addr}}$. Therefore, R_{addr} is available at the manager port of the root interconnect I_{root} after a total propagation delay of $t_{\text{addr}} + L \cdot d_{\text{Int}}^{\text{addr}}$, where it is sampled from the subordinate port of the FPGA-PS interface. The PS routes R_{addr} to the Memory Controller and provides to the FPGA-PS interface the first word of data after at most d_{PS}^{read} time units (see Section 2.3). At this point, the data words R_{data} corresponding to R traverse the L levels of the network of interconnects, in reverse order with respect to R_{addr}, until finally reaching τ_i. Due

1. Note that the contention-free bounds provided by the two lemmas do not pertain to the cases in which the transaction is served in isolation, but rather to cases in which no contention is experienced at the interconnects. This is because the delays introduced in Section 2.3 already cope with conditions of maximum contention at the PS and the memory controller.

to pipelining, being the data words propagated in sequence within the network of interconnects, the propagation latency experienced during the data phase is paid just once for the whole data burst. Hence, given t_{data} as the data time for each word and that $d_{\text{Int}}^{\text{data}}$ is the maximum latency introduced by any interconnect on data words, the overall latency paid to propagate the data burst along the interconnect network is $L \cdot d_{\text{Int}}^{\text{data}} + B \cdot t_{\text{data}}$. The lemma follows by summing up the delay contributions mentioned above. □

Lemma 2. *Under the same hypotheses of Lemma 1, the response time for a write transaction W issued by HW-task τ_i is bounded by*

$$d^{NoCont,write}(I_j) = t_{addr} + L \cdot \max\{d_{Int}^{addr}, d_{Int}^{data}\} + B \cdot t_{data} + \\ + d_{PS}^{write} + t_{bresp} + L \cdot d_{Int}^{bresp}.$$

Proof. As from the official AXI standard documentation [2], the write transaction W begins with the issue of the address write request W_{addr} by τ_i, which lasts t_{addr} time units. As mandated by the AXI standard, once W_{addr} is granted at the interconnect I_j, the HW-task τ_i is granted to provide the corresponding data words W_{data} on the write channel. W_{addr} and W_{data} are propagated through the interconnect network tree on the two corresponding channels, eventually reaching the FPGA-PS interface. Note that data can be propagated only after the corresponding address. Therefore, the latency experienced by W_{addr} and W_{data} when traversing an interconnect is no larger than the maximum between $d_{\text{Int}}^{\text{addr}}$ and $d_{\text{Int}}^{\text{data}}$. Overall, considering all the interconnects up to the FPGA-PS interface, the latency introduced on W_{addr} and the entire burst W_{data} is given by $t_{\text{addr}} + L \cdot \max\{d_{\text{Int}}^{\text{addr}}, d_{\text{Int}}^{\text{data}}\}$, which must be summed to the time to transmit the data themselves, i.e., $B \cdot t_{\text{data}}$. At this point, the PS routes W_{addr} and W_{data} to the memory controller. Following Section 2.3, after at most d_{PS}^{write} time units the write response W_{resp} is available at the FPGA-PS interface. Finally, W_{resp} is propagated through the interconnect tree, until reaching τ_i, experiencing a latency of $t_{\text{bresp}} + L \cdot d_{\text{Int}}^{\text{bresp}}$. The lemma follows by summing up the delay contributions mentioned above. □

Observe that the bounds provided by the two lemmas above just depend on the hierarchical level L at which interconnect I_j is placed, i.e., the one to which the HW-task under analysis is directly connected.

3.2 First bound on the number of interfering transactions

In this lemma, we proceed incrementally starting considering the interference generated at a single interconnect, for instance I_{root} in the most simple case (see Figure 2(a)). The lemma bounds the maximum number of interfering transactions that a transaction issued by the HW-task under analysis can suffer.

Lemma 3. *Consider the interconnect I_{root} and let $\tau_i \in \Gamma(I_{root})$ be the HW-task under analysis. In the worst-case, each address request for transaction issued by τ_i grants the access to the manager port of I_{root} after at most*

$$\sum_{\tau_j \in \Gamma(I_{root}) \setminus \{\tau_i\}} \min(\phi_j, \phi_I) \quad (1)$$

transactions.

Proof. By the model presented in Section 2, the interconnects solve conflicts on address requests issued by different HW-tasks by round-robin. In the worst-case scenario, τ_i is the last HW-task to be served in the round-robin arbitration cycle, i.e., after all the other HW-tasks in $\Gamma(I_{\text{root}})$. By Section 2.1, the maximum number of transactions issued by each HW-task τ_j that can be pending at the same time is ϕ_j. At the same time, by Section 2.2, the maximum number of transactions that an interconnect can grant to each HW-task for each round-robin cycle is ϕ_I. Therefore, I_{root} grants at most $\min(\phi_j, \phi_I)$ transactions for each interfering HW-task $\tau_j \in \Gamma \setminus \{\tau_i\}$ per round-robin cycle. The lemma follows by summing up this contribution for each interfering HW-tasks. □

Once defined Lemma 3, it is possible to proceed with bounding the maximum number of interfering requests in a generic interconnects network. We first observe that a HW-task τ_i can suffer two types of interference: **(1)** *direct interference*, which is the interference suffered by the transactions issued by τ_i at the interconnect to which τ_i is directly connected to; and **(2)** *indirect interference*, being the interference suffered by the transactions issued by τ_i, or other transactions that generate direct interference to τ_i, in other interconnects at shallower hierarchical levels on their way towards the FPGA-PS interface. Following, we provide further details on both kinds of interference.

Direct interference. The reasoning introduced in Lemma 3 can be extended to consider a hierarchical network of interconnects, as the one illustrated in Figure 2(b). It is worth noting that, in this case, *a HW-task* can also experience contention at an interconnect due to transactions issued by HW-tasks connected at higher hierarchical levels. To make an example, τ_i in Figure 2(b) (directly connected to I_{root}) can be interfered by transactions issued by the HW-task τ_z connected to I_1.

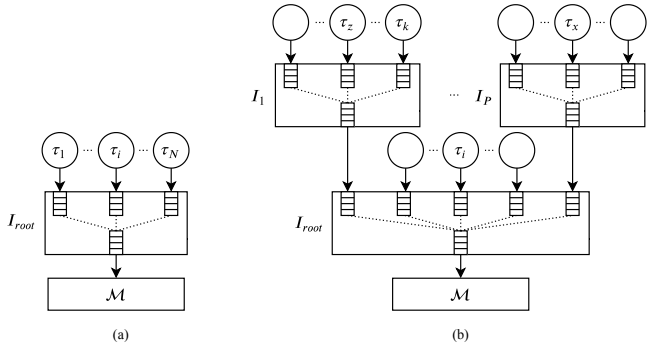

Fig. 2: **(a)** A set of HW-tasks directly connected to I_{root}. **(b)** A sample hierarchical network of interconnects and HW-tasks with two hierarchical levels. Circles are HW-tasks (only the ones mentioned in the text are assigned a name).

Lemma 4. *Consider an arbitrary interconnect I_j. Also, let $\tau_i \in \Gamma(I_j)$ be the HW-task under analysis. In the worst-case, each address request for transaction issued by τ_i reaches the manager port of I_j after at most*

$$Y^{direct}(\tau_i, I_j) = \sum_{\tau_j \in \Gamma(I_j) \setminus \{\tau_i\}} \min(\phi_j, \phi_I) + |\mathcal{H}(I_j)| \times \phi_I \quad (2)$$

transactions.

