Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs

FRANCESCO RESTUCCIA, Scuola Superiore Sant'Anna
MARCO PAGANI, Scuola Superiore Sant'Anna and CRIStAL (Univ. Lille, CNRS, Centrale Lille, UMR 9189)
ALESSANDRO BIONDI, Scuola Superiore Sant'Anna
MAURO MARINONI, Scuola Superiore Sant'Anna
GIORGIO BUTTAZZO, Scuola Superiore Sant'Anna

AMBA AXI is a popular bus protocol that is widely adopted as the medium to exchange data in field-programmable gate array system-on-chips (FPGA SoCs). The AXI protocol does not specify how conflicting transactions are arbitrated and hence the design of bus arbiters is left to the vendors that adopt AXI. Typically, a round-robin arbitration is implemented to ensure a fair access to the bus by the master nodes, as for the popular SoCs by Xilinx.

This paper addresses a critical issue that can arise when adopting the AXI protocol under round-robin arbitration; specifically, in the presence of bus transactions with heterogeneous burst sizes. First, it is shown that a completely unfair bandwidth distribution can be achieved under some configurations, making possible to arbitrarily decrease the bus bandwidth of a target master node. This issue poses serious performance, safety, and security concerns. Second, a low-latency (one clock cycle) module named *AXI burst equalizer* (ABE) is proposed to restore fairness. Our investigations and proposals are supported by implementations and tests upon three modern SoCs. Experimental results are reported to confirm the existence of the issue and assess the effectiveness of the ABE with bus traffic generators and hardware accelerators from the Xilinx's IP library.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Hardware → Reconfigurable logic and FPGAs**; **Programmable interconnect**; **Safety critical systems**;

Additional Key Words and Phrases: FPGA, AXI BUS, Arbitration, Embedded Systems

ACM Reference Format:
Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. 2019. Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs. *J. ACM* 00, 0, Article 000 (2019), 22 pages. https://doi.org/xxx

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2019

Authors' addresses: Francesco Restuccia, francesco.restuccia@santannapisa.it, Scuola Superiore Sant'Anna, Pisa, Italy; Marco Pagani, marco.pagani@santannapisa.it, Scuola Superiore Sant'Anna, Pisa, Italy, CRIStAL (Univ. Lille, CNRS, Centrale Lille, UMR 9189), Lille, France; Alessandro Biondi, alessandro.biondi@santannapisa.it, Scuola Superiore Sant'Anna, Pisa, Italy; Mauro Marinoni, mauro.marinoni@santannapisa.it, Scuola Superiore Sant'Anna, Pisa, Italy; Giorgio Buttazzo, giorgio.buttazzo@santannapisa.it, Scuola Superiore Sant'Anna, Pisa, Italy.

0004-5411/2019/0-ART000 $15.00
https://doi.org/xxx

1 INTRODUCTION

The AMBA AXI bus protocol was proposed by ARM more than 15 years ago as an evolution of previous solutions (such as AMBA APB and AMBA AHB) for internal connections in System-on-Chips (SoCs). Since then, the protocol has been widely adopted due to a number of advantages, such as being an open standard that is fully compatible with the popular ARM Cortex processors. The range of platforms that rely on AMBA AXI has grown, reaching devices with a level of complexity and flexibility way beyond those of the initial chips targeted by the protocol. A relevant case is the one of SoCs equipped with *field-programmable gate arrays* (FPGA) where the programmable logic (PL) communicates with the memory controller through the AMBA AXI protocol. AXI will be even adopted in the upcoming new-generation platforms by Xilinx that rely on a network-on-chip with AXI-based links [25].

FPGA SoCs are promising computing platforms to manage the increasing software complexity and computational demand of modern embedded systems. In particular, they are particularly attractive as they allow deploying high-performance, yet energy-efficient hardware accelerators onto the FPGA fabric, while making use of the plethora of pure software solutions (such as Linux, communication stacks, drivers, etc.) that can be executed upon the multicore processor(s) available in the SoC.

In these platforms, it is possible to deploy very different kinds of hardware modules, even dynamically programmed at run-time, which act as masters on the bus and are allowed to issue bus transactions with different structures. During the last decade, component-based design for applications that make use of FPGAs has been more and more adopted, as it allows designers to build systems by combining Intellectual Properties blocks (IPs) developed by different teams or even different companies. This trend has been accelerated with the development of advanced high-level synthesis (HLS) tools [12]. For instance, some companies sell proprietary IP libraries, which are usually distributed in a closed-source form that forbids the designers working at the integration level to view or edit their behavioral functional description.

Furthermore, it is also common that closed-source IPs do not allow controlling the way they issue bus transactions. This limitation becomes particularly relevant for the case of hardware accelerators, which are typically *memory-intensive* as they commonly work on large amount of data, and hence generate a consistent traffic on the bus. Indeed, the way bus transactions for memory access are managed in the SoC is crucial for the system performance [13, 27], especially when time-sensitive systems are designed.

Nevertheless, the AXI protocol leaves flexibility on the way bus transactions are structured, while not mandating specific arbitration policies, hence leaving the burden of designing bus arbiters to the vendors that adopt AXI. To ensure a *fair* redistribution of the bus bandwidth, the most common solution consists in adopting the round-robin arbitration policy, as for the popular SoCs by Xilinx (e.g., those belonging to the Zynq-7000 and Zynq-Ultrascale families). Despite the use of a round-robin arbitration results in a simple and effective solution, we found that its integration with the AXI protocol may lead to critical issues under specific configurations of the bus transactions.

Contributions. This paper first shows that the integration of the round-robin policy with the AXI protocol may lead to a *completely unfair* bandwidth distribution in the presence of bus transactions with heterogeneous burst sizes. Just to mention a representative result, in a scenario with three hardware accelerator IPs it is possible to reduce the memory bandwidth of a victim IP of about 91%, with respect to the expected bandwidth (i.e., the one assigned by a fair arbitration). In other scenarios, it is even possible to reach larger bandwidth reductions, bringing the available bandwidth of the victim IP arbitrarily close to 0%, thus completely jeopardizing its functionality.

The problem is shown to be originated by the structure of the memory transactions generated by each IP—hence, by a local property of each IP— and poses serious risks for at least two reasons. First, when applying component-based design, where IPs from different vendors or developed by different teams are integrated, this issue can be triggered any time when the structure of the memory transactions of each IP cannot be controlled at the stage of integration, e.g., in the presence of proprietary IPs. The presence of IPs with different burst sizes is a very plausible scenario, as the structure of memory transactions can even be controlled from *high-level synthesis* (HLS) by means of specific pragma statements. Second, since modern FPGAs offer *dynamic partial reconfiguration* (DPR) capabilities, which allows reprogramming a portion of the FPGA area while the system is operating, the system can be exposed to performance/safety/security issues if malicious IPs are dynamically configured to generate crafted bus transactions, e.g., to realize a denial-of-service attack.

A practical solution consisting in a low-latency (one extra clock cycle) IP named *AXI burst equalizer* (ABE) is proposed to restore fairness. The ABE mediates the bus traffic generated by IPs by (i) splitting outgoing address requests to an equalized (i.e., nominal) burst size, (ii) merging/splitting incoming data, and (iii) equalizing the number of outstanding transactions. The ABE has been evaluated upon three modern FPGA-based SoCs (two of the Zynq-7000 family and one from the Zynq-Ultrascale family by Xilinx) by means of benchmarks and case studies, demonstrating its ability to restore a fair bandwidth distribution in accessing the memory bus. Despite the experimental results reported in the paper focus on Xilinx FPGA SoCS, the ABE also applies to other products that adopt AXI and rely on round-robin bus arbitration.

2 ESSENTIAL BACKGROUND

A typical FPGA SoC architecture combines a *processing system* (PS) (generally based on one or more processors) with a Field Programmable Gate Array (FPGA) subsystem in a single device. Both the subsystems access a DRAM controller through which they can access a shared DRAM memory. Figure 1 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) [1].

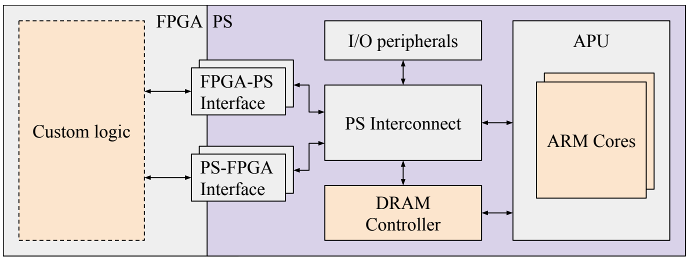

Fig. 1. Simplified architecture of a SoC FPGA platform.

