Modeling and Analysis of Bus Contention for Hardware Accelerators in FPGA SoCs

Francesco Restuccia
TeCIP Insitute and Dept. of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy
francesco.restuccia@santannapisa.it

Marco Pagani
TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
Université de Lille, CNRS, Centrale Lille, UMR 9189, CRIStAL, Lille, France
marco.pagani@santannapisa.it

Alessandro Biondi
TeCIP Insitute and Dept. of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy
alessandro.biondi@santannapisa.it

Mauro Marinoni
TeCIP Insitute and Dept. of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy
mauro.marinoni@santannapisa.it

Giorgio Buttazzo
TeCIP Insitute and Dept. of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy
giorgio.buttazzo@santannapisa.it

Abstract

FPGA System-on-Chips (SoCs) are heterogeneous platforms that combine general-purpose processors with a field-programmable gate array (FPGA) fabric. The FPGA fabric is composed of a programmable logic in which hardware accelerators can be deployed to accelerate the execution of specific functionality. The main source of unpredictability when bounding the execution times of hardware accelerators pertains the access to the shared memories via the on-chip bus. This work is focused on bounding the worst-case bus contention experienced by the hardware accelerators deployed in the FPGA fabric. To this end, this work considers the AMBA AXI bus, which is the de-facto standard communication interface used in most the commercial off-the-shelf (COTS) FPGA SoCs, and presents an analysis technique to bound the response times of hardware accelerators implemented on such platforms. A fine-grained modeling of the AXI bus and AXI interconnects is first provided. Then, contention delays are studied under hierarchical bus infrastructures with arbitrary depths. Experimental results are finally presented to validate the proposed model with execution traces on two modern FPGA-based SoC produced by Xilinx (Zynq-7000 and Zynq-Ultrascale+ families) and to assess the performance of the proposed analysis.

2012 ACM Subject Classification Hardware → Interconnect; Hardware → Hardware accelerators

Keywords and phrases Heterogeneous computing, Predictable hardware acceleration, FPGA SoCs, Multi-Master architectures

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2020.12

Supplementary Material ECRTS 2020 Artifact Evaluation approved artifact available at `https://doi.org/10.4230/DARTS.6.1.4`.

1 Introduction

Next-generation *cyber-physical systems* (CPS) require the execution of complex computing workload such as machine learning algorithms and image/video processing. Representative examples include autonomous driving, advanced robotics, and smart manufacturing. In order to perform high-performance computations while matching the timing constraints imposed by

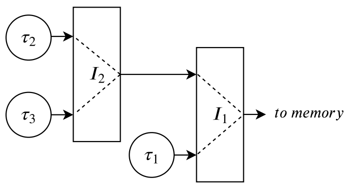

Figure 1 Example bus architecture with three HAs connected using two interconnects.

the physical world, these systems require coupling standard processing units with on-board *hardware accelerators* (HAs), which allow speeding up complex computations, especially those that are prone to large-scale parallelization. Heterogeneous computing platforms, such as system-on-chips (SoC) that integrate a multiprocessor with acceleration-oriented devices like field-programmable gate arrays (FPGAs) or general-purpose graphical processing units (GPGPUs) are de-facto establishing as the reference solutions to develop next-generation CPS. Examples of such platforms are the Zynq Ultrascale+ produced by Xilinx, which includes a large FPGA fabric, and the Xavier produced by Nvidia, which includes a GPGPU and other accelerators for machine learning algorithms.

A key issue when developing safety-critical CPS is to guarantee certain *timing constraints* for the control software. When hardware acceleration is used by critical software, the problem also extends to the consideration of the worst-case timing properties of HAs. Unfortunately, timing analysis for HAs can be particularly challenging, especially if very limited information on their internal architecture and resource management logic is publicly available, as it is the case for Nvidia platforms. To further complicate this issue, note that HAs are typically very memory-intensive. Indeed, they tend to work on a large amount of data (think of real-time video processing) and hence generate a consistent memory traffic that can have a paramount impact on their timing performance, especially when running together with other accelerators that cause contention at some stage on their path towards shared memories (such as buses and memory controllers).

FPGA-based heterogeneous platforms represent very promising solutions to cope with these issues. As a matter of fact, they allow deploying energy-efficient, yet powerful HAs on the FPGA fabric that have a very regular clock-level behavior [3, 21]. FPGA-based HAs are typically implemented as state machines and issue a fairly predictable pattern of bus transactions. As such, the execution times of HAs when running in isolation are characterized by extremely limited fluctuations, and are hence very predictable. The major phenomenon that harms the timing predictability of FPGA-based HAs that are statically programmed on the FPGA and access a shared memory, is the corresponding memory contention they can experience on the bus or at the memory controller.

Nevertheless, differently from other platforms, FPGAs expose a fine-grained control of the bus infrastructure to designers, which are free to organize the bus hierarchy at their own choice in order to match timing constraints, as well as to deploy custom arbitration modules to dispatch memory transactions towards the memory controller [1]. At last, designers are even free to deploy custom on-chip memories on the FPGA fabric for which they can have full control on how contention is regulated [18]. Such strategies can be used to achieve a higher degree of predictability for the memory traffic.

Focusing on most common approaches, FPGA designs for hardware acceleration in COTS SoCs typically consist of a set of accelerators that act as masters on the bus to access the main DRAM memory (off-chip) shared with the multiprocessor(s), e.g., see [27] [10] [33]. Being the number of ports to access the shared memory limited, the typical solution consists in multiplexing multiple masters on the same port by means of *interconnects*, which are usually available in the standard library of devices offered by FPGA vendors. Interconnects can also be hierarchically connected to form a hierarchical bus network: an example of such networks comprising three HAs (τ_1, τ_2, and τ_3) and two interconnects (I_1 and I_2) is depicted in Figure 1. Clearly, the topology of the bus hierarchy has a primary impact on how the access to memory is regulated, and hence on the corresponding delays due to memory contention. For instance, assuming that both I_1 and I_2 in Figure 1 implement round-robin arbitration with the granularity of one memory transaction per HA per round-robin cycle, it is possible to note that τ_1 is privileged in accessing the memory. Indeed, once every two round-robin cycles I_1 could grant one memory transaction issued by τ_1, while in the other cycle transactions from τ_2 and τ_3 are alternated. At a very high level, τ_1 have a privileged access to the memory controller.

It is crucial to note that, in FPGA SoC, the operating frequency of the FPGA fabric is much lower than the on-chip memory controller (which is realized in hard silicon, i.e., placed outside the FPGA) and the memory itself. For instance, in a Zynq-7000 by Xilinx, the default operating frequency of the FPGA is 100MHz, while the Processing System, which includes the memory controller, runs at 650MHz. As such, the delays introduced by a bus infrastructure realized on the FPGA by means of interconnects are typically of the same order of magnitude of the ones required to access the memory, and hence do not consist of a negligible contribution to the response times of HAs.

Contribution. This paper studies bus contention and proposes a worst-case response time analysis for HAs deployed on FPGA-based SoCs. The AXI open bus standard [2] is considered because of the following reasons: **(i)** AXI is the de-facto standard communication interface for COTS FPGA SoC platforms [31] [13], **(ii)** AXI is widely supported by well-established FPGA design tools such as Xilinx Vivado [30] and Intel Quartus Prime [14], **(iii)** many commercial (closed-source) HAs use AXI interfaces. To begin, a fine-grained model for the AXI bus and AXI interconnects is presented (Sec. 3). The model accounts for several kinds of delays experienced by bus transactions and the behavior of commercial interconnects. Then, the paper presents a response-time analysis to bound the worst-case response time of recurrent HAs that access a shared memory via an arbitrary *hierarchical* network of interconnects (Sec. 4). Finally, three experimental evaluations (Sec. 5) are reported. First, a set of experimental results obtained from a state-of-the-art FPGA SoC by Xilinx are presented to validate the model proposed in this paper. Second, a case study executed on the same platform is discussed by matching measurements extracted from its execution with the bounds provided by the proposed analysis. Third, experimental results obtained with synthetic workload are presented.