Proof. Following the model presented in Section 2.2, at each round-robin cycle, I_j serves at most ϕ_I transactions per its subordinate port. Note that this also holds when another interconnect I_h is connected to a subordinate port of I_j. Therefore, from the perspective of τ_i, any bus traffic coming from I_h can interfere by at most ϕ_I transactions per round-robin cycle, independently of the actual configuration of the sub-network connected to I_h. Overall, this means that all interconnects that are directly connected to I_j can interfere with each transaction issued by τ_i with at most $|\mathcal{H}(I_j)| \times \phi_I$ transactions. Finally, the first term of Equation (2) follows due to the same considerations done in the proofs of Lemma 3. Hence the lemma follows. □

Indirect interference. A request issued by the HW-task under analysis can also incur contention at shallower hierarchical levels while it is propagated through the network of interconnects. To make an example, in Figure 2(b) a transaction issued by τ_z can incur contention at I_{root} due to transactions issued by τ_i or τ_x. Moreover, indirect interference can also affect transactions generating direct interference to a request issued by a HW-task under analysis. This effect leads to a *transitive interference phenomenon*. Making an example, in Figure 2(b) a transaction issued by τ_k delaying τ_z in I_1 can experience contention at I_{root} due to a transaction issued by τ_x, hence in turn delaying τ_z too. In such scenario, the transaction of τ_x is transitively delaying τ_z.

Following, we introduce a set of lemmas to account for indirect interference. As done previously, we proceed incrementally, starting considering just two *adjacent* hierarchical levels.

Lemma 5. *Consider an arbitrary interconnect I_j, placed at hierarchical level $L \geq 2$, that issues Δ transactions in output to its manager port. The Δ transactions can be indirectly interfered by at most*

$$Y^{\textit{indirect}}_{\textit{2-level}}(\Delta, I_j) = \Delta \times \\ \times \left(\sum_{\tau_i \in \Gamma(\beta(I_j))} \min(\phi_i, \phi_I) + |\mathcal{H}(\beta(I_j)) \setminus \{I_j\}| \times \phi_I \right) \quad (3)$$

transactions at $\beta(I_j)$ (i.e., at hierarchical level $L-1$).

Proof. Let r one of the Δ transactions issued by I_j. As addressed by Lemma 4, r can incur direct interference at $\beta(I_j)$, i.e., the only interconnect directly connected to I_j at the lower hierarchical level $L-1$. Hence, the interference at $\beta(I_j)$ can be bounded as done for Lemma 4. The only differences here are the following ones: **(i)** Being r coming from another interconnect I_j, the transaction is not originated by a HW-task that is directly connected to $\beta(I_j)$. Therefore, no HW-task needs to be excluded from those that generate interfering transactions (first term in the sum of Eq. (2)). **(ii)** Interconnect I_j, being the one from which the transaction r under analysis is coming from, has instead to be excluded from the set of interconnects that can generate interfering transactions (second term in the sum Eq. (2)). hence the actual set of interconnects to consider is $\mathcal{H}(\beta(I_j)) \setminus \{I_j\}$. The lemma follows by accounting for the bound implied by the above reasoning for each of the Δ transactions issued by I_j. □

After introducing the above lemma, we can generalize the bound on the contribution of indirect interference for an arbitrary hierarchical structure having $L > 2$ levels.

Lemma 6. *Let τ_z be the HW-task under analysis directly connected to interconnect I_j at the hierarchical level $L \geq 2$. The total number of transactions that interfere with those issued by τ_z up to the l-th hierarchical level, with $l \in [1, L]$, is bounded by Y_z^l, which is recursively defined as follows for $l < L$:*

$$\begin{cases} Y_z^l = Y^{\textit{indirect}}_{\textit{2-level}}(N_z + Y_z^{l+1}, I^{l+1}) + Y_z^{l+1} \\ I^l = \beta(I^{l+1}), \end{cases}$$

and as follows for $l = L$ (base case):

$$\begin{cases} Y_z^L = N_z \times Y^{\textit{direct}}(\tau_z, I_j) \\ I^L = I_j. \end{cases}$$

Proof. The proof is by induction on the hierarchical level $l \in [1, L]$. The proof also shows that the interconnect traversed by τ_z's transactions at the l-th hierarchical level is I^l, which is defined as in the above equations.

Base case: HW-task τ_z is directly connected to I_j at the L-th hierarchical level: hence, $I^L = I_j$ and, at this interconnect, τ_z suffers direct interference only. By Lemma 4, for each of the N_z transactions issued by τ_z, the number of interfering transactions up to the L-th hierarchical level is bounded by $Y^{\text{direct}}(\tau_z, I_j)$.

Inductive case: The induction hypotheses are that Y_z^{l+1} safely bounds the number of transactions that interfere with τ_z up to the $(l+1)$-th hierarchical level and that I^{l+1} is the interconnect traversed by τ_z's transactions at the $(l+1)$-th level. We proceed by showing that Y_z^l is a safe bound for the l-th hierarchical level. First of all, by definition, observe that $I^l = \beta(I^{l+1})$ is the only interconnect traversed by the τ_z's transactions at the l-th hierarchical level. Second, note that the transactions that enter in I^l and that impact on the execution of τ_z must be **(i)** those issued by τ_z itself and **(ii)** those interfering with the ones issued by τ_z at the interconnects traversed by τ_z's transactions at higher hierarchical levels. By the HW-task model, the ones of case (i) are no more than N_z. By the induction hypotheses, the ones of case (ii) are bounded by Y_z^{l+1}. Observe that such requests are coming from I^{l+1} and can incur indirect interference at I^l: by Lemma 3, such an interference can be bounded by $Y^{\text{indirect}}_{\text{2-level}}(N_z + Y_z^{l+1}, I^{l+1})$. Now, it remains to account for all the interference, either direct or indirect, that τ_z's transactions can collect at the higher levels to bound the overall number of interfering requests up to the l-th hierarchical level, Again, by the induction hypotheses, such interference is bounded by Y_z^{l+1}. Hence the lemma follows. □

3.3 Second bound on the number of interfering transactions

We propose here an alternative approach to bound the maximum number of interfering transactions. From the model proposed in Section 2.1, we observe that the HW-tasks are executed periodically. Thus, the number of transactions generated by each HW-task in a given time window is limited and quantifiable.