The AXI bus. The AMBA AXI standard defines a master-slave interface to allow simultaneous bi-directional data exchange. An AXI port *interface* 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). Each channel is composed by a standard-defined set of *signals*. In an AXI-based interconnection, all the transactions are started by a master, which requests to read/write data from/to a slave interface through AR or AW channels, respectively. Data reads are routed back to the requesting device through R channels, while data writes are routed to the correct destination through W channels. The B channel is used by the destination device of a write request to acknowledge the master that its request has been correctly served. Each of the five AXI channels implements a handshake mechanism based on two signals: the *ready* signal and the *valid* signal. Depending on the channel type, one signal is controlled by the master interface, while the corresponding slave interface controls the other. When both signals are high, the request or the data are transmitted. 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, the number of outstanding transactions is limited by the designer of the IP. Following data requests on the AR and AW channels, 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 as requests have been routed to the corresponding address channel. This means that data channels are completely dependent on address channels, 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.* This feature is reported in the documentation of many commercial devices [18, 19] and has been experimentally validated for the platforms analyzed in this paper (see Section 5).

AXI ports. As it is illustrated in Figure 1, 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 slave interface to the FPGA and is used by the processors to control the hardware devices or access data in the FPGA. In a dual manner, the second one offers a slave interface to the PS and is used by modules deployed on the FPGA (e.g., hardware accelerators) to access the central DRAM memory or the on-chip memory in the PS. In the considered architecture, the slave port interface on the PS side (FPGA-PS interface) is split into different ports. Each of such ports allows accessing a single, but configurable range of contiguous addresses in the PS. Note that, when two requests refer to disjoint address ranges mapped to different ports, no bus arbitration is required, as they can be served in parallel.

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, which decides at each time which input port is granted to the output channel. The arbiters are completely independent from each other.

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 structure described above.

Arbitration policy. In this work, each arbiter is assumed to implement a round-robin policy, which to the best of our records is the most common solution in 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 [20, 24]). Round-robin arbitration should guarantee fairness in contending the bus; specifically, it should guarantee a fair distribution of the bus bandwidth among the IPs that contend a port.

Burst sizes. 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. Typically, hardware accelerators work on large amount of data. Therefore, burst transactions are preferable than single transactions to avoid issuing a large number of addresses, hence reducing the overhead of the hardware accelerator related to its addressing phases.

AXI allows for bursts of different sizes. The burst size is a property of each IP configured by the developer. Burst sizes may also be implicitly set to default values when using libraries or high-level synthesis (HLS), with the result that designers may be unaware of the burst size adopted by their IP. The configuration of burst sizes is also exposed to HLS with specific pragma statements. In general, when developing IPs in HDL, the developer has a lot of freedom in configuring the burst sizes, which may even change at run-time. Overall, when deploying multiple IPs on the same FPGA, possibly developed by different teams or even different companies, it is likely to have multiple masters on the bus that issue transactions with *heterogeneous* burst sizes. Furthermore, when integrating commercial closed-source IPs, it may not even be possible to configure their burst sizes.

3 PROBLEM DESCRIPTION

This section illustrates a problem that emerges when stimulating round-robin arbiters on address read *AR* and address write *AW* channels with address burst requests of different sizes. We focus on the arbitration required to solve conflicts of requests that target the same output port, and we hence assume that all the bus traffic is routed to the same port. Note that this setting represents the worst-case scenario from the point of view of bus contention. Moreover, being the number of available ports limited, scenarios in which a port is contended by multiple masters are common in typical realistic designs.

A simple experimental setup has been used to show the problem. Consider a general AXI Interconnect with one output port connected to the FPGA-PS interface and three Direct Memory Access IP blocks (DMA) D_1, D_2, and D_3, each connected to one of the input ports of the considered AXI Interconnect.

The time needed by D_2 to complete a memory transfer has been measured, while D_1 and D_3 have been used to generate continuous interference in accessing the memory. Specifically, D_2 is the DMA under test, configured to perform a memory transfer of 128 KB, while D_1 and D_3 manage memory transfers of 4 MB. The purpose of this experiment is to measure the bandwidth of D_2 by (i) varying the size s of the bursts issued by D_1 and D_3, (ii) while keeping the size of the bursts issued by the IP under analysis D_2 *constant* and equal to 16 words.

The bandwidth of D_2 can be obtained as the ratio between 128 KB and the time needed to complete the transaction. Such a bandwidth can finally be normalized to the total bus bandwidth to compute the percentage of bandwidth used by D_2. This test has been executed on both the Xilinx Zynq-7020 and the Xilinx Ultrascale+ platforms using the AXI SmartConnect and the DMA IPs by Xilinx (note that these IPs allow configuring burst sizes from 2 to 256 words, which corresponds to the range specified in the AXI standard definition).

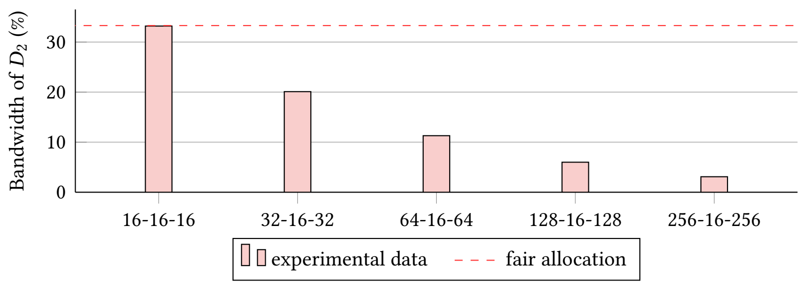

Fig. 2. Percentage of the bandwidth used by D_2 as a function of the burst size of D_1 and D_3. The labels below each pair of bars indicate the burst size of D_1, D_2 and D_3, respectively.

The red bars in Figure 2 report the results of the experiment for $s \in \{16, 32, 64, 128, 256\}$. Ideally, the round-robin arbitration should guarantee a fair redistribution of the available bandwidth, reserving one third of the total bandwidth (about 33%) to each DMA. However, as clear from Figure 2, this is the case only when all the three DMAs issue transactions with the same burst size. The results also show that the bandwidth of D_2 significantly decreases as the burst size of the other DMAs increases, reaching about the 3% of the total bandwidth for $s = 256$. Note that the latter case corresponds to just the 9% of the expected bandwidth, which means a 91% drop with respect of the expected bandwidth! To make things worse, a larger bandwidth reduction can be obtained by increasing the number of interfering DMAs, reaching configurations in which the execution of a DMA is completely jeopardized (further details will be provided in Section 5). From this experiment, it is possible to conclude that *the integration of round-robin arbitration with the AXI protocol can cause an unfair distribution of the bandwidth in the presence of heterogeneous burst sizes.* As mentioned in the introduction, this issue may be critical when integrating multiple IPs developed by independent teams (possibly with closed-source, and hence unknown non-configurable burst sizes), or in the presence of systems in which part of the FPGA could be reconfigured to maliciously introduce "bandwidth-stealing" modules.

3.1 In-depth analysis

This section aims at analyzing in details the problem presented above. To this end, consider the design illustrated in Figure 3. The design is composed of three hardware accelerators ($\mathtt{HWA}_1$, $\mathtt{HWA}_2$, and $\mathtt{HWA}_3$) that issue burst transactions (reads or writes) to the central memory controller (DRAM CTRL) located in the PS, while the AXI Interconnect arbitrates the address burst requests to access the OUT slave port interface in the FPGA-PS interface. $\mathtt{HWA}_2$ issues burst transactions with a length of S words. Differently, $\mathtt{HWA}_1$ and $\mathtt{HWA}_3$ issue burst transactions with size $2S$. For instance, $\mathtt{HWA}_1$ can be a typical hardware task synthesized with HLS, while $\mathtt{HWA}_2$, and $\mathtt{HWA}_3$ can be two DMAs.

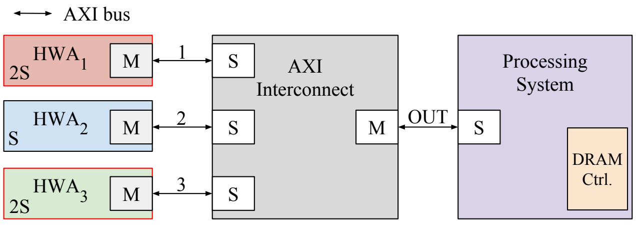

Fig. 3. Sample architecture to probe the bandwidth allocation in the presence of heterogeneous burst size transactions.