2 Essential Background

A typical FPGA SoC architecture combines a *Processing System* (PS), which includes one or more processors, with a FPGA subsystem in a single device. Both subsystems access a shared DRAM controller through which they can access a DRAM memory. Figure 2 illustrates a typical SoC FPGA architecture in which two interfaces allow the communication between the FPGA subsystem and the processing system (PS). The de-facto standard interface for interconnections is the ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI) [2].

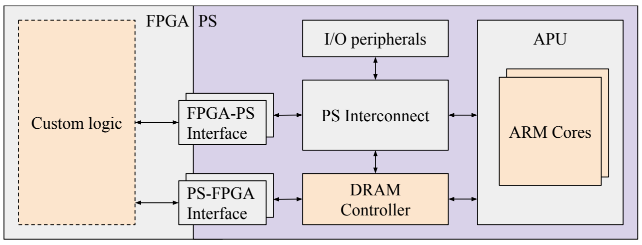

Figure 2 Simplified architecture of a SoC FPGA platform.

The AXI bus. The AMBA AXI standard defines a master-slave interface allowing simultaneous, bi-directional data exchange. An AXI *interface* (also referred to as port) is composed of five independent *channels*: Address Read (AR channel), Address Write (AW channel), Data Read (R channel), Data Write (W channel), and Write Response (B channel). This paper considers that data are transmitted back to the master on the R channel (for read data) or provided to the W channel (for write data) in the same order with which the corresponding requests have been routed to the address channel. In other words, address requests are served in-order, that is, *the access to the output data channels* R *and* W *depends on the order in which requests are routed to the address channels.* Even though this assumption does not directly derive from the standard, it is a popular design choice reported in the documentation of many commercial devices such as those produced by Xilinx [28, 31].

The AXI standard allows masters to issue multiple pending requests. This means that, in principle, each master is allowed to issue an unlimited number of outstanding transactions (typically limited by the designers of devices connected to the bus). AXI offers two methods for transmitting data between masters and slaves: single transactions or transaction *bursts*. When operating in burst mode, the requesting device can issue a single address request to fetch/write up to 256 data words per request.

AXI ports. As it is illustrated in Figure 2, 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. The first one offers a set of slave interfaces 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 set of slave interfaces to the PS and is used by devices deployed on the FPGA (e.g., hardware accelerators) to access the central DRAM memory or the on-chip memory in the PS. Being the number of available ports in the FPGA-PS interface limited, scenarios in which a port is contended by multiple master devices deployed on the FPGA are common in realistic designs. To cope with the case in which bus contention is maximized, this paper is focused on the arbitration required to solve conflicts of requests that target the same output port. Nevertheless, the results of this work can also be easily extended to scenarios in which multiple ports are used.

AXI interconnects. Whenever multiple AXI masters want to access the same output port, an AXI *interconnect* is in charge of arbitrating conflicting requests to the same port. The access to each channel of the output AXI port is managed by a multiplexer. Each multiplexer is controlled by an arbiter that decides, at each time, which slave channel is granted to the master channel. The arbiters are completely independent from each other. Each port (slave or master) of the AXI interconnect is buffered with a FIFO queue (which is typically quite large). For instance, in FPGA SoCs by Xilinx, two implementations of the interconnect are available: *AXI Interconnect* (deprecated in the latest platforms) and *AXI SmartConnect*. Both the implementations are multiplexer-based and therefore comply with the specification described above.

Arbitration policy. In this work, each arbiter is assumed to implement a round-robin policy. To the best of our records, round-robin is the most common solution in commercial off-the-shelf platforms. For instance, the AXI arbiters for FPGA SoCs by Xilinx implement round-robin (both the AXI interconnect and the AXI SmartConnect, see [35], p.6 and [32], p.7). Note that fixed-priority arbitration has been discontinued in the AXI SmartConnect. Furthermore, even though the AXI standard defines QoS signals to regulate the quality-of-service of transactions, these signals are ignored by state-of-the-art interconnects (see [35], p. 8 and [32], p. 9).

Hierarchical interconnection. State-of-the-art interconnects dispose of a limited amount of slave ports. However, AXI interconnects can even be connected between each other, creating a network tree of interconnects with multiple hierarchical levels. In such a structure, each inner node of the tree represents an interconnect, each leaf represents a master device, and the root node represents the sole interconnect connected to the slave port of the FPGA-PS interface (i.e., the sink of all the traffic towards the FPGA-PS interface). Thanks to such hierarchical structures, it is possible to connect as many devices as desired to a single AXI port of the FPGA-PS interface (provided that there is enough area on the FPGA to deploy all the modules). Clearly, the address requests (both read and write) and the data issued by a device connected at some interconnect I in a hierarchical network must traverse all the interconnects encountered on the path from I to the FPGA-PS. In a dual manner, write responses and the data read by the same device must traverse the same path in reverse order, i.e., from the FPGA-PS interface to I. Note that, due to the intrinsic parallelism of the AXI bus, and the fact that each interconnect is an independent engine that executes in parallel with the others, a network of interconnects exhibits a pipelined behavior.

Read transactions. A general read transaction issued by a master device τ starts with the issue of the address request R_{addr} on the AR channel of its master port M_τ, which is sampled by the corresponding slave port of the AXI interconnect to which τ is directly connected. R_{addr} is then routed through a network of one or multiple AXI interconnects until reaching the FPGA-PS interface (and then the memory controller). After a service delay related to the logic in the Processing System, the memory controller, and the DRAM memory, the requested data R_{data} become available on the R channel of the FPGA-PS interface. Hence, data are routed back to τ through the same interconnect network traversed by R_{addr}, but in reverse order. Once available at M_τ, data R_{data} are sampled by τ, hence completing the read transaction.

Write transactions. A general write transaction issued by a master device τ starts with the issue of the address request W_{addr} on the AW channel of its master port M_τ, which is sampled by the corresponding slave port of the AXI interconnect I to which τ is directly connected. W_{addr} is then routed through a network of one or multiple AXI interconnects until reaching the FPGA-PS interface (and eventually the memory controller). In parallel, once W_{addr} is granted by I, the corresponding data W_{data} are provided by τ to the W channel of its master port and flow through the path reaching the FPGA-PS interface following W_{addr} (i.e., reaching the Processing System and then the memory controller). After a service delay (introduced by the PS, the memory controller, and the DRAM memory), the Processing System provides a write response W_{resp} on the B channel of the FPGA-PS interface to acknowledge τ. W_{resp} is routed from the FPGA-PS interface through the same network of interconnects traversed by W_{addr} and W_{data}, but in reverse order. Once available at M_{τ}, W_{resp} is sampled by τ and the write transaction is completed.

3 System model

This section focuses on modeling the components of a system comprising a set of AXI-based hardware accelerators, deployed on the FPGA fabric of a FPGA-SoC platform and connected to a shared DRAM memory on the Processing System through the FPGA-PS interface.