Lemma 7. *Let τ_i be the HW-task under analysis and let I^l the interconnect traversed by τ_i's transactions at the l-th hierarchical level. In a schedulable[2] system, the number of transactions that can interfere with τ_i up to I^l is bounded by*

$$Y^{time}(\tau_i, I^l) = \sum_{\tau_j \in \Gamma^+(I^l) \setminus \{\tau_i\}} \eta_{i,j},$$
$$where \quad \eta_{i,j} = \left\lceil \frac{T_i + T_j}{T_j} \right\rceil \times N_j$$

Proof. Without loss of generality, consider a periodic instance of τ_i that starts at time 0. Note that, to generate transactions that interfere with those issued by τ_i, a job of another HW-task τ_j must be released no earlier than time $-T_j$. Otherwise, such a job would be already completed when τ_i is released. Similarly, note that a job of τ_j that generates interfering transactions must be released before time T_i, otherwise τ_i would already be completed and no interference would hence be possible. It then follows that the time window of interest to analyze the contention suffered by τ_i and generated by τ_j is $(-T_j, T_i]$. Its length is clearly $T_j + T_i$. Observe that in such a time window τ_j can release at most $\lceil (T_i + T_j)/T_j \rceil$ jobs. Each of them can issue at most N_j transactions. Therefore, there are at most $\lceil (T_i+T_j)/T_j \rceil \times N_j$ transactions that can interfere with τ_i. The total number of transactions that can interfere with τ_i is hence bounded by the sum of such terms computed for each HW-task that can interfere with τ_i. Note that only the HW-tasks whose transactions traverse I^l can interfere at I^l, i.e., those in the set $\Gamma^+(I^l)$. Clearly, τ_i has to be excluded from $\Gamma^+(I^l)$ as it cannot interfere with itself. Hence the lemma follows. □

3.4 Third bound on the number of interfering transactions

This section proposes a third and last bound on the maximum number of interfering transactions. As introduced in Section 2.1, each HW-task can issue a limited amount of outstanding transactions. This means that at any moment of time the number of interfering transactions issued by an interfering HW-task in the network is limited.

Lemma 8. *Let τ_i the HW-task under analysis and let I^l be the interconnect traversed by τ_i's transactions at the l-th hierarchical level. The number of transactions that can interfere with the execution of a job of τ_i up to I^l is bounded by*

$$Y^{outs}(\tau_i, I^l) = N_i \times \sum_{\tau_j \in \Gamma^+(I^l) \setminus \{\tau_i\}} \phi_j$$

Proof. From the model in Section 2.1, each interfering HW-task $\tau_j \in \Gamma^+(I^l)$ can have at most ϕ_j pending transactions in the network at any moment in time. By definition of set $\Gamma^+(I^l)$, the transactions issued by HW-tasks $\tau_j \in \Gamma^+(I^l)$ are those that pass through interconnect I_l. Whenever a transaction issued by τ_i reaches I^l, each HW-task $\tau_j \in \Gamma^+(I^l) \setminus \{\tau_i\}$ can have at most ϕ_j pending transactions, each of which may be propagated before the one of τ_i. Hence $\sum_{\tau_j \in \Gamma^+(I^l) \setminus \{\tau_i\}} \phi_j$ bounds the number of transactions that can interfere with τ_i up to I^l. The lemma follows by recalling that τ_i issues at most N_i transactions per job. □

2. When bounding response times of real-time tasks, it is common to assume that the interfering tasks complete by their deadlines to get rid of circular dependencies that arise in response-time equations. Please refer to [19] (Sec. VI.C) for further details about the validity of this approach.

3.5 Combining the bounds

The following lemma combines the three bounds proposed in Section 3.2, Section 3.3, and Section 3.4 to propose an improved (less pessimistic) bound for the overall maximum number of interfering transactions for an arbitrary HW-task set and interconnect network architecture. The formula proposed in the lemma is iterative – iterating the formula for each interconnect in the path between a HW-task under analysis and until reaching the FPGA-PS interface it is possible to compute the number of interfering transactions suffered by a request under analysis.

Lemma 9. *In a schedulable system, the same claim of Lemma 6 still holds if Y_z^l is recursively defined as follows for $l < L$:*

$$\begin{cases} Y_z^l = \min\Big(Y_{2\text{-}level}^{indirect}(N_z + Y_z^{l+1}, I^{l+1}) + Y_z^{l+1}, Y^{time}(\tau_z, I^l) \\ \qquad\qquad Y^{outs}(\tau_z, I^l)\Big) \\ I^l = \beta(I^{l+1}) \end{cases}$$

and as follows for $l = L$ (base case):

$$\begin{cases} Y_z^L = \min\Big(N_z \times I^{direct}(\tau_z, I_j), Y^{time}(\tau_z, I^L), Y^{outs}(\tau_z, I^L)\Big) \\ I^L = I_j. \end{cases}$$

Proof. The lemma follows as for Lemma 6 after recalling that Lemma 6, Lemma 7, and Lemma 8 provide a safe bound on the number of transactions that can interfere with τ_z. Hence, the minimum of the three bounds is still a safe bound. □

3.6 Delay introduced by an interfering transaction

As explained in Section 2, when building a network, interconnects and HAs are stacked one on top of the other so that the whole network behaves as a pipeline that propagates requests and data. Due to pipelining, the propagation of interfering transactions through each interconnect of the network does not affect the service time of a transaction under analysis, i.e., the contention delay introduced by an interfering transaction is limited to its service time and its data propagation time, as formalized by the following lemma.

Lemma 10. *Let τ_i be the HW-task under analysis connected to interconnect I at the L-th hierarchical level. Supposing that τ_i issues a request for transaction, the worst-case delay contribution of an interfering transaction issued by HW-task τ_j placed at an arbitrary hierarchical level is bounded by:*

$$d^{Pipe,read}(I) = t_{addr} + d_{PS}^{read} + B \cdot t_{data},$$
$$d^{Pipe,write}(I) = t_{addr} + B \cdot t_{data} + d_{PS}^{write} + t_{bresp}$$

for read and write transactions, respectively. This result is independent of the hierarchical level of τ_j.

Proof. A network of AXI interconnects and HAs manages addresses, data, and write responses as a pipeline. As the pipeline is assumed not to get full (see Sec. 2.2), multiple address requests can both be issued and propagated in parallel into the network. Hence, the delay an interfering address

request traversing the corresponding network pipeline (i.e., the one composed by all the stages of the interconnects related to the AR/AW channel) can generate to another request is bounded by the maximum execution time of the pipeline stages. By Section 2.2, this delay is bounded by the hold time t_{addr}. Now, consider read interfering transactions. It remains to bound the delay they generate to τ_i's transactions due to the data in response of address requests. By Section 2.3, data read responses follow the order of address read requests: hence, each interfering transaction can delay τ_i up to $d_{\text{PS}}^{\text{read}}$ time units, waiting for the PS to respond. The data propagation phase of interfering transactions is also subject to pipelining. This means that each of the B words of data can delay a τ_i's transaction by at most the maximum execution time of the data read pipeline stages (R channel). By Section 2.2, this delay is less then t_{data}. Hence, the overall delay is bounded by $B \cdot t_{\text{data}}$. Finally, consider write interfering transactions. By Section 2.3, the write data follow the address requests in order, hence each interfering transaction can delay τ_i up to $d_{\text{PS}}^{\text{write}} + B \cdot t_{\text{data}}$ time units for the same reason mentioned above for read transactions. Write responses are then propagated back through the network of interconnect. Again, analogously to address requests and data, due to pipelining, for each interconnect they can contribute to the delay suffered by each τ_i's transaction by at most the maximum execution time of the pipeline stages for handling write responses (B channel). By Section 2.2, this delay is bounded by the hold time t_{bresp}. The lemma follows. □

3.7 Response-time analysis algorithm

In this section, we present the proposed algorithm bounding the response time of a HW-task connected at an arbitrary hierarchical levels of a generic network of interconnects. Differently from the lemmas already presented (whose bound only the number of interfering transactions), the following algorithm derives the temporal interference assigning a contention cost to interfering transactions. It is worth mentioning that, set a HW-task τ_z under analysis, a safe bound can be derived by computing Y_z^1 from Lemma 9, which is able to bound the total number of interfering transactions across the entire hierarchical network of interconnects (until reaching I_{root}), and then multiplying Y_z^1 by the largest contention cost, that is, the one related to the highest hierarchical level. Nevertheless, a more accurate bound can be derived accounting for level-specific contention cost for each interfering transaction, by identifying the corresponding highest hierarchical level it can interfere with.