Consider the case in which the three hardware accelerators simultaneously issue a read request on the corresponding AXI Address Read channel AR_1, AR_2, and AR_3, respectively (the same example can be extended to write channels). Also, to cope with the worst-case scenario, it is assumed that the corresponding hardware accelerators can provide a new request immediately after the AXI Interconnect granted a request and presented it to the slave OUT Port Address Channel in the FPGA-PS interface (named AR_{OUT}).

Figure 4 reports a simplified waveform diagram re-created by observing real traces on a Xilinx Zynq SoC platform (a bus signal analysis has been performed upon three state-of-the-art platforms, namely the Zynq Z-7010, Zynq Z-7020, and the Zynq Ultrascale+ ZU9EG, obtaining the same results). The corresponding design has been developed with Xilinx Vivado 2018.2, making use of the state-of-the-art Xilinx IPs.

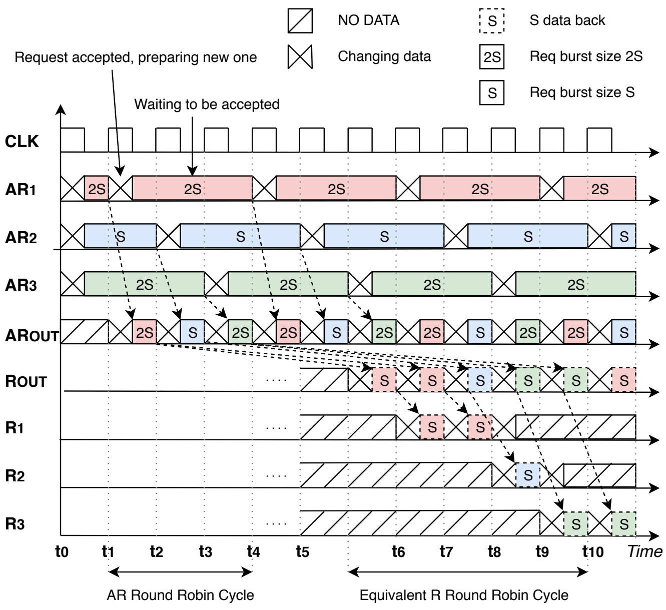

Fig. 4. Simplified waveform of the arbitration for the sample architecture of Figure 3.

The diagram illustrates the input address requests and the corresponding outputs in accordance to the round-robin arbitration policy. To simplify the presentation and focus on relevant aspects, it is assumed that **(i)** the PS is always ready to accept address requests from the AXI slave port under analysis in the FPGA-PS interface (i.e., the one related to the OUT port); and **(ii)** the read channels (i.e., R_{OUT}, R_1, R_2, and R_3 in the figure) have a width of S words each, so that S words can be transferred in one clock cycle.

As it can be noted from the diagram, at time t_1, each of the three hardware accelerators provides its first burst request (i.e., AR_1, AR_2, and AR_3). At time t_2, the round-robin arbiter on AR channel decides to accept the request coming from $\mathtt{HWA}_1$ and routes it through the AR channel of the OUT port. After the request is sampled, a new one is provided by $\mathtt{HWA}_1$ on AR_1. Analogously, the request coming from $\mathtt{HWA}_2$ is admitted at time t_3, while the one coming from $\mathtt{HWA}_3$ is granted at time t_4. This completes the round-robin period of the input AR channels (each hardware accelerator has routed a request to the output). Hence, at time t_5, a new request can be accepted from $\mathtt{HWA}_1$.

At time t_6, the first chunk of data is provided from memory on the common read channel R_{OUT}. Since the address requests are served in order, this data is related to the first request accepted from $\mathtt{HWA}_1$; thus, the chunk is routed to $\mathtt{HWA}_1$ through its read channel R_1. The second chunk of data coming from memory at time t_7 is still related to $\mathtt{HWA}_1$, so it is also routed to R_1. The same happens at time t_8 for the data requested by $\mathtt{HWA}_2$ (single chunk), and at times t_9 and t_{10} for the data requested by $\mathtt{HWA}_3$ (two chunks).

The diagram reported in Figure 4, and in particular the timelines of channels R_{OUT}, R_1, R_2, and R_3, allow us to clearly identify the cause of the problem presented in the previous section. Despite the requests on the address channel AR_{OUT} are accepted in round-robin order, their corresponding data transfers to the hardware accelerators do not respect a perfectly-balanced round-robin order (see the R_{OUT} channel)—that is, *the round-robin arbitration is preserved on address channels, but with different per-master granularity, which depends on the burst sizes.* Indeed, as data must be transmitted in accordance to the order with which the address requests are granted, they cannot be provided on channels R_1, R_2, and R_3 in a fair manner, i.e., with the granularity of S words per hardware accelerator.

3.2 Analytical characterization

From a theoretical point of view, in the above example each hardware accelerator should have been able to leverage 1/3 of the total bandwidth. However, this is not the case, since the round-robin arbiter on address channels performs the arbitration without considering the burst sizes, thus granting a request of $2S$ words to both HWA_1 and HWA_3, and only S words to HWA_2. The final result is that the read channel R_{OUT} allocates 2/5 of the bandwidth to both HWA_1 and HWA_3, and only 1/5 of the bandwidth to HWA_2.

From this analysis, it is possible to derive a formula that calculates the actual bandwidth B_i of a hardware accelerator HWA_i in the case in which **(i)** it requests bursts of length β_i words; **(ii)** while sharing a port via an AXI Interconnect with others $N-1$ hardware accelerators requesting β_{others} words per request (with $\beta_i \leq \beta_{\text{others}}$). First note that, the total amount of data transmitted in a round-robin cycle is $(N-1) \cdot \beta_{\text{others}} + \beta_i$. The fraction of data for the hardware accelerator under analysis within each round-robin cycle is hence given by $\beta_i / ((N-1) \cdot \beta_{\text{others}} + \beta_i)$, which can be rewritten as

$$B_i = \frac{1}{\frac{\beta_{\text{others}}}{\beta_i} \cdot (N-1) + 1}. \tag{1}$$

Consequently, the ratio between the real bandwidth and the one expected with a fair round-robin arbitration can be computed as:

$$\frac{B_i}{\frac{1}{N}} = \frac{N}{\frac{\beta_{\text{others}}}{\beta_i} \cdot (N-1) + 1}. \tag{2}$$

The analysis can easily be extended to the general case in which each hardware accelerator has a different burst size. Consider a set of hardware accelerators $\mathcal{H} = \{\mathsf{HWA}_1, \ldots, \mathsf{HWA}_N\}$ connected to an interconnect I. Each hardware accelerator $\mathsf{HWA}_i \in \mathcal{H}$ has a burst size of β_i. The bandwidth assigned by the interconnect to HWA_i can be computed as

$$B_i = \frac{\beta_i}{\sum_{\mathsf{HWA}_j \in \mathcal{H}} \beta_j}. \tag{3}$$

Table 1. Percentage of the total bandwidth allocated to a hardware accelerator in the presence of other interfering hardware accelerators.

Interfering accel.		Burst size of interfering accelerators				
		16	**32**	**64**	**128**	**256**
Num.	**1**	50.0%	33.34%	20.0%	11.11%	5.88%
	2	33.4%	20.0%	11.11%	5.88%	3.03%
	3	25.0%	14.29%	7.69%	4.0%	2.04%
	4	20.0%	11.11%	5.88%	3.03%	1.54%
	5	16.67%	9.09%	4.76%	2.44%	1.24%
	6	14.29%	7.69%	4.00%	2.04%	1.03%
	7	12.50%	6.67%	3.45%	1.75%	0.89%

Table 1 reports the percentage of total bandwidth, computed by Equation (2), which is allocated to a hardware accelerator in the presence of up to seven interfering hardware accelerators. The burst size of the hardware accelerator under analysis remains fixed to 16, while the burst size of the interfering hardware accelerators is the same and varies from 16 to 256. As it can be noted from the table, in the presence of seven interfering hardware accelerators with burst size equal to 256, the bandwidth of the victim IP can be reduced to the 0.89% of the total bus bandwidth!

4 PROPOSED SOLUTION

This work addresses the issue presented in the previous section by proposing a practical solution that consists of an IP named *AXI Burst Equalizer* (ABE). The ABE is conceived to be placed between each hardware accelerator and an input port of an AXI Interconnect and serves the purpose of *equalizing* the address burst requests issued by the hardware accelerator. The main objective of the ABE is to achieve a fair bus bandwidth allocation in the presence of round-robin arbitration. This feature is particularly useful when hardware accelerators are subject to worst-case timing analysis, e.g., in the case they are part of a real-time system [3].