3.1 Hardware task model

Each hardware accelerator implements a specific functionality; therefore, from now on, they are referred to as *hardware tasks* (HW-tasks for short). Each HW-task includes an AXI memory-mapped master interface through which it can autonomously load and store data from the DRAM memory. Each HW-task τ_i is periodically executed every T_i clock cycles, hence generating an infinite sequence of periodic instances referred to as *jobs*. Each job of τ_i **(i)** issues at most N_i^R read transactions and N_i^W write transactions, both with a fixed burst size B; **(ii)** issues at most ϕ_i outstanding transactions per type, i.e., it can have at most ϕ_i pending read transactions and ϕ_i pending write transactions at any time; **(iii)** computes for at most C_i clock cycles; and **(iv)** has a relative deadline equal to T_i (each job must complete before the release of the next one). Read transactions and write transactions are supposed to be independent. Furthermore, note that read and write transactions are routed through independent AXI channels, that is, they do not influence each other when the corresponding data is transmitted.

It is important to observe that no specific memory access pattern for the HW-tasks is assumed, i.e., the requests for memory transactions can be arbitrarily distributed over time across the jobs. This assumption makes the results presented in this paper more general and robust with respect to the HW-tasks' behavior. However, at the same time, it limits the number of timing properties related to bus pipelining that can be used at the stage of analysis as, in the worst-case, transactions can be sufficiently spread far apart such that HW-tasks do not fully exploit pipelining.

3.2 AXI interconnect model

The system can comprise several interconnects connected in a hierarchical fashion. Each interconnect I_j has S_j slave ports and one master port. As each interconnect has a single master port, the incoming traffic (at the master port and) directed to the slave ports does not experience any conflict. On the other hand, address requests of the same type (read or write) issued by different HW-tasks can experience conflicts, which are managed by independent, per-channel, arbiters (see Section 2). The granularity of the round-robin arbiters is ϕ_I, i.e., at each round-robin cycle the master port grants at most ϕ_I read requests (resp., write requests) to each HW-task. To ease the notation in the analysis presented in Section 4, it is assumed that all the interconnects in the system share the same parameter ϕ_I (the analysis can be easily extended to the case of different per-interconnect round-robin granularities). Finally, it is assumed that the FIFO queues associated with the ports of the interconnects are large enough to never saturate during the execution[1]. Each interconnect introduces a propagation delay in address and data propagation. Specifically, we denote by $d_{\text{Int}}^{\text{addr}}$ the latency introduced in the propagation of address requests, by $d_{\text{Int}}^{\text{data}}$ the latency introduced in the propagation of a word of data (read or write), and by $d_{\text{Int}}^{\text{bresp}}$ the latency introduced in the propagation of a write response. These propagation delays can be derived from the specifications in the official documentation of the considered interconnect (when available) or by employing experimental profiling. The AXI standard defines hold times as the numbers of clock cycles that the address or data must be kept on the corresponding AXI channel while both *valid* and *ready* signals are asserted. Address and data hold times are modeled with the following terms: t_{addr} denotes the hold time of an address request, t_{data} denotes the hold time of a word of data, and t_{bresp} denotes the hold time of a write response.

3.3 Processing System and Memory Controller model

The DRAM memory controller is a global system resource shared among all HW-tasks. Being part of the Processing System, it is accessed from the FPGA fabric through the FPGA-PS interface. Each port of the FPGA-PS interface can be configured to map a contiguous range of addresses, which is referred to as a memory region. As typical for hardware acceleration, it is assumed that each HW-task loads and stores data from a private memory buffer. To address the case in which the maximum contention is experienced, we focus on the case in which all the memory buffers are allocated in the same memory region and accessed through a single AXI port at the FPGA-PS interference. Note that the results of this work can also be extended to the case in which the HW-tasks access the DRAM memory via multiple ports at the FPGA-PS interface: this case is left as future work due to lack of space.

The DRAM memory controller included in the Processing System can be conceptually divided into two main blocks: **(i)** the AXI interface block and **(ii)** the DDR physical core block. The AXI interface block is in charge of receiving and arbitrating the incoming AXI transactions from the AXI slave ports, while the DDR physical core schedules and issues the corresponding read and write requests to the controller's physical layer, which eventually drives the DRAM memory by generating control and data signals.

Typically, the internal architecture of the DDR physical core includes multi-level queues structures, managed with dedicated scheduling policies that reorder transactions to maximize throughput and efficiency [11]. On many commercial platforms, the internals of the DDR physical core block, including the scheduling policies and the queues structure, are not publicly disclosed or are not well documented. For this reason, a fine-grained modeling of the DDR physical core block goes beyond the scope of this paper and it is not addressed here. Rather, being our focus on the conflicts at the interconnects, a coarse-grained modeling of the DRAM-related delays is adopted here: if the internals of the DDR controller are known, then our results can be refined (e.g., by adopting the results from [11]).

From the perspective of the FPGA-PS interface, address requests directed to the DDR memory controller are served *in order* (see [28], p. 297, and [31], p. 440). This means that the order of the data read responses on the data read channel follows the order of the address read requests granted at the address read channel. In the same way, write address requests are served and acknowledged in order. These properties are guaranteed by the

[1] Note that ensuring this condition is an orthogonal problem to timing analysis. That is, it is an a-priori requirement that can be verified independently of the timing performance of the system.

DRAM Memory Controller AXI Interface block. Note that this feature is independent of the internal scheduling policies of the DDR Physical core block, which may include internal reordering, hence affecting the worst-case service time of a request.

Following these considerations, this work assumes that the Processing System and the memory controller introduce the following (cumulative) delays:

- $d_{\mathrm{PS}}^{\mathrm{read}}$ 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
- $d_{\mathrm{PS}}^{\mathrm{write}}$ 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.

Note that, by definition, these delays include the propagation times introduced by the internal logic of the Processing System and the overall service time at the memory controller. These parameters depend on the internals of the Processing System and can be quantified using the documentation provided by the SoC producer (when available) or through experimental profiling (and over-provisioning).

3.4 Overall architecture

Formally, the system is composed of a set $\Gamma = \{\tau_1, \dots, \tau_n\}$ of n HW-tasks, a set $\mathcal{H} = \{I_1, \dots, I_s\}$ of s AXI interconnects, and a memory controller $\mathcal{M}$ included in the Processing System. The HW-tasks in Γ are interconnected through a network of the AXI Interconnects in the set $\mathcal{H}$ that is organized as follows. Each slave port of the AXI interconnects can be directly connected to the master port of a HW-task or, in a hierarchical manner, to the master port of another interconnect. The set of HW-tasks directly connected to interconnect I_j is denoted by $\Gamma(I_j)$. Similarly, the set of interconnects directly connected to the slave ports of I_j (i.e., in input) is denoted by $\mathcal{H}(I_j)$. Furthermore, the set of HW-tasks whose transactions traverse I_j is denoted by $\Gamma^+(I_j)$, i.e., those that are directly or transitively connected to I_j. The interconnect at the bottom of this hierarchy has its master port directly connected to the slave port of the FPGA-PS interface (i.e., to reach $\mathcal{M}$). All the transactions issued by the HW-tasks must pass through this latter interconnect, which is referred to as the *root* interconnect I_{root}. Note that, as interconnects have a single master port, the master port of each interconnect $I_j \neq I_{\mathrm{root}}$ is connected to a slave port of exactly one interconnect, which is denoted by $\beta(I_j)$. For consistency, $\beta(I_{\mathrm{root}}) = \emptyset$. The topology of the whole system resembles a tree where I_{root} is the root node, the HW-tasks in Γ are the leaves, and the interconnects in $\mathcal{H} \setminus \{I_{\mathrm{root}}\}$ are the intermediate nodes (see Figure 3(b)). An interconnect I is said to be placed at the *hierarchical level* L_I if a HW-task directly connected to I has to traverse L_I interconnects before reaching the FPGA-PS interface (I_{root} is at first level, i.e., $L_{I_{\mathrm{root}}} = 1$). The main symbols used in the paper are summarized in Table 1.

4 Response-time analysis

This section proposes an analysis to bound the response times of HW-tasks connected to an arbitrary hierarchical network of interconnects as presented in the previous section.