Such strategy is implemented by Algorithm 1. As introduced in Section 3, read and write transactions are managed and propagated through separated channels by AXI-based interconnects. Thus, they can be treated separately. As for all of the lemmas presented in this Section, Algorithm 1 holds for both read and write transactions. To avoid duplicating its definition, the algorithm considers a contention cost generated by an interfering transaction $d^{\text{Pipe}}(I_j)$ that has to be replaced with $d^{\text{Pipe,read}}(I_j)$ or $d^{\text{Pipe,write}}(I_j)$ depending on the type of transactions that are studied. Consequently, the algorithm can be used to produce two outputs, respectively one of read and one for write transactions, which to keep a compatible notation are named $d_z^{\text{interf,read}}$ and $d_z^{\text{interf,write}}$.

Input: HW-task $\tau_z \in \Gamma$ directly connected to I_j at level L
Output: d_z^{interf}
$I^L = I_j$
$N^{\text{acc}} \leftarrow 0$
for $l = L, L-1, \ldots, 1$ **do**
  $N^l \leftarrow Y_z^l$ from Lemma 9
  $d_z^{\text{interf}} \leftarrow d_z^{\text{interf}} + (N^l - N^{\text{acc}}) \times d^{\text{Pipe}}(I^l)$
  $I^{l-1} = \beta(I^l)$
  $N^{\text{acc}} \leftarrow N^{\text{acc}} + N^l$
end
return d_z^{interf}

Algorithm 1: The proposed algorithm providing a temporal bound on the maximum contention delay experienced by τ_z and caused by the interfering transactions propagated across the entire hierarchical network of interconnects.

Essentially, the algorithm iterates over all hierarchical levels interested by the HW-task τ_z under analysis, starting from $l = L$ down to $l = 1$, and considers the maximum number of interfering transactions collected up to each interconnect traversed by the transactions issued by τ_z. At each interconnect I^l traversed at the l-th hierarchical level, it is accounted the contention delay of the interfering transactions insisting on I^l that have not already been accounted at a higher hierarchical level.

The presented algorithm allows to finally bound the worst-case response time of each HW-task, which is composed of **(i)** its worst-case execution time, **(ii)** the time spent in performing its transactions (read and write), and **(iii)** its maximum experienced contention delay. Hence, the response time of each HW-task τ_z connected to interconnect I_j is bounded by the following:

$$\mathcal{R}_z = C_z + N_z^R \times d^{\text{NoCont,read}}(I_j) + N_z^W \times d^{\text{NoCont,write}}(I_j) + \\ + d_z^{\text{interf,read}} + d_z^{\text{interf,write}}. \tag{4}$$

Finally, a system is deemed schedulable if all HW-tasks meet their deadlines, i.e., if $\mathcal{R}_z \leq T_z, \forall \tau_z \in \Gamma$.

4 Experimental validation

This section describes an experimental evaluation performed to validate the system model and the effectiveness of the proposed analysis in providing safe timing bounds. All experiments described in this section have been performed on two modern, commercial Xilinx FPGA SoC platforms: the Zynq 7020 (on the PYNQ board) and the Zynq UltraScale+ XCZU9EG (on the ZCU102 board). In the experimental setup, the system DRAM memory is accessed through the high-performance (HP) ports of the FPGA-PS interface on both platforms. This route is used in most real-world designs since it provides the maximum possible throughput to the DRAM memory. Our evaluation showed that both platforms (Zynq 7020 and Zynq UltraScale+ XCZU9EG) present a similar behavior for the scope of this paper. Hence, for the sake of brevity, only the experiments performed on the more recent Zynq UltraScale+ XCZU9EG platform are reported. Finally, this Section concluded by presenting the experimental results obtained in simulation with a synthetic workload.

4.1 Experimental setup

For this experimental evaluation, we developed two custom modules in order to perform accurate, clock-level measurements: **(i)** a traffic generator module, named *greedy HW-task* (also called GHW-task), and **(ii)** a multi-channel *timer* module. The GHW-tasks are used to model any possible bus behavior of HW-tasks in a controllable manner. In this way, GHW-tasks can mimic any transactions pattern issued by real-world HW-tasks to stress bus contention. Each GHW-task can be programmed to generate accurate patterns of transactions compliant with the AXI standard. The transactions can have custom burst lengths and offsets. The multi-channel timer is leveraged to retrieve clock-level accurate measurements of the response times of the GHW-tasks, without interfering with their execution. Both of the modules have been synthesized and implemented leveraging Xilinx Vivado 2020.2. In this evaluation, we keep the FPGA clock set to its default value (100 MHz) in both of the platforms (Zynq 7020 and Zynq UltraScale+ XCZU9EG).

4.2 Platform profiling

This first set of experiments aims at characterizing *propagation delays* and *hold times* introduced in Section 2.2 for the AXI SmartConnect, a state-of-the-art interconnect developed by Xilinx. To this end, we developed a test setup composed of three GHW-tasks connected to the HP0 port of the FPGA-PS interface through an AXI SmartConnect. The test setup also includes an Integrated Logic Analyzer (ILA) [34] used for storing the execution traces of AXI links connecting the GHW-tasks to the AXI interconnect (AXI SmartConnect in this evaluation) and the AXI link connecting the AXI interconnect to the HP0 port. The traces provided by the ILA have been measured to estimate the delays experienced by address and data transactions while traversing the AXI SmartConnect. The propagation delay observed for address read and write transactions is $d_{\text{Int}}^{\text{addr}} = 12$ clock cycles, while the delay observed for read and write data transactions is $d_{\text{Int}}^{\text{data}} = 11$ clock cycles. Finally, the delay observed for write response has been observed is $d_{\text{Int}}^{\text{bresp}} = 9$ clock cycles. The hold times t_{addr}, t_{data}, and t_{bresp} have been observed to be constant and equal to one clock cycle. It is worth noting that such constant delays can be larger in other settings, e.g., when the HW-tasks can delay data sampling (this scenario is not considered in this paper). As introduced in Section 2.3, the delays experienced by read and write data transactions (d_{PS}^{read} and d_{PS}^{write}) while accessing the DRAM memory in the FPGA-PS interface are highly dependent on the DRAM memory controller and its internal policies. As the main focus of this work is on the contention generated at the FPGA subsystem, in the proposed experimentation no memory-intensive workload from the processors was stimulated.[3] Instead, we estimated experimentally that performing a read transaction requires $d_{PS}^{\text{read}} = 50$ clock cycles, while committing a write transaction takes $d_{PS}^{\text{write}} = 40$ clock cycles.

3. The extension of our analysis to consider diverse workloads generated from the processors in PS is left as future work to holistically analyze the PS subsystem.