The ABE preserves standard interconnections by employing two AXI ports: the master port of the hardware accelerator connects to a slave port of the ABE, while the slave input port of the AXI Interconnect connects to a master port in the ABE, as illustrated in Figure 5. The ABE implements the standard AXI handshaking mechanism and supports any permissible burst size.

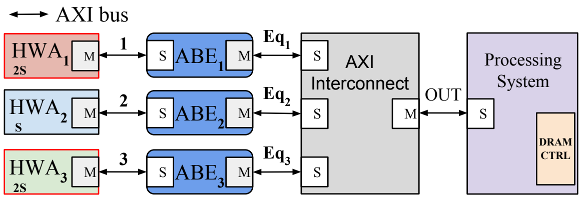

Fig. 5. Extension of the sample architecture to include the ABEs.

An ABE is characterized by two configuration parameters, i.e., **(i)** a *nominal* burst size $\widehat{\beta}$, and **(ii)** a maximum number $\widehat{L}$ of outstanding transactions, and complies with the following rules:

R1 In accordance to the AXI standard, each hardware accelerator can initiate a transaction by issuing to its corresponding ABE two types of requests: **(i)** address read requests, denoted by R_R, with burst size β_R; and **(ii)** address write requests, denoted by R_W, with burst size β_W.

R2 If $\beta_R \leq \widehat{\beta}$ (or $\beta_W \leq \widehat{\beta}$ for write requests), the ABE forwards the request issued by the corresponding hardware accelerator to the AXI Interconnect without any modification. Otherwise, when $\beta_R > \widehat{\beta}$ (or $\beta_W > \widehat{\beta}$), request R_R (or R_W) is split into $C = \left\lceil \beta_R/\widehat{\beta} \right\rceil$ sub-requests R_{R_1},R_{R_2},...,R_{R_C} (or $C = \left\lceil \beta_W/\widehat{\beta} \right\rceil$ sub-requests R_{W_1},R_{W_2},...,R_{W_C}) that have at most the nominal burst size $\widehat{\beta}$.

R3 The ABE *sequentially* propagates the sub-requests mentioned in the previous rule to the AXI Interconnect.

R4 When an address read request R_R is split into the R_{R_1},R_{R_2},...,R_{R_C} sub-requests, the ABE reassembles the corresponding incoming data chunks D_{R_1},D_{R_2},...,D_{R_C} into a single data response D_R that is compliant (according to the AXI standard) with the original request R_R.

R5 When an address write request R_W is split into the R_{W_1},R_{W_2},...,R_{W_C} sub-requests, the ABE also splits the corresponding outgoing write data D_W into C outgoing data chunks D_{W_1},D_{W_2},...,D_{W_C}. The corresponding write responses K_{W_1},K_{W_2},...,K_{W_C} on the B channel are merged into a single write response K_W.

R6 The ABE can issue at most $\widehat{L}$ nominal outstanding read transactions and $\widehat{L}$ nominal outstanding write transactions. Once the limit of outstanding transactions has been reached, the ABE must wait for the completion of a pending transaction before issuing a new transaction to the AXI Interconnect.

An implementation of the ABE has been realized in VHDL language and exported as an IP-XACT standard package. It can be integrated on any platform compatible with the IP-XACT standard, as well as easily ported and packaged to other platforms that are not compatible with the IP-XACT standard. The internal hardware architecture of the ABE is presented in Appendix 4.5. Our implementation leverages cycle-level parallelism to minimize the latency introduced when splitting transactions: further details are provided in Section 4.2. The reason for which the ABE limits the number of outstanding transactions (rule R6) is explained in Section 4.3.

4.1 Example: the ABE in action

Let us now consider the same example introduced in Section 3 and analyze the differences in the bus signals when the ABEs are installed (the same example can be extended to the case of write transactions). As in Section 3, $\mathtt{HWA}_2$ issues address burst requests S words long, while $\mathtt{HWA}_1$ and $\mathtt{HWA}_3$ issue address burst requests $2S$ words long. The equalized channels mediated by the ABEs are denoted with the subscripts eq1, eq2 and eq3. The nominal burst size $\widehat{\beta}$ is set to S. The (simplified) bus signals in the presence of ABEs are illustrated in Figure 6 (this figure has also been created by observing real execution traces obtained from the same platforms considered in the previous section).

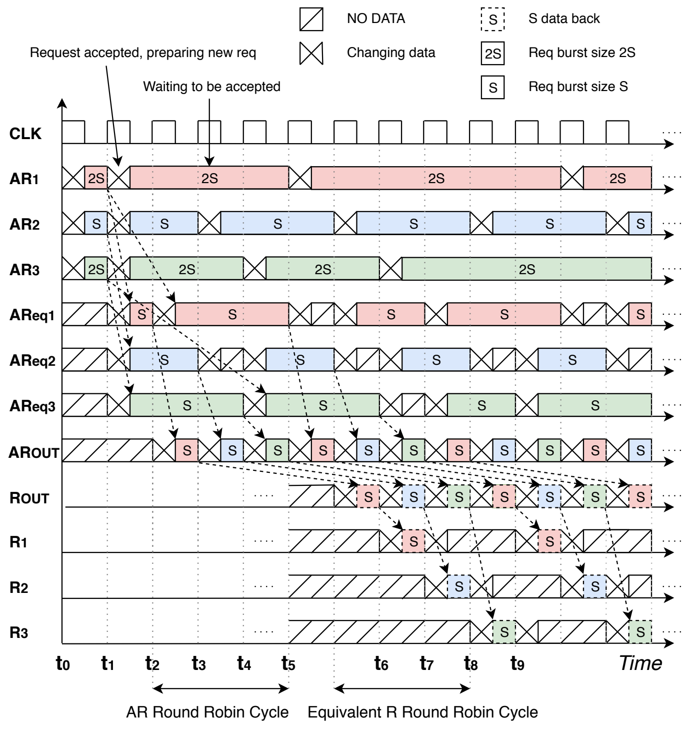

Fig. 6. Waveform of an arbitration sequence in the presence of ABEs.

Each hardware accelerator issues a burst request at time t_0 on the respective address read channel (AR_1, AR_2, and AR_3) to be routed to the PS through the OUT port in the FPGA-PS interface. At time t_1, the ABEs (ABE_1, ABE_2, and ABE_3) latch the address requests coming from the respective hardware accelerators, thus releasing the buses (AR_1, AR_2 and AR_3) and allowing the hardware accelerators to provide a new burst request. Since the size of the burst request coming from $\mathtt{HWA}_2$ is the same of the nominal one, this request is directly routed to the output AR_{eq2} channel without modifications (rule R2). Conversely, since $\mathtt{HWA}_1$ and $\mathtt{HWA}_3$ provide requests of size $2S$, ABE_1 and ABE_3 issue to AR_{eq1} and AR_{eq3}, respectively, partial burst requests with length S words (rule R2).

At time t_2, the AXI Interconnect has in input three bursts requests of S words from ABE_1, ABE_2 and ABE_3. Let us suppose that the first granted request is the one on AR_{eq1}. Hence, the first partial burst request coming from $\mathtt{HWA}_1$ is granted and immediately latched on the AR_{OUT} channel at time t_3. At the same time, ABE_1 generates the next partial burst request, which completes the original data request of size $2S$ and will become available on the AR_{eq1} channel at time t_3. Next, at time t_4, the request on AR_{eq2} is granted to the output port by the round-robin arbiter. Since it has the nominal burst size, it can be routed as it is by ABE_2. After that, a new nominal burst request on the AR_2 channel is provided to the AR_{eq2} channel by ABE_2. Then, similarly as for the request generated by HWA_1, the first chunk of the request issued by HWA_3 is granted and latched on the AR_{OUT} channel at time t_5: this concludes the first round-robin cycle. The same arbitration scheme is repeated until the addressing phases are completed. As the requests have been granted by following a round-robin arbitration scheme with a fair burst granularity of S words, the corresponding data are also transmitted back on the R_1, R_2, and R_3 channels with a granularity of S words (see Fig. 6 from time t_6 on). This allows achieving a fair bandwidth redistribution for the bus, with a consequent impact on the response times of the hardware accelerators — please refer to Appendix 4.4 for further details.