The analysis is structured in incremental lemmas. First, Section 4.1 proposes a bound on the worst-case response time for a single transaction assuming no contention at the interconnects. Both read and write transactions are considered. Subsequently, Section 4.2 and Section 4.3 propose two different methods to bound the number of interfering transactions that affect a job of a HW-task under analysis. These two bounds are then combined in

Table 1 Main symbols used throughout the paper.

N_i	Number of transactions issued by τ_i (can have superscript R or W)
ϕ_i	Maximum number of outstanding transactions for τ_i
ϕ_I	Max. number of trans. granted per round-robin cycle by interconnects
B	Burst size of a transaction
t_{addr}	Hold time for a single address request on the bus
t_{data}	Hold time for a single data word on the bus
t_{bresp}	Hold time for a single write response on the bus
$d_{\text{PS}}^{\text{read}}$	Max. latency introduced by the PS on a read transaction
$d_{\text{PS}}^{\text{write}}$	Max. latency introduced by the PS on a write transaction
$d_{\text{Int}}^{\text{data}}$	Propagation latency of data word through an interconnect
$d_{\text{Int}}^{\text{addr}}$	Propagation latency of address request through an interconnect
$d_{\text{Int}}^{\text{bresp}}$	Propagation latency of write response through an interconnect
$\Gamma(I_i)$	Set of the HW-task directly connected to I_j
$\mathcal{H}(I_j)$	Set of the interconnects directly connected to slave ports of I_j
$\beta(I_j)$	Interconnect connected to the master port of I_j
$\Gamma^+(I_j)$	Set of HW-tasks whose transactions pass through I_j

Section 4.4. Finally, Section 4.5 presents an iterative algorithm that uses the results of the previous sections to bound the maximum response time of a HW-task of interest.

As the AXI standard defines the same methods to handle both read and write address requests, the bounds derived in this section hold for both read and write transactions. For this reason, in order to keep a compact notation, this section uses just the symbol N_i instead of N_i^R or N_i^W to denote the number of transactions issued by τ_i.

4.1 No contention at the interconnects

This first lemma establishes an upper bound on the response time of a single memory transaction issued by an arbitrary HW-task under analysis τ_i, connected to an interconnect I placed at an arbitrary hierarchical level L, assuming no bus contention from the other HW-tasks in the system[2].

Remember that AXI manages read and write transactions on independent channels: as such, they are separately considered by the following two lemmas.

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

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

Proof. Following Section 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,

[2] It is worth noting that this contention-free bound does not properly correspond to the case in which the transaction under analysis is served in isolation, but rather just to the case in which no contention is experienced at the interconnects. This is because, for the reasons discussed in Section 3.3, the delay related to the Processing System and the memory controller already cope with conditions of maximum contention.

R_{addr} goes through the interconnect network tree, traversing the remaining $L-1$ interconnects. As for I_j, each of them introduces a latency bounded by $t_{\text{addr}} + d_{\text{Int}}^{\text{addr}}$. Therefore, R_{addr} is available at the master port of the root interconnect I_{root} after an overall propagation delay of $L \cdot (t_{\text{addr}} + d_{\text{Int}}^{\text{addr}})$, where it is sampled from the slave port of the FPGA-PS interface. The Processing System 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 3.3). At this point, the data words R_{data} corresponding to R traverse the L levels of the interconnect tree, in reverse order with respect to R_{addr}, until reaching τ_i. Since data words are sequentially propagated on the interconnect tree, the propagation latency in the data phase is paid just once on all the burst of data due to pipelining. Hence, considering that t_{data} is the data time (for each word) and that $d_{\text{Int}}^{\text{data}}$ is the latency introduced by each interconnect on data words, the overall latency paid to propagate the data burst on the interconnect tree is $L \cdot (t_{\text{data}} + d_{\text{Int}}^{\text{data}})$. The lemma following by summing up these contributions. ◀

▶ **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*

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

Proof. The write transaction W begins with the issue of the address write request W_{addr} by τ_i, which lasts t_{addr} time units . Following 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 corresponding (parallel) channels, until reaching the FPGA-PS interface. Data can be propagated only after the corresponding address; hence, the latency experienced by W_{addr} and W_{data} to traverse 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 $W_{\text{dat}a}$ is given by $L \cdot (t_{\text{addr}} + \max\{d_{\text{Int}}^{\text{addr}}, d_{\text{Int}}^{\text{data}}\})$, which must be summed to the time to transmit the data themselves, i.e., $B \cdot t_{\text{data}}$. At this point, the Processing System routes W_{addr} and W_{data} to the memory controller. Following Section 3.3, 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 $L \cdot (t_{\text{bresp}} + d_{\text{Int}}^{\text{bresp}})$. The lemma follows by summing up these contributions. ◀

It is worth noting that the bounds provided by the two above lemmas just depend on the hierarchical level L (identified by the interconnect I_j) at which a HW-task is directly connected.

4.2 First bound on the number of interfering transactions

We proceed in an incremental manner by starting from the simple case in which contention at a single interconnect is considered, say I_{root} for simplicity (see Figure 3(a)). The following lemma establishes a bound on the number of interfering transactions (issued by other HW-tasks) 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 master port of I_{root} after at most*

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

transactions.

Proof. As mentioned in Section 3, the interconnects implement a round robin arbitration to solve conflicts on address requests issued by different HW-tasks. In the worst-case scenario, τ_i is the last HW-task served in the round robin arbitration cycle, i.e., after all the other HW-tasks in $\Gamma(I_{\text{root}})$. From the model in Section 3.1, each HW-task τ_j can have at most ϕ_j pending transactions. On the other hand, from the model in Section 3.2, the maximum number of transactions granted to each HW-task for each round robin cycle by interconnects is equal to ϕ_I. For these reasons, I_{root} grants at most $\min(\phi_j, \phi_I)$ for each interfering HW-task $\tau_j \in \Gamma \setminus \{\tau_i\}$ per round-robin cycle. The lemma follows by summing up the contribution of each interfering HW-tasks. ◀

With the above lemma in place, we can proceed to bound the number of interfering requests under a general hierarchical network of interconnects. We note that a HW-task τ_i can incur two kinds of interference: **(i)** *direct interference*, which is the one that τ_i's transactions experience at the interconnect to which τ_i is directly connected to; and **(ii)** *indirect interference*, which is the one that τ_i's transactions, or other transactions that generate direct interference to τ_i, experience in other interconnects at shallower hierarchical levels on their way towards the FPGA-PS interface. Further details on both kinds of interference are provided in the following.

Direct interference. The same reasoning used for Lemma 3 can be extended when considering a hierarchical network of interconnects such as the one illustrated in Figure 3(b). Note that, in such a case, a HW-task can also experience contention at an interconnect due to transactions coming from other interconnects placed at higher hierarchical levels. For instance, in Figure 3(b), τ_i (directly connected to I_{root}) can incur in a contention due to a transaction issued by τ_z, which is directly connected to I_1.

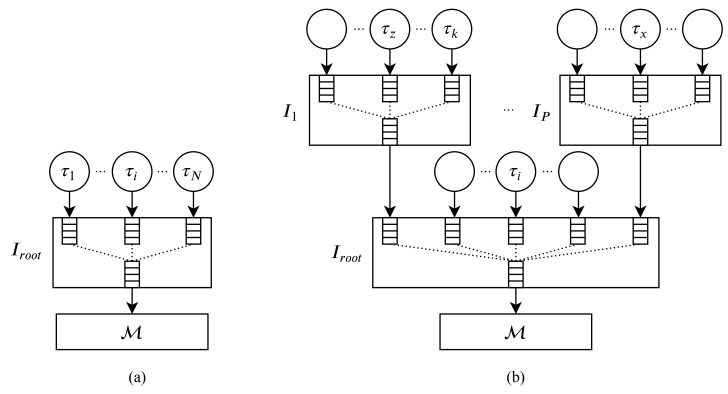

Figure 3 **(a)** A set of HW-tasks directly connected to I_{root}. **(b)** Example hierarchical network of interconnects and HW-tasks with two hierarchical levels. Each circle corresponds to a HW-task (only the ones mentioned in the text are assigned a name).