4.3 Validation of the model

In these experiments, we aim at validating the assumptions proposed in Section 3 for modeling the maximum interference experienced by a HW-task due to transactions issued by interfering HW-tasks. The outcomes will be used as building blocks for assembling and simulating complex hierarchies that are impossible to evaluate on current-generation FPGA SoCs due to excessive resource requirements. To this end, we first consider a flat architecture and then proceed by characterizing the base cases of hierarchical networks.

Interference in a flat network architecture. This experiment evaluates the behavior of the AXI SmartConnect when arbitrating the transactions concurrently issued by multiple GHW-tasks in a flat network. In particular, it aims at estimating the worst-case response time of a transaction, occurring when the transaction loses the entire arbitration cycle of the interconnect. To this end, we configured all of the GHW-tasks for issuing a single request with a burst length of sixteen 4 byte words. The GHW-tasks are activated at the same clock cycle, using a single start signal. The test setup under analysis in these experiments includes four GHW-tasks $\tau_0, \ldots, \tau_3$. Each of the GHW-task is connected to one of the subordinate ports of one AXI interconnect I. The manager port of the interconnect is connected to the HP0 port of the FPGA-PS interface (as shown in Figure 2(a)). In this architecture, all of the transactions issued by the GHW-tasks are subject to a single arbitration cycle of the AXI interconnect to reach the HP0 port. We measured the activation and finishing times of the GHW-tasks using the custom timer module deployed on the FPGA fabric. The maximum response time observed for GHW-tasks, compared with two analytical upper-bounds (Upper Bound A and Upper Bound B) derived from the analysis proposed in Section 3.7 for the flat architecture under analysis, are reported in Figure 3. *Upper-bound A* does not consider the results on pipelining proposed in Lemma 10, so that the delay introduced by each interfering transaction is accounted for as a full transaction cost (i.e., in Algorithm 1, $d^{\text{Pipe}}(I^l) = d^{\text{NoCont}}(I^l)$, see Lemmas 1 and 2. This correspond to the original bound proposed in [25]). Differently, *Upper Bound B* leverages the results of Lemma 10 for setting the terms $d^{\text{Pipe}}(I^l)$ in Algorithm 1, which allow reducing the delay impact of each interfering transaction by leveraging pipelining.

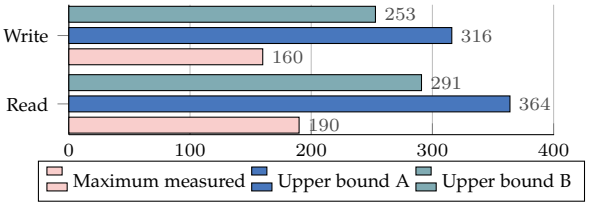

Fig. 3: Maximum measured response times for read and write transactions compared with the two upper bounds proposed in Section 3 (results are reported in clock cycles).

Figure 3 shows that when a HW-task loses an entire arbitration cycle (i.e., in the worst-case scenario), the observed response times are safely bounded by the analysis presented in Section 3. The result provided by Lemma 10 allows reducing the pessimism of the analysis considerably.

Indeed, Upper Bound B provides a 20% improvement for both read and write transactions with respect to Upper Bound A.

Interference in a hierarchical network. This second group of experiments validates the assumptions proposed in Section 3 to characterize the interference caused by the multiple interfering HW-tasks in a hierarchical network of interconnects. To this end, the test setup developed for these experiments comprises four GHW-tasks, τ_0, τ_1, τ_2, and τ_3, and three interconnects, I_0, I_1, I_2, organized as pictured in Figure 4. The GHW-tasks are configured and released as in the previous experiment. In this architecture, the address

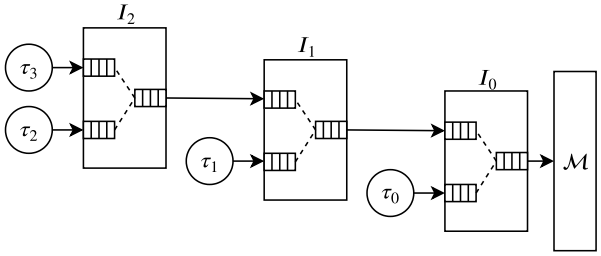

Fig. 4: Hierarchical network architecture considered in the model validation.

requests issued by τ_0 pass a single arbitration phase, which occurs at the interconnect I_0. Differently, the requests issued by τ_1 must traverse two arbitration phases: the first one at I_1, and then at I_0. Finally, the requests issued by the GHW-tasks τ_2 and τ_3 must traverse three interconnect steps, respectively, I_2, I_1, and I_0. The first experiment aims at validating the model for **(i)** the interference caused by interfering HW-tasks connected to the same interconnect (direct interference), and **(ii)** the interference generated by HW-tasks connected to the lower-level interconnects (indirect interference). In this experiment, τ_3 is the HW-task under investigation. It has been configured for issuing a single request for transaction AR_3 (read and write). At the same time, the interfering tasks, τ_2, τ_1, τ_0, have been configured to issue eight consecutive interfering transaction requests of the same type as AR_3. In order to generate maximum contention at the interconnects, τ_1 is released with an offset equal to the interconnect propagation delay $d_{\text{Int}}^{\text{addr}}$. Differently, τ_0 is released with a delay equal to $2d_{\text{Int}}^{\text{addr}}$ (the reported offsets are considered with respect to the release time of AR_3 by τ_3). Similar to previous experiments, an ILA module is used to monitor AXI links between the GHW-tasks and the AXI interconnect, and the single link between the AXI interconnect and the HP0 port. Likewise, we use our custom timer to measure the response times of the GHW-tasks.

Figure 5 reports an ILA trace for read transactions issued by all of the GHW-tasks. It can be observed that all GHW-tasks are simultaneously released at time 15. Then, τ_3 issues its address read request AR_3 (time 16). At the same clock cycle, τ_2 starts issuing its first transaction request, AR_2^0, causing contention at the interconnect I_2. The arbitration round is won by τ_2. Thus, I_2 propagates first AR_2^0 to I_1 and then AR_3. At this level, the interference is compatible with the direct interference described by Lemma 4. After the propagation delay of the interconnect, I_2 issues the requests at the corresponding subordinate port of I_1. At the same clock cycle instant, τ_1 releases its first transaction request, AR_1^0. Thus, another contention happens, and the arbitration round at I_1 is won by τ_1. Consequently, I_1 forwards to I_0 the transaction requests in the following order: $AR_1^0, AR_2^0, AR_1^1, AR_3$, *hence according to round-robin arbitration as assumed in our model*. It is worth noting also that the amount of interfering requests observed on AR_3 at this level is compatible with the one found in Lemma 5 for indirect interference. When I_1 propagates this sequence of requests to I_0, τ_0 starts issuing its transaction requests, hence again causing contention. The arbitration round is won by τ_0. Hence, the transaction requests are issued by I_0 to the HP0 port in the following order: $AR_0^0, AR_1^0, AR_0^1, AR_2^0, AR_0^3, AR_1^1, AR_0^3, AR_3$[4].