4.2 Extra latency introduced by the ABE

Implementing the ABE in a HDL language allowed achieving high performance and optimizing its area consumption. The ABE has been designed to leverage the parallelism offered by the AXI protocol to introduce as less latency as possible. It analyzes AXI channels in parallel by defining a specific VHDL process for each channel. Note that, as the ABE has to analyze the request for transactions to identify their burst size, at least one cycle of latency is required to process the request. Our implementation is able to perform the analysis of requests in the minimum time (one cycle). Conversely, when managing the data channels, our implementation does not introduce additional latency as **(i)** it directly connects the upstream and downstream data channels; and **(ii)** monitors the data traffic to proactively perform splitting and merging tasks. Specifically, knowing the nominal burst size of transactions, the ABE counts the number of words transmitted on data channels and can hence be prepared to either split (by inserting the AXI-standard LAST signal on the W channel) or merge (by removing LAST signals on the R channel) transactions *one clock cycle in advance.*

Figure 7 reports three examples to illustrate the latency introduced by the ABE: the first two are related to transactions issued without the ABE, and are used to explain the third example that refers to the case in which the ABE is present. To simplify the figure, only 8-word write transactions are considered (the same examples can be extended to other sizes and read transactions).

First of all, note that a 8-word write data request can be served via either **(i)** a single burst transaction with size 8 words; or **(ii)** with eight one-word long transactions. These two options are illustrated by Case 1 and Case 2 in Figure 7, respectively. The bus traffic on both the address (AW) and data (W) channels is considered.

Case 1. In this case, a 8-word burst is issued by a hardware accelerator to the memory controller. At time t_1, the address burst request is accepted on the AW channel; hence, at the next clock cycle, the hardware accelerator can start providing data on the W channel (time t_2). Assuming (to simplify

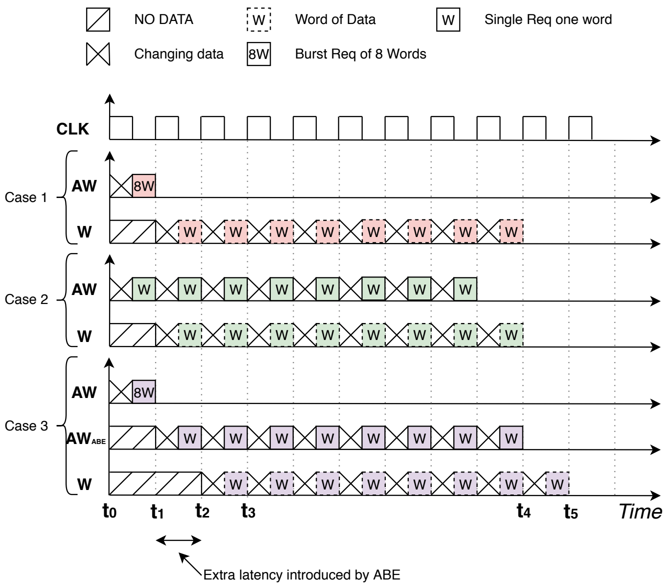

Fig. 7. Latency introduced by ABE to serve an 8-word write transaction.

the example) that the output port is always ready to accept data on the W channel, at time t_4 all the eight words have been transferred.

Case 2. In this case, the hardware accelerator still wants to transfer eight words of data, but issuing eight single-word transactions. According to the AXI standard, each word of data on the W channel has to be preceded by the corresponding address request on the AW channel. At time t_1, the first address request is accepted on the AW channel. Assuming that the output port is always ready to accept data from the W channel, the hardware accelerator provides the first word of data at the next clock cycle (time t_2). Thanks to the *parallelism* of AXI (i.e., the presence of multiple channels), in the same clock cycle the hardware accelerator can provide a new address request for the next word of data on the AW channel, while also transmitting data on the W channel. Consequently, the eight transactions complete in the same amount of clock cycles needed in Case 1.

Case 3. Here, the ABE is installed to mediate the traffic issued by the hardware accelerator under analysis. The ABE is configured with a nominal burst size of length one word. As in Case 1, the hardware accelerator issues a 8-word address burst request to the ABE on the AW channel at time t_1. The ABE receives the request and, in one clock cycle, is able to provide the first nominal address burst request to the AW_{ABE} channel at time t_2. This extra clock cycle corresponds to the only latency introduced by the ABE. Indeed, at time t_3, the hardware accelerator can issue the first word of data in the data channel W (corresponding to the first nominal burst) and, since data channels are not decoupled by ABE, data are accepted directly at the output port without any other latency. At the same clock cycle, the ABE is ready to provide the next nominal address burst request on the AW_{ABE} channel, and so on until the data transfer is completed. The transaction completes at time t_5, which is just one clock cycle after the completion of the transaction(s) in Cases 1 and 2. This single-cycle latency is originated by a stage of the ABE that computes (and provides to the output port) the next nominal address burst request. Note that this latency is not affected by the size of the nominal burst, nor by the size of the address burst requests in input to the ABE.

4.3 Limiting the number of outstanding transactions

As discussed in Section 2, the AXI standard supports multiple outstanding transactions by allowing AXI masters modules to issue multiple reads or write address burst requests without waiting for their corresponding completions. The AXI standard itself does not impose any limit on the maximum number of outstanding transactions; however, in practical implementations of IPs, a maximum number of outstanding transactions is typically configured. In general, the number of maximum outstanding transactions depends on the specific IP implementation. Once a master module has reached the maximum number of outstanding transactions, it must wait for the completion of one pending transaction before issuing a new transaction. This practical limitation may be another cause of unfair bandwidth allocation.

For instance, consider two hardware accelerators connected to the same Interconnect: $\mathtt{HWA}_1$ issuing 16-word burst transactions and $\mathtt{HWA}_2$ issuing 64-word burst transactions. Assume that the maximum number of outstanding transactions is set to 2 for both hardware accelerators. Despite the number of outstanding transactions is the same, the ones issued by $\mathtt{HWA}_2$ request four times more data than the ones issued by $\mathtt{HWA}_1$. This means that $\mathtt{HWA}_2$ can issue four times more outstanding data than $\mathtt{HWA}_1$. Note that the same would occur in the presence of ABEs without rule R6.

Now, consider the case in which ABE modules, configured with nominal burst sizes of 16 words, are placed between each hardware accelerator and the Interconnect. In this scenario, the ABE will fragment each 64-word address burst request issued by $\mathtt{HWA}_2$ into 4 ones, each consisting of 16-word bursts. However, from the perspective of $\mathtt{HWA}_2$, only a single transaction has been issued. Therefore, the ABE protecting $\mathtt{HWA}_2$ could potentially issue $4 \cdot 2 = 8$ outstanding 16-word transactions before reaching its original limit (i.e., 2 outstanding 64-word transaction), while $\mathtt{HWA}_1$ can issue only two 16-word outstanding transactions. This difference may lead to an unfair bus bandwidth allocation, as it is illustrated in the (simplified) waveform diagram reported in Figure 8.

At time t_0, both $\mathtt{HWA}_1$ and $\mathtt{HWA}_2$ prepare their first address requests. One cycle later, the ABEs propagate the address requests to their corresponding AR_{eq} channels. The first arbitration cycle starts at time t_1 and ends at t_2. The Interconnect arbitrates the first 4 address requests in a fair way up to time t_3, at which all address requests issued by $\mathtt{HWA}_1$ have been served. At this time, having already reached the limit of 2 outstanding transactions, $\mathtt{HWA}_1$ cannot issue further transactions. From this point on, all address requests from $\mathtt{HWA}_2$ are propagated without suffering contention up to time t_4, when $\mathtt{HWA}_1$ receives back the data corresponding to its first address request and can finally issue a new transaction. In this scenario, $\mathtt{HWA}_1$ issued only 2 transactions during the first 4 arbitration cycles, while $\mathtt{HWA}_2$ issued 6 transactions. Overall, this corresponds to an unfair bus bandwidth allocation in which $\mathtt{HWA}_1$ and $\mathtt{HWA}_2$ are assigned the 25% and the 75% of the total bus bandwidth, respectively.