▶ **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 grants the access to the master port of I_j after at most*

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

transactions.

Proof. Following the model of Section 3.2, at each round-robin cycle I_j grants at most ϕ_I transactions per its slave port. Clearly, this consideration is true also when an interconnect I_h, placed at a higher hierarchical level, is connected to a slave port of I_j. Hence, 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, i.e., independently of the number of HW-tasks or interconnects connected to I_h. Hence, the interconnects directly connected to I_j interfere with at most $|\mathcal{H}(I_j)| \times \phi_I$ transactions. The first term of Eq. (2) follows due to the same considerations done for Lemma 3. Hence the lemma follows. ◀

Indirect interference. While propagated through a series of interconnects on their way towards the FPGA-PS interface, the transactions issued by the HW-task under analysis can also incur contention at shallower hierarchical levels. For instance, the transactions issued by τ_z in Figure 3(b) can incur contention at I_{root}, e.g., due to other transactions issued by τ_i or τ_x. Furthermore, note that indirect interference can also affect other transactions that generate direct interference to the HW-task under analysis, hence leading to a *transitive* interference phenomenon. For instance, still considering Figure 3(b), a transaction issued by τ_k that delays 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 this case, we say that a transaction of τ_x transitively delays τ_z.

In the following, a set of lemmas are presented to account for indirect interference. We proceed in an incremental manner by starting from the consideration of just two *adjacent* hierarchical levels.

▶ **Lemma 5.** *Consider an arbitrary interconnect I_j at hierarchical level $L \geq 2$ that issues Δ transactions in output to its master port. In the worst-case scenario, the Δ transactions can be indirectly interfered by*

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

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

Proof. Consider one of the Δ transactions, say r. As addressed by Lemma 4, r can incur direct interference at the (only) interconnect $\beta(I_j)$ directly connected to I_j at hierarchical level $L-1$. As such, the interference at $\beta(I_j)$ can be bounded as for Lemma 4. The only differences here are that **(i)** as r comes from another interconnect I_j, it means that it has not been originated by a HW-task connected to $\beta(I_j)$ and hence no HW-task has to be excluded from those that generate interfering transactions (first term in the sum of Eq. (2)); and **(ii)** I_j has to be excluded from the interconnects that generate interfering transactions as it is the one from which the interfered transaction is coming from (second term in the sum Eq. (2)). Note that the second term in the multiplication of Eq. (3) serves this purpose. The lemma follows by accounting for this bound for each of the Δ transactions. ◀

With the above lemma in place, it is possible to generalize the bound of indirect interference to an arbitrary hierarchical structure with $L > 2$ 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 $l < L$:*

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

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 $l \in [1, L]$. We also show that I^l is the interconnect traversed by τ_z's transactions at the l-th hierarchical level. **Base case:** At hierarchical level L, τ_z is directly connected to I_j: hence, $I^L = I_j$ and τ_z suffers direct interference only. Therefore, the number of interfering transactions up to the L-th hierarchical level is bounded by accounting for the bound provided by Lemma 4 for each of the N_z transactions issued by τ_z. **Inductive case:** We proceed by assuming that Y_z^{l+1} yields a safe bound for the number of interfering transactions up to the $(l+1)$-th hierarchical level and that I^{l+1} is the interconnect traversed by τ_z's transactions at the same level. Now, we show that Y_z^l provides a safe bound for the l-th hierarchical level. First, note that by definition, $I^l = \beta(I^{l+1})$ denotes the (only) interconnect across which τ_z's transactions can pass at the l-th hierarchical level. Second, observe that the transactions that are received in input by I^l and that affect τ_z's execution are **(i)** those issued by τ_z itself and **(ii)** those that generated interference to τ_z at the interconnects traversed at higher hierarchical levels. The former are no more than N_z (by the model), while the latter are Y_z^{l+1} (by inductive assumption). Such requests are coming from I^{l+1} and can incur indirect interference at I^l, which can be bounded by Lemma 3 as $Y_{\text{2-level}}^{\text{indirect}}(N_z + Y_z^{l+1}, I^{l+1})$. To bound the overall number of interfering requests up to the l-th hierarchical level, it then remains to account for all the (direct and indirect) interference collected at the higher levels, which is given by Y_z^{l+1} (by inductive assumption). Hence the lemma follows. ◀

Thanks to the above lemma, the total number of transactions that interfere with τ_z (under analysis) *across the entire hierarchical network of interconnects* can be bounded by looking at the interference collected up to the root interconnect, i.e., Y_z^1.

4.3 Second bound on the number of interfering transactions

A different approach can be used to derive an alternative bound on the number of interfering transactions by leveraging the observation that the HW-tasks are periodically executed, and hence can only generate a limited number of transactions in a given time window.

▶ **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 system, the number of transactions that can interfere with τ_i up to I^l is bounded by*

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

Proof. Consider HW-task τ_i and assume all HW-tasks never execute after their deadlines[3]. Without loss of generality, suppose that a period instance of τ_i begins at time 0. To interfere with τ_i, a job of another HW-task τ_j must be released after time $-T_j$, otherwise, it would be completed when τ_i is released. In the same way, an interfering job of τ_j must be released before time T_i, otherwise τ_i would already be completed and hence no contention can be generated. As a result, the time window of interest to study the contention generated by τ_j to τ_i is $(-T_j, T_i]$ with length $T_j + T_i$. In this time window there can be at most $\lceil (T_i + T_j)/T_j \rceil$ jobs of τ_j. As each job of τ_j can issue at most N_j transactions, there are at most $\lceil (T_i + T_j)/T_j \rceil \times N_j$ transactions that can interfere with τ_i. The number of interfering transactions is hence bounded by the sum of such contributions from 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: according to the system model, the set of such tasks is $\Gamma^+(I^l)$. Clearly, τ_i has to be excluded from $\Gamma^+(I^l)$ as it cannot interfere with itself. Hence the lemma follows. ◀

4.4 Combining the two bounds

This lemma combines the bounds proposed in Section 4.2 and Section 4.3 to introduce a less pessimistic bound on the overall number of interfering transactions for an arbitrary interconnect architecture tree and HW-task set. The proposed formula is iterative on the interconnect levels. Iterating the formula for each interconnect in the path, from the interconnect to which the HW-task under analysis is directly connected to I_{root}, it is possible to find the overall number of interfering transactions a request under analysis issued by the HW-task under analysis suffers.

▶ **Lemma 8.** *In a schedulable system, the same claim of Lemma 6 still holds if Y_z^l is recursively defined as follows for $l < L$:*

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

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

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

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

4.5 Response-time analysis algorithm

Leveraging the results of the previous sections, this section presents an algorithm to bound the worst-case response time of HW-tasks connected at arbitrary hierarchical levels. While the lemmas presented in the previous sections allow bounding the *number* of interfering transactions, this section is concerned with assigning a contention cost to them in order to obtain the corresponding temporal interference.

[3] Assuming a schedulable system to bound response times is a typical approach when circular dependencies are present in the response-time equations. The interested reader is invited to refer to [20] (Sec. VI.C) for an explanation about why this is a sound approach to bound response times.

To begin, note that the contention cost associated to each interfering transaction is not constant: indeed, following the model of Section 3, transactions experience a propagation delay each time they traverse an interconnect. Hence, a transaction that interferes at a high hierarchical level generates more delay than another one that interferes at a shallower hierarchical level.