Thus, in the worst-case scenario, AR_3 (issued by the GHW-task under analysis) is interfered by seven requests issued by the interfering GHW-tasks, *as by the direct and indirect interference described by the analysis proposed in Section 3*. As the FPGA-PS interface serves the requests in-order, τ_3 receives its corresponding data after all of the interfering requests have been served. At time 274, the first word of data corresponding to AR_3 reaches τ_3. At time 292 the transaction is completed. Figure 5 also confirms that the AXI SmartConnect complies with the model proposed in Section 2.2 and is characterized by $\phi_I = 1$.

Figure 6 reports the maximum measured response times for read and write transactions for the architecture under analysis, compared against the two proposed upper bounds computed using the results of our analysis. As in the previous experiment, Upper Bound A is the original bound proposed in [25] while Upper Bound B accounts for the new results proposed in Lemma 10. Again, such results confirm that the delay is safely bounded by both the Upper Bounds. Nevertheless, Upper Bound B shows even more improved (reduced) pessimism in hierarchical architecture with respect to Upper Bound A for both read and write (improvement around 30%).

Propagation pipelining Figure 5 also shows the effect of pipelining on address requests and data propagation captured by Lemma 10. Given the arbitration policy of the interconnects and the corresponding analytical understanding matured in the above sections, the case reported in the figure leads to the worst-case interference for τ_3, which issues a single transaction AR_3 released at time 16. As explained in the previous example, AR_3 can be interfered at most by seven transactions issued by τ_2, τ_1, and τ_0 at the three crossed levels of interconnects. To reach the FPGA-PS interface, the transactions issued by τ_2 need to traverse three interconnects. The ones issued by τ_1 have to traverse two interconnects, while the transactions issued by τ_0 traverse one interconnect. Accounting for the propagation of AR_3 and the interfering transactions as sequential operations (i.e., without considering the results of Lemma 10) would provide a safe bound for the propagation time equal to $2 \cdot (t_{\text{addr}} + 3 \cdot d_{\text{Int}}^{\text{addr}})$ (i.e., AR_3 and AR_2^0 traversing three interconnects) + $2 \cdot (t_{\text{addr}} + 2 \cdot d_{\text{Int}}^{\text{addr}})$ (i.e., AR_1^0 and AR_1^1

4. We obtained such a result by randomly generating multiple hardware execution tracks and collecting the one of interest in which the round-robin arbitration is lost at any traversed interconnect.

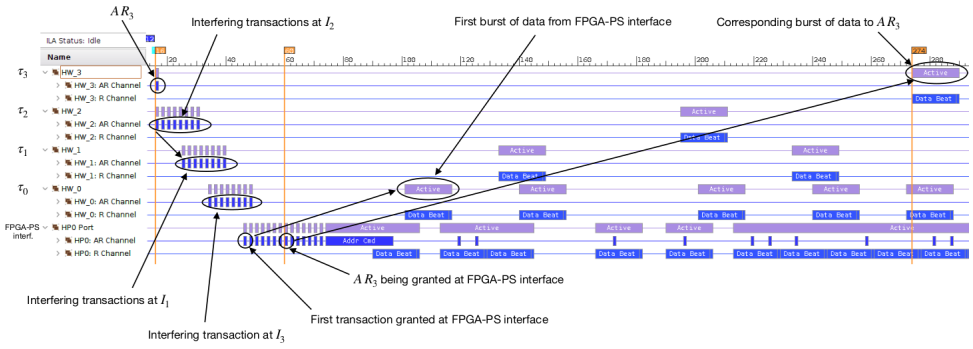

Fig. 5: Waveform track captured using the Integrated Logic Analyzer on a Xilinx Zynq Ultrascale+ platform.

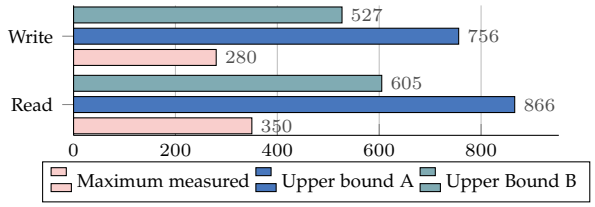

Fig. 6: Maximum measured response times for read and write transactions compared with the upper bounds proposed in Section 3 (in clock cycles).

traversing two interconnects) + $4 \cdot (t_{\text{addr}} + 1 \cdot d_{\text{Int}}^{\text{addr}})$ (i.e., AR_0^0, AR_0^1, AR_0^2, and AR_0^3 traversing one interconnect) = 176 clock cycles. However, note that AR_3 is available to be sampled at the FPGA-PS interface at time 60. Indeed, by the effects of pipelining, its overall propagation time is only 44 clock cycles. This delay is determined by the time required to traverse three levels of interconnects and the hold time. Clearly, considering all such operations as sequential is way too pessimistic. Thanks to the highly predictable behavior of the hardware, Lemma 10 allows bounding the propagation time of the request exactly to 44 clock cycles. This corresponds to an improvement of 75% with respect to the previous bound. The same pipelining effect just described also holds in the propagation of data and write responses.

A considerable amount of the remaining pessimism is ascribable to the coarse-grained model assumed for the DDR controller, whose internal details are mostly not publicly available. However, where a more fine-grained model is provided, the proposed algorithm could exploit it to diminish pessimism and provide a tighter bound.

4.4 Synthetic workload experiments

This section presents an experimental study aimed at evaluating the analysis presented in Section 3 with synthetic workloads. Moreover, this study aims at assessing the impact of the outstanding transactions limit (Lemma 8) and the effects of transaction pipelining (Lemma 10) on the timing feasibility of the system. To this end, this evaluation compares two implementations of the analysis presented in Section 3. The first implementation, named (A), was originally presented in [25] and considers the interference bounds of Sections 3.2 and 3.3 only. The second implementation, named (B), extends the first implementation (A) by incorporating the interference bound based on the number of outstanding transactions (Section 3.4) and the effects of transactions pipelining (Section 3.6). As such, implementation (B) is a refinement of (A).

The AXI system considered in this experiment includes N HW-tasks $(\tau_1, \ldots, \tau_N)$ connected through a binary tree of M interconnects $(I_1, \ldots, I_M)$. The following methodology has been used for generating the task sets: the period T_i and the computation time C_i of the generic HW-task τ_i have been generated using the *fixedrandsum* algorithm [7] ($T_{min} = 10ms <= T_i <= T_{max} = 100ms$, using log-normal distribution). As a reference, the task set utilization has been kept equal to 1. Please note that execution times of the HW-tasks are not relevant for bus contention. The number of transactions issued by the HW-tasks have been generated by first computing the maximum number of transactions that the generic HW-task τ_i can perform in isolation ($N_i^{max} = (T_i - C_i)/\max(d^{NoCont,read}, d^{NoCont,write})$). Following, the total number of transactions $N_i^{R+W} = N_i^R + N_i^W$ is computed by multiplying N_i^{max} with a transaction density factor $\rho \in (0, 1]$ such that $N_i^{R+W} = \rho \cdot N_i^{max}$. The transaction density factor ρ regulates the transaction load that HW-task generates on the system. Finally, the N_i^{R+W} transactions are split in reads and writes using a random uniformly-generated ratio (range $\nu \in [0.4, 0.6]$), such that $N_i^R = \nu \cdot N_i^{R+W}$ and $N_i^W = (1 - \nu) \cdot N_i^{R+W}$. All of the HW-tasks have been configured with $\phi_i = 6$ (a typical value we found from experimental profiling of HAs in the Xilinx IP library) and $B_i = 16$, while all interconnects have $\phi_I = 1$. For the purpose of testing realistic configurations, we assume that each interconnect cannot have more than 16 input ports (as in the case of the Xilinx SmartConnect [35]).