The analysis presented in Section 3.2 can easily be extended to account for a set of hardware accelerators with different number of outstanding transactions. Considering that each hardware accelerator $\mathtt{HWA}_i \in \mathtt{HWA}$ has a burst size of S_i and can issue L_i outstanding transactions (on each write and read channel), Equation (3) can be extended as follows:

$$B_i = \frac{S_i \cdot L_i}{\sum_{\mathtt{HWA}_j \in \mathtt{HWA}} S_j \cdot L_j}. \tag{4}$$

To cope with this phenomenon, the ABE limits the number of outstanding nominal transactions to a configurable value $\widehat{L}$ (rule R6). This parameter is set as $\widehat{L} = \min_{\mathtt{HWA}_i \in \mathtt{HWA}} \left\lfloor S_i \cdot L_i / \widehat{\beta} \right\rfloor$. In this way, all the hardware accelerators in the system can have at most the same amount of outstanding data, independently of their number of outstanding transactions (and also of their burst size). This

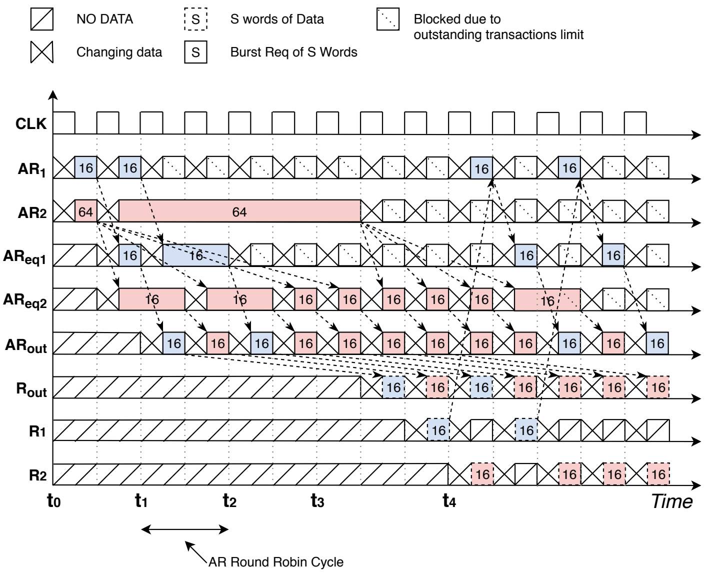

Fig. 8. Waveforms of an arbitration sequence in the presence of ABEs.

feature is essential for complex designs that may include multiple hardware accelerators hardwired with a different maximum number of outstanding transactions.

4.4 Response times with ABEs

As discussed in Section 3, hardware accelerators that issue requests with large bursts are privileged in contending the bus bandwidth. When ABEs are installed, a fair bus bandwidth redistribution is restored: this means that some hardware accelerators will experience larger response times with respect to the case of a stock configuration of the bus, while others will experience shorter response times. This section aims at briefly quantifying this phenomenon. Only the case of read requests is addressed (the case of write requests follows analogously).

Example. To begin, consider as an example the same architecture described in Section 3 and the corresponding waveform diagram in Figure 4. Assume that ABEs are not installed. Let HWA_3 be the hardware accelerator under analysis. For each Interconnect round-robin cycle on the address read channels, in the worst-case scenario each $2S$-word burst request issued by HWA_3 can be preceded by a $2S$-word burst request issued by HWA_1 and one S-word burst request issued by HWA_2. Therefore, the time needed for HWA_3 to complete a request includes the time required for the data phases of HWA_1 and HWA_2, i.e., it requires $2S + S + 2S = 5S$ clock cycles to retrieve $2S$ words of data ($2S$ clock cycles for the data phase of HWA_1, S clock cycles for the data phase of HWA_2, and $2S$ clock cycles for its data phase). The same worst-case delay is suffered by burst requests issued by HWA_1 (as it has the same configuration of HWA_3). Now, consider a request issued by HWA_2. In the worst-case, the request suffers the same delay of the ones issued by HWA_1 and HWA_3; however, note that it retrieves just S words of data (instead of $2S$). If $2S$ words of data are used to compare the delay suffered by the three accelerators, HWA_2 can experience a worst-case delay equal to $2 \cdot (2S + 2S + S) = 10S$ to retrieve the same data that HWA_1 and HWA_3 can get in a single request. Note that the per-word delay of HWA_2 is twice the one of HWA_1 and HWA_3.

Now, consider the case in which ABEs are installed and the waveform diagram of Figure 6. Requests with burst sizes larger than the nominal one are split. This means that, considering a nominal burst size of S, each request coming from $\texttt{HWA}_1$ and $\texttt{HWA}_3$ is split into two requests. In the worst-case scenario, both the sub-requests lose the arbitration in favor of a request issued by each of the other two accelerators. Therefore, a $2S$ words burst request issued by $\texttt{HWA}_1$ or $\texttt{HWA}_3$ is served with a worst-case delay equal to $2 \cdot (S + S + S) + 1 = 6S + 1$, where the term $+1$ accounts for the additional clock cycle of latency introduced by the ABE as described in Section 4.2. Since $\texttt{HWA}_2$ is issuing burst requests S words long, two burst requests need to be issued to fetch $2S$ words of data. In the worst-case scenario, both the requests lose the arbitration and are preceded by 2 sub-requests S words long issued by $\texttt{HWA}_1$ and $\texttt{HWA}_3$, respectively, experiencing the same worst-case delay of $\texttt{HWA}_1$ and $\texttt{HWA}_3$. Hence, in this case the per-word delay is the same for all the accelerators.

It is worth noting that the increase of the worst-case delay of burst requests issued by $\texttt{HWA}_1$ and $\texttt{HWA}_3$ does not merely correspond to a performance drop; rather, it should be seen as an effect introduced by the fair bandwidth distribution. Overall, with the ABEs, the worst-case delay in accessing a given amount of data is constant and does not depend on the structure of the interfering transactions (i.e., their burst size).

Generalization. To generalize the analysis of the previous example, consider a case with N hardware accelerators, with $\texttt{HWA}_i$ being the hardware accelerator under analysis. $\texttt{HWA}_i$ issues burst requests with size β_i, and each of them is split into $\lceil \beta_i / \widehat{\beta} \rceil$ nominal sub-requests by the ABE. Each other hardware accelerator $\neq \texttt{HWA}_i$ can interfere with at most one nominal sub-request per round-robin cycle, i.e., for a total of $(N-1) \cdot \widehat{\beta}$ clock cycles for each sub-request issued by $\texttt{HWA}_i$. Hence, the total worst-case delay suffered by $\texttt{HWA}_i$ to complete a burst request β_i-word wide is given by

$$\lceil \beta_i / \widehat{\beta} \rceil \cdot (N-1) \cdot \widehat{\beta} + 1.$$

Note that, thanks to the ABE, this delay is analytically independent of the burst sizes of the other accelerators.

4.5 Hardware architecture of the ABE

The internal hardware architecture of the ABE is illustrated with a block diagram in Figure 9. The ABE is implemented with five VHDL processes, each in charge of managing a corresponding AXI channel. The processes operate in parallel and are sensitive to the rising edge of the system AXI clock. This design choice allows keeping track of the whole AXI bus while preserving the intrinsic parallelism of the AXI standard. Please also refer to Section 4 for the functionality of the ABE discussed in the following.

The *address read splitter process* (ARP process in short) manages the AXI Address Read channel (AR Channel). The process is in charge of splitting the address *read* burst requests R_R coming from the master module (i.e., a hardware accelerator) and provides the C sub-requests R_{R_1},R_{R_2},...,R_{R_C} to the slave port. For each address burst request received, the number C of sub-requests into which the request is split is sent to the *read merger* process (RP process in short). The ARP process includes a counter to keep track of the current number of outstanding transactions (i.e., the ones not yet completed) of the equalized module, and decouples the channel when the maximum number of outstanding transactions set by the user is reached. Such a counter is incremented when a sub-request is accepted at the corresponding interconnect slave interface, and is decremented when an outstanding sub-request completes (a signal is transmitted from the RP process). The ARP process is implemented as a *finite-state machine.*

The *read data merger* process (RP process in short) is in charge of merging the incoming data bursts D_{R_1},D_{R_2},...,D_{R_C} coming from the AXI interconnect to a single request D_R directed to the