Clearly, given a HW-task τ_z under analysis, a safe bound can be obtained by first computing Y_z^1 from Lemma 8, which provides a bound on the number of interfering transactions across the whole hierarchical network of interconnects (i.e., up to I_{root}), and then multiplying Y_z^1 by the largest contention cost, i.e., the one related to the highest hierarchical level. However, a more accurate bound can be devised if a level-specific contention cost is accounted for each transaction by detecting the highest hierarchical level at which it can interfere.

Algorithm 1 Bounding the worst-case contention delay experienced by τ_z due to interfering transactions across the whole hierarchical network of interconnects.

```
Input: HW-task $\tau_z \in \Gamma$ directly connected to $I_j$ at level $L$
Output: $d_z^{\text{interf}}$
$d_z^{\text{interf}} \leftarrow 0$
$I^L = I_j$
$N^{\text{acc}} \leftarrow 0$
for $l = L, L-1, \dots, 1$ do
    $N^l \leftarrow Y_z^l$ from Lemma 8
    $d_z^{\text{interf}} \leftarrow d_z^{\text{interf}} + (N^l - N^{\text{acc}}) \times d^{\text{NoCont}}(I^l)$
    $I^{l-1} = \beta(I^l)$
    $N^{\text{acc}} \leftarrow N^l$
end
return $d_z^{\text{interf}}$
```

This strategy is implemented by Algorithm 1. As mentioned at the beginning of Section 4, read and write transactions are independently managed by AXI and hence can be separately treated. For this reason, analogously as for the lemmas presented above, Algorithm 1 holds for both read and write transactions. To avoid duplicating its definition, the algorithm considers a contention cost $d^{\text{NoCont}}(I_j)$ that has to be replaced with $d^{\text{NoCont,read}}(I_j)$ or $d^{\text{NoCont,write}}(I_j)$ depending on the type of transactions that are studied. Consequently, the algorithm can be used to produce two outputs, which to keep a compatible notation are named $d_z^{\text{interf,read}}$ and $d_z^{\text{interf,write}}$. In essence, the algorithm iterates over all hierarchical levels interested by the HW-task τ_z under analysis (from $l = L$ to $l = 1$) and copes with the number of interfering transactions collected up to each interconnect traversed by τ_z transactions. For each interconnect I^l traversed at the l-th hierarchical level, the algorithm accounts for the contention delay of the interfering transactions that insist on I^l but have not been accounted at a higher hierarchical level. As said before, this is because the contention cost $d^{\text{NoCont}}(I_j)$ is monotone with the hierarchical level (the higher the level the larger the cost).

Thanks to this algorithm, it is finally possible to bound the worst-case response time of each HW-task, which is given by **(i)** its worst-case execution time, **(ii)** the time required to perform its read and write transactions, and **(iii)** the contention delay experienced by the latter. Hence, for each HW-task τ_z connected to interconnect I_j it is bounded by

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

A system is then schedulable if all HW-tasks meet their deadlines, i.e., if $\mathcal{R}_z \le T_z, \forall \tau_z \in \Gamma$.

5 Experimental results

This section first presents an experimental evaluation that has been conducted to validate the system model and assess the performance of the proposed analysis (Section 5.3). The experiments have been performed on two state-of-the-art Xilinx SoC FPGA platforms, namely the Zynq-7020 and the ZCU102 Zynq Ultrascale+. On both platforms, one of the high-performance (HP) ports of the Processing System is used in the FPGA-PS interface. Due to the lack of space, this section reports only the results of the experiments performed on Zynq Ultrascale+, since the Zynq-7000 exhibit comparable behavior. Finally, Section 5.4 reports other experimental results obtained with the synthetic workload.

5.1 Experimental setup

In order to perform a clock-level accurate evaluation, two custom IPs have been developed: a programmable traffic generator IP, named *greedy* HW-task (GHW-task for short), and a multichannel *timer* IP. The purpose of the GHW-task IP is to generate, in a controllable way, cycle-accurate patterns of transactions compliant with the AXI standard with arbitrary offsets, spacing, burst size, and maximum number of outstanding transactions. GHW-tasks have been developed to cope with any possible bus behavior of HW-tasks, i.e., they can mimic any kind of pattern of bus transactions issued by real-world, memory-intensive HW-tasks, and are hence useful to stress bus contention. On the other hand, the multichannel *timer* IP is used to perform clock-level accurate measurements of the GHW-tasks' response times without perturbating their execution. Both IPs have been synthesized and implemented using Xilinx Vivado 2018.2. The FPGA clock is set to the default value (100 MHz), while the Processing System runs at the default clock speed of 1.2 GHz.

5.2 Profiling

This set of experiments aims at characterizing the propagation *delay* and the *hold* times, introduced in Section 3.2, for the AXI SmartConnect. To this end, a test setup composed of three GHW-tasks connected to the HP0 port in the FPGA-PS interface through an AXI SmartConnect has been realized. An Integrated Logic Analyzer (ILA) [34] module has been placed to monitor the AXI links that connect each GHW-task to the AXI SmartConnect and the AXI link that connects the AXI SmartConnect to the HP0 port. From the waveform track provided by the ILA, we measured the delays experienced by addresses and data while traversing the AXI SmartConnect (respectively, $d_{\text{Int}}^{\text{addr}}$ and $d_{\text{Int}}^{\text{data}}$, see Section 3.2) and the hold times t_{addr}, t_{data}, and t_{bresp} introduced in Section 3.2. The propagation delays (in clock cycles) observed on both hardware platforms are $d_{\text{Int}}^{\text{addr}} = 12$, $d_{\text{Int}}^{\text{data}} = 11$, and $d_{\text{Int}}^{\text{bresp}} = 9$, while the hold times t_{addr}, t_{data}, and t_{bresp} have been observed to be all constant and equal to one clock cycle. We note that these constant delays may be larger in different settings (not considered in this work) in which HW-tasks are not always ready to sample data or write responses, or when the FIFO queues of the interconnect or the FPGA-PS interface saturate. As mentioned in Section 3.3, the cumulative delays d_{PS}^{read} and d_{PS}^{write} in accessing the DRAM memory from the FPGA-PS interface depend on several aspects and on the masters that insist on the memory controller. In our experiment we did not used memory-intensive workload executed on the processors and we experimentally estimated these delays as $d_{PS}^{\text{read}} = 50$ clock cycles and $d_{PS}^{\text{write}} = 40$ clock cycles.

5.3 Model validation

This experiment aims at validating the assumptions made in Section 4 to characterize the interference that a HW-task may suffer from other HW-tasks. We distinguish between the case of a flat network and the one of a hierarchical network.

Interference in a flat network. The test setup used for these experiments comprises four GHW-tasks $\tau_0, \ldots, \tau_3$ directly connected to a single interconnect I, which is in turn directly connected to the HP0 port exported by the FPGA-PS interface (e.g., likewise as in Fig. 3). The GHW-tasks' activation and finishing times are measured using one of the custom timer IP deployed on the fabric. In this experiment, all the GHW-tasks are simultaneously activated (at the same clock cycle) by the Processing System using a single shared logic signal generated by an AXI GPIO module. With this experimental setup, all transactions issued by the GHW-tasks are subject to a single arbitration step performed by the AXI SmartConnect. The purpose of this evaluation is to experimentally evaluate the behavior of the AXI SmartConnect in the condition of contention. Furthermore, this experiment aims at experimentally measuring the maximum response time of a transaction in the worst-case scenario, i.e., when it loses an entire arbitration cycle, and comparing it with the proposed upper bound on the response time proposed in Section 4.5. To this end, all the GHW-tasks have been programmed to issue a single read (or write) request corresponding to a burst of sixteen 32-bit words. The experiment has been repeated for both read and write transactions. Figure 4 reports the maximum response time measured among all the GHW-tasks, compared with the upper-bound proposed for the flat architecture considered in this experiment.