This evaluation considers 16 topological configurations generated by varying combinations of parameters N and M such that $N \in \{4, 8, 16, 24\}$ and $M \in \{1, 2, 4, 8\}$. Configurations where at least one interconnect hosts only a single HW-task are discarded since the interconnect would only increase the latency without performing any arbitration. For each valid configuration (N, M), 100 random values for the bus load factor ρ are uniformly chosen in the range $[0.1, 1.0)$. Then, for each value of bus load ρ, $K = 50000$ synthetic task sets have been generated. Each task set comprises N HW-tasks evenly distributed over M interconnects (i.e., each generic interconnect can connect at most $\lceil N/M \rceil$ HW-tasks). We distributed the HW-tasks on the

network of interconnects in accordance with their slack times $S_i = T_i - C_i$ – the tasks having shorter slacks are collocated closer to the root interconnect. Figure 7 reports the results of such an experimental study. It is worth remembering that each interconnect cannot connect more than 16 tasks – the topologically unfeasible configurations are not considered in the experimentation. The experimental results show that analysis (B) outperforms analysis (A) by a significant margin. The gap between the two approaches becomes larger as the bus load factor ρ increases. These results confirm that the analysis presented in [25] can be significantly improved by considering the effects of transactions pipelining (Lemma 10) and the bound on the number of outstanding transactions (Lemma 8). Moreover, it is still worth noting that increasing the number of interconnects enables connecting more HW-tasks. Also, it can improve the system schedulability ratio by moving HW-tasks showing longer slack time at higher hierarchical levels of the network. Following this methodology, it is possible to reduce the interference on time-constrained HW-tasks (i.e., the ones showing shorter slack times). Nevertheless, it is worth mentioning that moving HW-tasks to higher hierarchical levels increases the worst-case contention experienced by its transactions. Exploring such trade-off requires the investigation of allocation strategies for HW-tasks – we leave this task as future works.

5 Related work

The issue of improving the predictability of response times in SoCs has been deeply addressed, examining different aspects. HW prefetch and arbitration received novel mechanisms and policies to bring improvements from the architectural side [12], [15], [26]. Some authors presented solutions for task allocation that include the memory interface [16], [17].

Enhancing schedulability analysis with the integration of memory interference has been widely investigated, exploiting COTS solutions and proposing ad-hoc ones. Each contribution typically focused on a defined element of the memory tree, such as caches [9], [18], busses [6], [8], and memory controllers [4], [11]. Some authors [5] investigated the effect that memory interference produces on the performance of control applications. The possibility of allotting multiple independent HW-tasks has favorably increased interest in FPGA-based SoCs. However, the access to a shared buffer to perform such allocations implies that these platforms also suffer the effect of memory interference. The de-facto standard in this architecture is the AXI bus [2] whose design focused on performance and flexibility, overlooking time predicability. It allows only the evaluation of the interference exploiting through hardware monitors [29] provided to observe the performance of HW-tasks. A critical aspect for time predictability consists in the decision of excluding several design details from the standard [32] with the implicit assumption that the specific implementations adhere to the guidelines in the standard. Recently, some authors addressed increasing predictability by presenting several novel mechanisms. A bandwidth reservation technique for memory access of HW-tasks has been proposed by Pagani et al. [21]. Restuccia et al. presented several solutions to enforce a fair bandwidth distribution among HW-tasks [25] and avoid that bus transaction suffers an unbounded delay [24]. They also designed a predictable AXI interconnect handled at hypervisor-level [23]. However, the focus of these works is limited to a single interconnect. Thus, they do not address the need for fine-grained timing analysis of bus transactions traversing multiple interconnects.

6 Conclusions

This paper focused on multi-accelerators architectures deployed on FPGA SoC platforms and proposed a detailed model and analysis for interconnects based on the AXI standard. The pessimism of the analysis is reduced by proposing an accurate model capturing the effects of the pipelining in AXI-based interconnects and the features of commercial hardware accelerators. We validated our model and analysis with experimental results involving real designs running on commercial FPGA SoCs from the ZYNQ-7000 and the ZYNQ-Ultrascale+ families from Xilinx. Future work should focus on providing advanced allocation strategies and bus network synthesis tools leveraging the proposed analysis to allocate a given set of hardware accelerators in a bus structure.

References

[1] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a predictable SDRAM memory controller. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256. ACM, 2007.

[2] ARM. *AMBA AXI and ACE Protocol Specification*, 2011.

[3] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo. A framework for supporting real-time applications on dynamic reconfigurable fpgas. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 1–12, 2016.

[4] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo. A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling. In *Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2020)*, 2020.

[5] W. Chang, D. Goswami, S. Chakraborty, L. Ju, C. J. Xue, and S. Andalam. Memory-aware embedded control systems design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(4):586–599, April 2017. https://doi.org/10.1109/TCAD.2016.2613933 doi:10.1109/TCAD.2016.2613933.

[6] Sudipta Chattopadhyay, Lee Kee Chong, Abhik Roychoudhury, Timon Kelter, Peter Marwedel, and Heiko Falk. A unified WCET analysis framework for multicore platforms. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):124, 2014.

[7] Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the synthesis of multiprocessor tasksets. In *proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*, pages 6–11, 2010.

[8] Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Increasing confidence on measurement-based contention bounds for real-time round-robin buses. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2744769.2744858 doi:10.1145/2744769.2744858.

[9] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 245–254. ACM, 2009.

[10] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. A Survey of FPGA-based Neural Network Inference Accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(1):2, 2019.

[11] Mohamed Hassan and Rodolfo Pellizzoni. Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2323–2336, 2018.