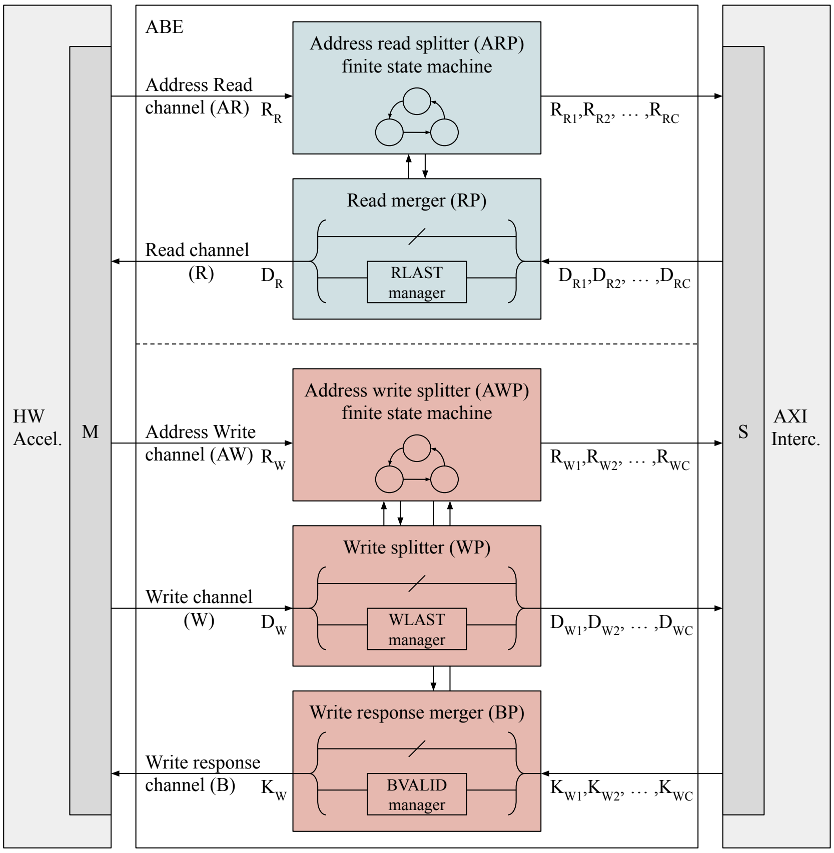
HW Accel.
M
ABE
Address read splitter (ARP) finite state machine
Address Read channel (AR) R_R
$R_{R1}, R_{R2}, \cdots, R_{RC}$
Read merger (RP)
RLAST manager
Read channel (R) D_R
$D_{R1}, D_{R2}, \cdots, D_{RC}$
Address write splitter (AWP) finite state machine
Address Write channel (AW) R_W
$R_{W1}, R_{W2}, \cdots, R_{WC}$
Write splitter (WP)
WLAST manager
Write channel (W) D_W
$D_{W1}, D_{W2}, \cdots, D_{WC}$
Write response merger (BP)
BVALID manager
Write response channel (B) K_W
$K_{W1}, K_{W2}, \cdots, K_{WC}$
S
AXI Interc.

Fig. 9. Internal hardware architecture of the ABE.

hardware accelerator involved in the transaction. It operates in a proactive way (1 clock cycle in advance) on the LAST signal of the AXI Read channel (R channel), while implementing a pass-through logic for the other signals. It uses the information provided by the ARP process to resize the incoming burst data to a single-data response. The RP process acknowledges the ARP process when an outstanding read sub-request has been correctly served.

The *address write requests splitter* process (AWP process in short) implements the same behavior of the ARP process, but on *write* burst requests, hence operating on the AXI Address Write channel (AW channel). The number of sub-requests C into which an address write request R_W is split is communicated to the *write splitter* process (WP process in short) and to the *write response merger* process (BP process in short). As for the ARP process, the AWP includes a counter for keeping track of the number of outstanding transactions, which is incremented when a sub-request is accepted, and decremented when an acknowledge signal related to completed sub-requests is received from the BP process. The AWP process is also implemented as a finite-state machine.

The WP process manages the AXI Write channel (W channel) and is in charge of splitting the outgoing write data D_W into the C chunks D_{W_1},D_{W_2},...,D_{W_C}. The process operates in a proactive way on the WLAST signal, using the information provided by the the AWP process, while implementing a pass-through logic for the other signals on the W channel.

The BP process is in charge of merging the incoming write response coming from the data producer and directed to the hardware accelerator of interest. It operates on the AXI B channel, merging write responses using the information provided by the AWP process. Furthermore, each time the process receives a write sub-response (meaning that an outstanding write sub-request is completed) it acknowledges the AWP process.

5 EXPERIMENTAL RESULTS

This section presents the experimental evaluation that has been conducted to assess the effectiveness of the ABE in restoring a fair bus bandwidth allocation. Two experimental campaigns have been performed. The first one makes use of traffic generators (DMA IPs) that aim at recreating a realistic scenario in which a set of high-performance hardware IPs (e.g., accelerators) concurrently access a shared memory through an AXI Interconnect, as illustrated in Figure 5. The second one is a case study. Furthermore, the resource consumption of the ABE has been quantified.

5.1 Tests with DMAs as traffic generators

Experimental setup. The experimental setup adopted in this work has been implemented both on the Xilinx Zynq Z-7020 SoC using the Digilent PYNQ-Z1 board, and on the Zynq UltraScale+ ZU9EG SoC using the ZCU102 evaluation kit. The results obtained on the two platforms were essentially the same. Therefore, due to space reasons, only those obtained on the Xilinx Zynq Z-7020 are reported in the paper. The corresponding designs have been developed with Xilinx Vivado 2018.2, using the state-of-the-art Xilinx IPs.

The setup employs a set of AXI DMA modules deployed on the FPGA, each managed by a corresponding software task running in the PS on top of the FreeRTOS operating system. The AXI DMA modules have been chosen because they provide high-bandwidth memory access and are used in many designs to feed and receive data from AXI4-Stream peripherals. In fact, many relevant Xilinx library IPs like FFT [22], FIR filter [23], and Convolution Encoder [21] make use of the AXI4-Stream interface and require the support of an AXI DMA to access the system memory. Note that the AXI DMA is capable of transmitting and receiving the maximum amount of data per clock cycle (one word) on each channel. For this reason, the AXI DMAs can generate the maximum bus load, and can hence be used to stress the bus under worst-case scenarios.

The test setup comprises three AXI DMA modules (D_1, D_2, and D_3), each configured by their software task to perform the copy of a memory buffer located in the system DRAM. As in Section 3, D_2 is the DMA under test, configured to perform a memory transfer of 128 KB, while D_1 and D_3 manage memory transfers of 4 MB. Each DMA is attached to an ABE, which is in turn connected to an AXI Interconnect as shown in Figure 5. The AXI Interconnect is connected to the FPGA-PS Interface through an AXI port to the DRAM controller located in the PS. This configuration is tested against the case of a stock configuration, i.e., when the DMAs are directly connected to the AXI Interconnect as shown in Figure 3. The Xilinx DMAs are closed-source IPs and their maximum number of outstanding transactions is not documented: nevertheless, by analyzing their behavior, we empirically found that this parameters is set to 6.

Effectiveness of ABEs. The experimental setup described above is the same used to perform the experiments reported in Section 3. The same experiment has been repeated under the configuration in which the ABEs are installed, varying the burst size of two DMAs (D_1 and D_3) and keeping the size of the bursts issued by D_2 *constant* and equal to 16 words. Both the experiments have been repeated for 10000 runs for each configuration, collecting the response times for the DMA under analysis (D_2) and for the interfering DMAs (D_1 and D_3). Thanks to the predictability of the hardware platform, the variability of the measurements resulted very low (in the order of the 1%),

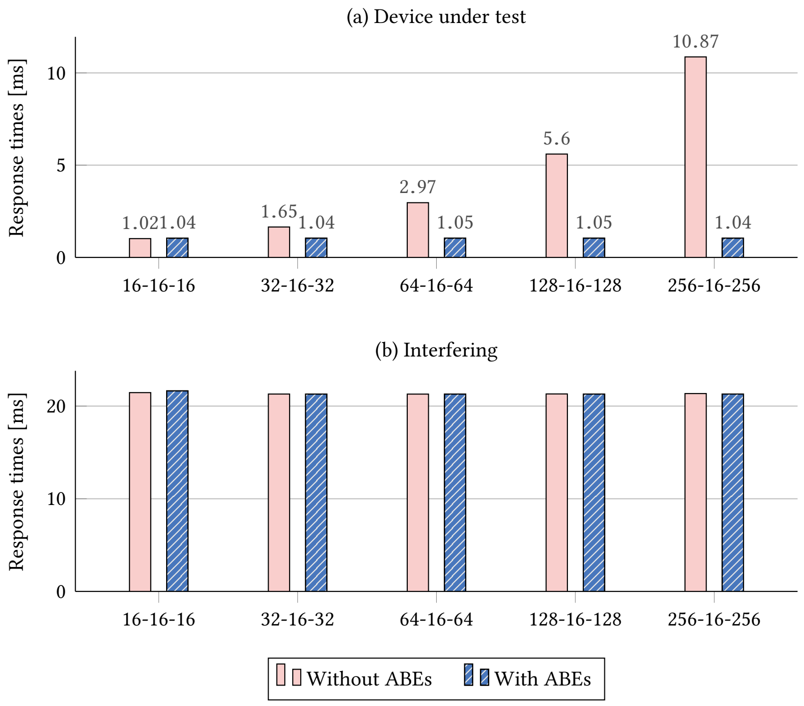

Fig. 10. Response times for the case study.