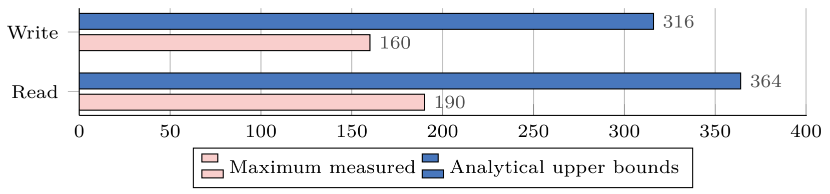

Figure 4 Maximum measured response times for read and write transactions compared with the upper bound proposed in Section 4 (in clock cycles).

The results reported in Figure 4 confirm that in the worst-case scenario stimulated here, i.e., when a HW-task loses an entire arbitration cycle, the measured response times can be safely bounded by the upper bound proposed in Section 4.

Interference in a hierarchical network. This set of experiments aims at validating the assumptions made in Section 4 to characterize the interference that a HW-task may suffer in a hierarchical network of Interconnects, due to the interfering HW-tasks in the system. The test setup used for this set of experiments comprises four GHW-tasks, $\tau_0, \ldots, \tau_3$, and three interconnects, I_0, I_1, I_2, organized as shown in Figure 5.

In this configuration, the transaction requests issued by τ_0 pass through a single step of arbitration occurring at interconnect I_0, while the requests issued by τ_1 traverse two arbitration steps occurring at I_1 and then I_0. Finally, the transaction requests issued by τ_2 and τ_3 pass through three arbitration steps at I_2, I_1, and I_0. The GHW-tasks are programmed and released as for the previous experiment. The first subset of experiments aims at validating the direct interference that a HW-task may suffer due to other HW-tasks connected to the same interconnect and the indirect interference coming from HW-tasks

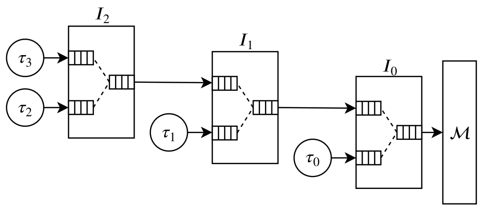

Figure 5 Reference architecture for the model validation in hierarchical network.

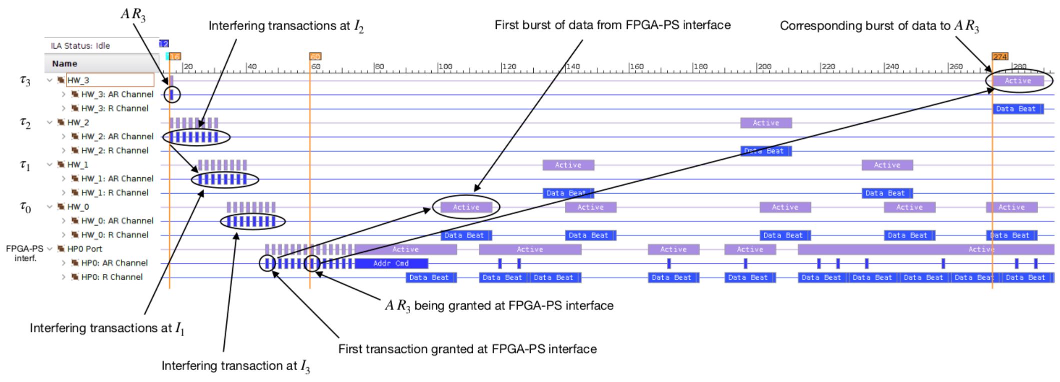

Figure 6 Sample waveform track from the Integrated Logic Analyzer on Zynq Ultrascale+.

connected to the lower-level Interconnects. In this experiment, τ_3 is the HW-task under investigation. τ_3 is programmed to issue a single request for transaction AR_3 (read or write) while the interfering tasks, τ_2, τ_1, τ_0, are programmed to issue eight consecutive interfering requests for transactions of the same type of AR_3. In order to stimulate contention at the interconnects, τ_1 is released with an offset equal to the interconnect propagation delay $d^{\text{addr}}_{\text{Int}}$, while τ_0 is released with a delay equal to $2d^{\text{addr}}_{\text{Int}}$ (offsets are with respect to the release time of AR_3 by τ_3). Each GHW-task-to-SmartConnect AXI link and the SmartConnect-to-HP0 AXI link are monitored by an ILA module deployed on the fabric and the HW-task's response times are measured using the timer module.

Figure 6 reports the ILA waveform track for read transactions acquired on the Zynq-7020 SoC using Vivado 2018.2. At time 15, all GHW-tasks are simultaneously released. Soon afterwards, at time 16, τ_3 issues its address request AR_3. At the same time, τ_2 starts issuing its first transaction request, say AR^0_2, causing a contention at the interconnect I_2. The arbitration round is won by τ_2. Hence I_2 first propagates AR^0_2 to I_1 and then AR_3. The interference at this level is compatible with the direct interference described in Lemma 4. After the propagation delay of the interconnect, I_2 issues the requests at the corresponding slave port of I_1. At the same instant, τ_1 releases its first transaction request, AR^0_1. Hence 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: $AR^0_1, AR^0_2, AR^1_1, AR_3$, *hence according to round-robin arbitration as assumed in our model.* Note also that the amount of interfering requests 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 causing contention. τ_0 wins the arbitration round, hence the transaction requests are issued by I_0 to the HP0 port in the following order: $AR^0_0, AR^0_1, AR^1_0, AR^0_2, AR^3_0, AR^1_1, AR^3_0, AR_3$. Therefore, in the worst case, the request AR_3

issued by the GHW-task under investigation is interfered by seven requests coming from interfering GHW-tasks, *as considered by direct and indirect interference phenomena captured by our analysis in Section 4*. Since HP0 serves the incoming requests in order, τ_3 receives its data response only after all the interfering requests have been served with data. At time 274, the first word of data corresponding to AR_3 reaches τ_3 and at time 292 the transaction is completed. It is worth observing that Figure 6 also confirms that the AXI SmartConnect is compatible with the model introduced in Section 3.2 and that is characterized by $\phi_I = 1$.

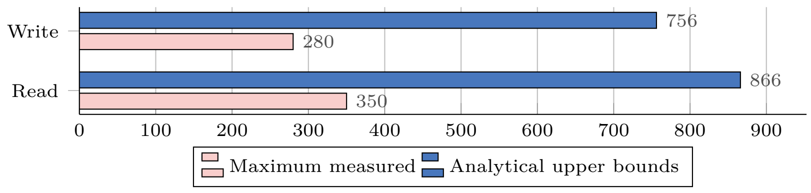

Figure 7 Maximum measured response times for read and write transactions compared with the upper bound proposed in Section 4 (in clock cycles).

Figure 7 compares the maximum measured response times for read/write transactions with the upper bounds computed by our analysis for the architecture under evaluation in this experiment. Also in this case, the results confirm that the delay incurred by transactions can be safely bounded.