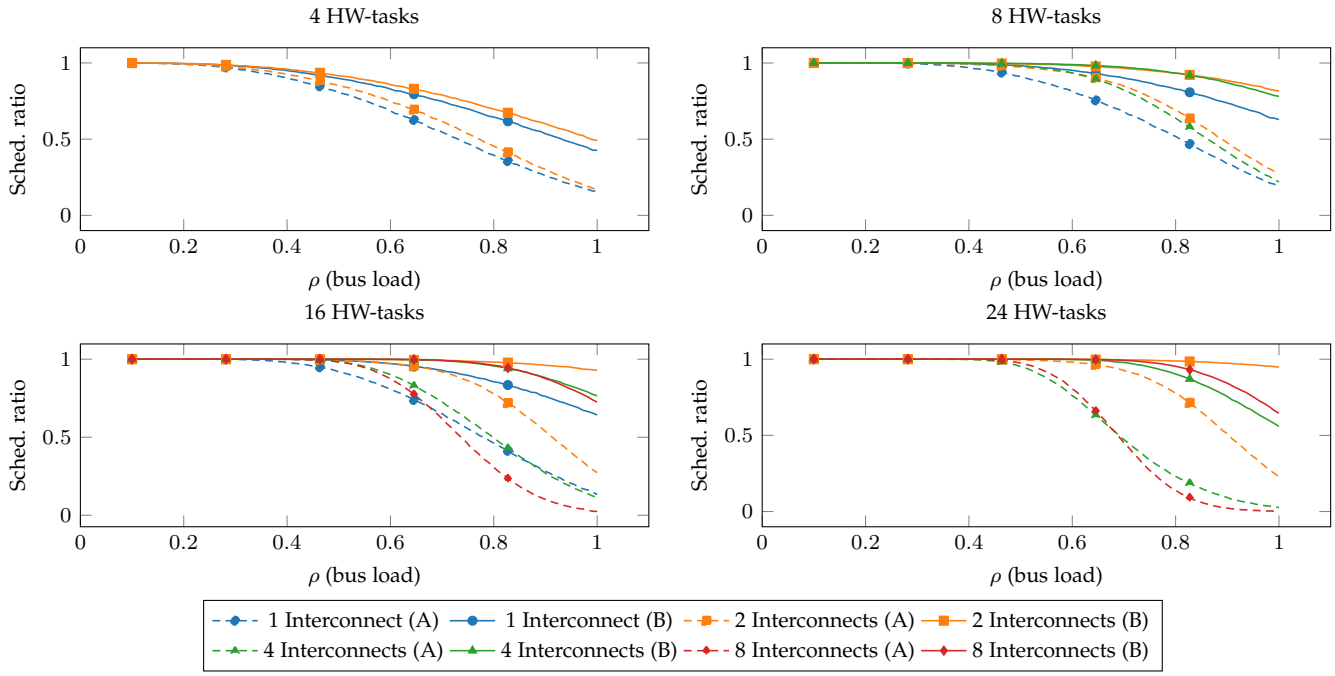

Fig. 7: Experimental results with synthetic workload (each interconnect cannot connect more than 16 tasks).

[12] Farouk Hebbache, Florian Brandner, Mathieu Jan, and Laurent Pautet. Work-conserving dynamic time-division multiplexing for multi-criticality systems. *Real-Time Systems*, 56, 04 2020. https://doi.org/10.1007/s11241-019-09336-w doi:10.1007/s11241-019-09336-w.

[13] Intel. *Stratix 10 GX/SX Device Overview*, 10 2017.

[14] Intel FPGA. *Custom IP Development Using Avalon® and Arm AMBA AXI Interfaces*. OQSYS3000.

[15] J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus designs for time-probabilistic multicore processors. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014. https://doi.org/10.7873/DATE.2014.063 doi:10.7873/DATE.2014.063.

[16] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding memory interference delay in COTS-based multi-core systems. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014.

[17] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding and reducing memory interference in COTS-based multi-core systems. *Real-Time Systems*, 52(3):356–395, May 2016.

[18] Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi. A survey on static cache analysis for real-time systems. *Leibniz Transactions on Embedded Systems*, 3(1):05–1–05:48, 2016. URL: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v003-i001-a005, https://doi.org/10.4230/LITES-v003-i001-a005 doi:10.4230/LITES-v003-i001-a005.

[19] Geoffrey Nelissen and Alessandro Biondi. The SRP Resource Sharing Protocol for Self-Suspending Tasks. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 361–372. IEEE, 2018.

[20] Marco Pagani, Alessio Balsini, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. A linux-based support for developing real-time applications on heterogeneous platforms with dynamic fpga reconfiguration. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 96–101. IEEE, 2017.

[21] Marco Pagani, Enrico Rossi, Alessandro Biondi, Mauro Marinoni, Giuseppe Lipari, and Giorgio Buttazzo. A Bandwidth Reservation Mechanism for AXI-Based Hardware Accelerators on FPGAs. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:24, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[22] Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, and Giorgio 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.

[23] Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, Giorgiomaria Cicero, and Giorgio Buttazzo. AXI HyperConnect: A Predictable, Hypervisor-level AXI Interconnect for Hardware Accelerators in FPGA SoC. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC 2020)*, 2020.

[24] Francesco Restuccia and Ryan Kastner. Cut and forward: Safe and secure communication for fpga system on chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022. https://doi.org/10.1109/TCAD.2022.3197343 doi:10.1109/TCAD.2022.3197343.

[25] Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Modeling and analysis of bus contention for hardware accelerators in fpga socs. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[26] M. Slijepcevic, C. Hernandez, J. Abella, and F. J. Cazorla. Design and implementation of a fair credit-based bandwidth sharing scheme for buses. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 926–929, March 2017. https://doi.org/10.23919/DATE.2017.7927122 doi:10.23919/DATE.2017.7927122.

[27] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[28] Xilinx. *Zynq-7000 All Programmable SoC - Reference Manual*, 9 2016. UG585.

[29] Xilinx. *AXI Performance Monitor v5.0*, 2017. PG037.

[30] Xilinx. *Vivado Design Suite: AXI Reference Guide*, 7 2017. UG1037.

[31] Xilinx. *Zynq UltraScale+ Device - Reference Manual*, 12 2017. UG1085.

[32] Xilinx. *AXI Interconnect, LogiCORE IP Product Guide*, 2018. PG059.

[33] Xilinx Inc. *The CHaiDNN official github website*. https://github.com/Xilinx/chaidnn.

[34] Xilinx Inc. *Integrated Logic Analyzer, LogiCORE IP Product Guide*, 2016. PG172.

[35] Xilinx Inc. *SmartConnect, LogiCORE IP Product Guide*, 2018. PG247.

Francesco Restuccia is a postdoctoral researcher at the University of California, San Diego. He received his Ph.D. in Computer Engineering (cum laude) from Scuola Superiore Sant'Anna Pisa in 2021. His main research interests include hardware security, on-chip communications, timing analysis for heterogeneous platforms, cyber-physical systems, and time predictable hardware acceleration of deep neural networks on commercial FPGA SoC platforms.

Marco Pagani is a postdoctoral researcher at the Real-Time Systems (ReTiS) Laboratory of Scuola Superiore Sant'Anna. In 2016, he received an M.Sc. in Embedded Computing Systems at the University of Pisa and Scuola Superiore Sant'Anna. In 2020, he received a Ph.D. in computer engineering in a cotutelle program between Scuola Superiore Sant'Anna and Université de Lille, under the supervision of Prof. Giorgio Buttazzo and Prof. Giuseppe Lipari. His main research interests include predictable hardware acceleration on heterogeneous platforms and system-level software for real-time systems.

Alessandro Biondi is associate professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. He graduated (cum laude) in Computer Engineering at the University of Pisa, Italy, within the excellence program, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and safe and secure machine learning. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

Mauro Marinoni received his M.S. in Computer Engineering at the University of Pavia (Italy) in 2003, where he also obtained his Ph.D. in Computer Engineering in 2007. He is working since 2007 at the Real-Time Systems Laboratory (ReTiS), where he has been an Assistant Professor from 2009 to 2020. He has been local coordinator of the FP7 JUNIPER project, the Eurostars RETINA project, and several industrial projects exploiting the ReTiS Lab research outcomes in different application fields, from e-Health devices to autonomous and distributed systems.

Giorgio Buttazzo is full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa, received a M.S. degree in Computer Science at the University of Pennsylvania, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He is Editor-in-Chief of Real-Time Systems, Associate Editor of the ACM Transactions on Cyber-Physical Systems, and IEEE fellow since 2012. He has authored 7 books on real-time systems and more than 300 papers in the field of real-time systems, robotics, and neural networks, receiving 13 best paper awards.