5.4 Experiments with synthetic workload

This experimental study has been carried out to evaluate the analysis presented in Section 4 with synthetic workloads. We considered systems with N HW-tasks $(\tau_1, \dots, \tau_N)$ connected over a binary tree of M interconnects $(I_1, \dots, I_M)$. Task sets have been generated as follows. First, the period T_i and computation time C_i of each HW-task τ_i have been generated using the *fixedrandsum* algorithm [7] (T_i between $T_{min} = 10ms$ and $T_{max} = 100ms$, using log-normal distribution) by keeping the task set utilization equal to 1 as a reference value (note that the tasks' execution times are not relevant for bus contention in this context). Second, the number of transactions have been generated by first computing the maximum number of transactions that each HW-task τ_i can perform in isolation, as $N_i^{max} = (T_i - C_i)/\max(d^{NoCont,read}, d^{NoCont,write})$. Then, 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 $N_i^{R+W} = \rho \cdot N_i^{max}$. Finally, the N_i^{R+W} transactions are split between reads and writes using a random uniformly-generated ratio in the 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 HW-tasks have been configured with $\phi_i = 6$ (we found it being a typical value from experimental profiling of HAs in the Xilinx IP library) and $B_i = 16$, while all interconnects have $\phi_I = 1$. In order to test realistic configurations, it has been assumed that each interconnect cannot have more than 16 input ports (as it is the case for the Xilinx SmartConnect [35]).

The study considers 16 possible configurations generated by testing combinations of parameters N and M such that $N \in \{4, 8, 16, 24\}$ and $M \in \{1, 2, 4, 8\}$. Unuseful configuration, in which at least one interconnect hosts just a single HW-task, are discarded. This because, in such configurations, that Interconnect(s) would not perform any arbitration, adding only additional latency. For each valid configuration (N, M), 100 random values for ρ are uniformly chosen in the range $[0.1, 1.0)$. Then, for each value of ρ, $K = 50000$ synthetic task sets have been generated, each comprising N HW-tasks evenly distributed over M

interconnects (i.e., each interconnect is directly connected to at most $\lceil N/M \rceil$ HW-tasks). The HW-tasks have been distributed over the interconnect tree according to their slack times $S_i = T_i - C_i$, i.e., tasks with shorter slack times are placed closer to the root interconnect.

Figure 8 reports the results of the experimental study. Please note that, since each interconnect cannot be connected to more than 16 tasks, some configurations are topologically unfeasible. Hence, they are not considered and the corresponding data is not reported. The experimental results show that increasing the number of interconnects not only allows to connect a larger number of HW-tasks, but also can improve the system schedulability ratio by moving HW-tasks with larger slack time to interconnects at higher hierarchical levels, thus reducing their interference on more time-constrained HW-tasks (i.e., HW-tasks with shorter slack times). However, moving HW-tasks to a higher hierarchical level also increases the latency and the worst-case contention experienced by its transactions. The exploration of this trade off requires investigating on allocation strategies for HW-tasks, which is left as future work.

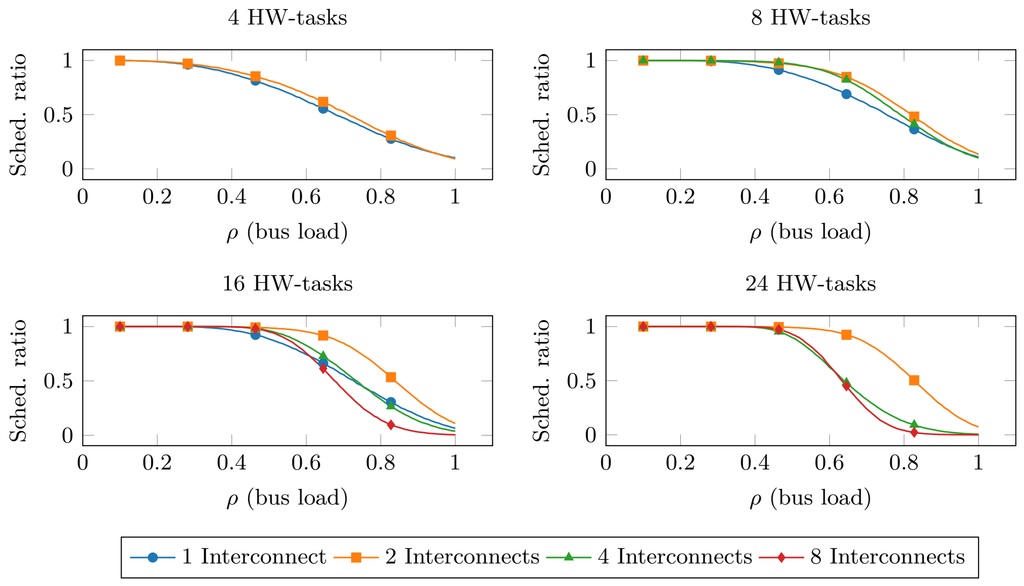

Figure 8 Experimental results with synthetic workload.

6 Related work

Considerable efforts have been spent in bounding and controlling response times in SoCs by addressing the problem from several prospectives. From an architectural point of view, several mechanisms and policies have been proposed, as support for HW prefetch and new arbitration policies [12, 15, 26]. Other works proposed to consider memory inteference in the context of task allocation [16, 17]. Significant work has been dedicated to the integration of memory interference in the schedulability analysis of both COTS and ad-hoc solutions, with a focus on specific elements in the memory tree, like the contribution of caches [9, 19], busses [6, 8], and the memory controller [4, 11]. Also, the explicit effect on the performance of control applications has been investigated [5]. Recently, FPGA-based SoCs have received particular interest, but allocating multiple HW-tasks inside the FPGA requires the use of a shared bus to access the off-chip memory. The AXI bus [2] is the de-facto standard but has been designed considering flexibility and performance, not time predictability. In fact, the evaluation of bus interference is achieved with hardware monitors in charge of observing HW-tasks performance [29]. Moreover, the standard entrusts several design details to the single implementation and assumes all components behave accordingly [32]. Some mechanisms have been proposed to increase predictability. For example, Pagani et al. [22] proposed an approach to apply bandwidth reservation techniques to HW-tasks memory accesses, while Restuccia et al. designed a mechanism to guarantee fairness among HW-tasks transactions [25], a mechanism to prevent unbounded delays during bus transactions [23], and proposed a predictable, Hypervisor-level AXI interconnect for FPGA SoC [24]. However, these contributions only address single interconnects and are not concerned with a fine-grained timing analysis of bus transactions.

7 Conclusion and future work

This work focused on FPGA-based SoC and presented a fine-grained model for the AXI bus and AXI interconnects. An analysis has been proposed to bound the contention delays experienced by HW-tasks under hierarchical networks of interconnects that allow reaching the FPGA-PS interface (and hence shared memories connected to the Processing System). The model and the effectiveness of the analysis have been validated with experimental results on two modern FPGA SoC by Xilinx. Future work should focus on deriving a more accurate model and analysis of the AXI bus to capture pipelining effects, and on allocation strategies and bus network synthesis for a given set of HAs.

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. `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. `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.

12 F. Hebbache, M. Jan, F. Brandner, and L. Pautet. Shedding the shackles of time-division multiplexing. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 456–468, December 2018. `doi:10.1109/RTSS.2018.00059`.

13 Intel. *Stratix 10 GX/SX Device Overview*, October 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. `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 Jörg Henkel Lars Bauer, Marvin Damschen. Runtime-reconfigurable architectures for WCET guarantees and mixed criticality. In *Special session at ESWEEK 2019: Analyses and Architectures for Mixed-Critical Systems: Industry Trends and Research Perspective*. ACM, 2019.

19 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. `doi:10.4230/LITES-v003-i001-a005`.

20 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.

21 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.

22 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.

23 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.

24 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.

25 Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs. *ACM Trans. Embedded Computing Systems*, 18(5s):51:1–51:22, October 2019.

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. `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*, September 2016. UG585.

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

30 Xilinx. *Vivado Design Suite: AXI Reference Guide*, July 2017. UG1037.

31 Xilinx. *Zynq UltraScale+ Device - Reference Manual*, December 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.