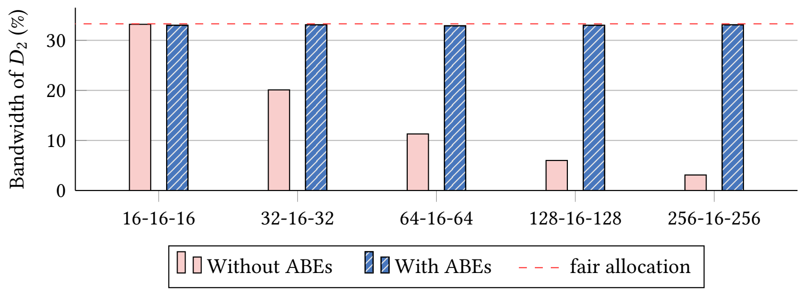

Fig. 11. Percentage of the total bandwidth used by D_2 as a function of the burst size of D_1 and D_3. The labels below each pair of bars indicate the burst size of D_1, D_2 and D_3, respectively.

therefore only the maximum times are reported in the paper. Figure 10 reports the response times of the three DMAs (the one under test in inset (a), and the others in inset (b)) as a function of their configuration. Both the results with the ABEs (blue bars) and without (red bars) are reported. When the ABEs are not installed, the response time of D_2 (under test) increases with the burst size of the interfering DMAs, reaching a maximum of 10.87 ms when D_1 and D_3 issue 256-word bursts. In the latter case, the response time is more than 10 times than the one under a fair bandwidth allocation. Conversely, when the ABEs are installed, the response time of D_2 remains the same, interdependently of the burst size of the interfering DMAs D_1 and D_3 (see the blue bars in Fig. 10(a)). Figure 10(b) reports the response times of the interfering DMAs D_1 and D_3. Since these two DMAs have the same functionality and configuration, under our experimental setting their response times never differ: hence, a single graph is reported. It is worth observing that, since D_1 and D_3 move an amount of data one order of magnitude larger than the one moved by D_2, the interference suffered by D_1 and D_3 due to D_2 is limited. Therefore, their response times are almost the same with and without ABEs. This result also confirms that the latency introduced by the ABE is marginal.

The corresponding bus bandwidths of the DMA under test D_2 are reported in Figure 11. As it can be noted from the graph, when ABEs are installed, the bandwidth of D_2 matches the theoretical one under a fair allocation, *independently of the size of the bursts issued by the other two DMAs.*

5.2 Case study

The effectiveness of the ABE has also been validated by means of a practical application comprising two common IPs from the Xilinx IP library: the Fast Fourier Transform (FFT) accelerator IP, and a Finite Impulse Response (FIR) filter IP synthesized by the Xilinx FIR Compiler. The case study is completed by a DMA (as those mentioned in the previous section) that generates interference on the bus. The FIR filter and the FFT accelerator issue burst transactions of 16 and 64 words, respectively. The DMA is configured to issue burst transactions of 256 words. Each execution instance of the FIR filter processes 256 KB of samples, while each execution instance of the FFT accelerator processes a window of 64 K sample words of 32 bits, resulting in a total amount of 256 KB of data. Finally, the DMA moves memory blocks of 1 MB for each execution instance. Note that this experiment confirms the seamless applicability of the ABE to real-world IPs.

In this experimental evaluation, all hardware modules have been periodically and synchronously activated with the same period of 100 ms for 10000 runs. Figure 12 reports the longest observed response times under three configurations: **(i)** each hardware module running *alone* (i.e., the other two are off); **(ii)** the three hardware modules running together (i.e., they experience contention on the bus) without ABEs; and **(iii)** the same of (ii) but with ABEs. Average and minimum values are not reported as they differ from the maximum values by less than 1%. From a profiling of the behavior of the DMA and the FIR modules when running alone, it emerged that they move one word of data at almost every clock cycle. Their response times in isolation are consistent with this observation: indeed by dividing the amount of data moved by the modules by their response time the obtained memory bandwidth is very close to the one documented for the target platform (about 380 MB/s). Conversely, the FFT interleaves memory transactions with computation stages, and when running alone is able to utilize about half of the bus bandwidth (256 KB/1.32 ms $\simeq$ 194 MB/s).

When running together all the three modules, the FIR filter exhibits the largest response time, as it is penalized by the fact that it issues burst transactions with the smallest size (with respect to the other two modules). Despite being the more data-eager module and the one that when running alone has the largest response time, the DMA has the shortest response time when running together with the other modules. This is because it issues burst transactions with the largest size. With the ABEs is then possible to restore a fair bandwidth redistribution. First note that the response times with the ABEs follow the same order of the response times when the modules run alone. Being the one with the shortest response time, the FIR module always experiences contention generated by the other two modules. Dividing the amount of data moved by the FIR module (256 KB) by the corresponding response time (2.07) it results that it uses 1/3 of the total bandwidth (about 124 MB/s), which corresponds to the expected fair share. A similar analysis can be performed for the DMA, but taking into account that the amount of contention varies over time, i.e., up to 2.07 ms it receives interference from both the FFT and the FIR modules, from 2.07 ms to 3.27 it receives interference from the FFT module only, and finally it is running alone until it completes. An accurate timing analysis of the modules is omitted due to lack of space.

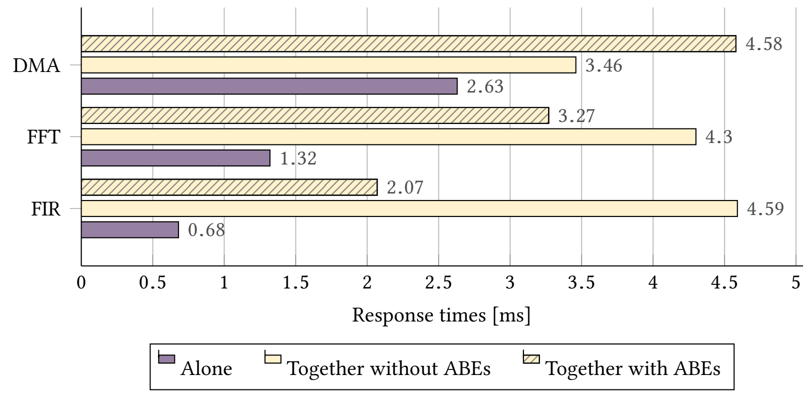

Fig. 12. Comparison of the response times for the case study.

5.3 Resource consumption

The ABE has been packaged as an IP, making easier to integrate it in realistic designs. Table 2 reports the logical resources required by the implementation of an ABE on two state-of-the-art FPGA SoCs, namely the Xilinx Zynq Z-7020 and the Xilinx Zynq ZU9EG UltraScale+. The percentage of resources required by the ABE (with respect to the total numbers available in the platform) is also reported in the table. Furthermore, Table 3 reports the resource consumption of the ABEs used for the case study presented in the previous section, compared with the resource consumption of the whole AXI bus infrastructure deployed on the FPGA fabric and the adopted Xilinx IPs. The results are the same on both the ZYNQ-Z7020 and ZYNQ Ultrascale+ platforms.

The tested implementations have been synthesized using Xilinx Vivado 2018.2. As it can be observed from the tables, the resource utilization of ABE is relatively limited, not only compared with the total amount of resource available on the evaluated platforms, but also compared to the AXI bus programmable logic and the IPs used in the case study.

Table 2. Resource utilization of an ABE.

SoC	Resources			
	LUT	**FF**	**BRAM**	**DSP**
Z-7020	1131/53200 (2.1%)	582/106400 (0.5%)	0/140	0/220
ZU9EG	1167/274080 (0.4%)	486/274080 (0.2%)	0/912	0/2520

Table 3. Resource consumption for all the components in programmable logic used for the case study of Section 5.2 (ZYNQ Z-7020 platform) .

ZYNQ Z-7020	Resources			
	LUT	**FF**	**BRAM**	**DSP**
Xilinx IPs. DMA + FIR + FFT	10935/53200 (20.55%)	15006/106400 (14.1%)	85/140 (60.7%)	51/220 (23.2%)
ABEs (3)	3393/53200 (6.3%)	1746/106400 (1.6%)	0/140	0/220
AXI Bus in FPGA Fabric	16163/53200 (30.4%)	19129/106400 (18.0%)	0/140	0/220

6 RELATED WORK

The dominant architectural solutions for implementing on-chip communication infrastructures for systems on chip (SoC) platforms are mostly based on traditional transaction-based buses or packet-based networks on chip (NoC) [2, 15]. Many research efforts have been dedicated to enhance predictability and improve the performance using NoC architectures [7–9].

In a bus-based SoC architecture, the processing elements are generally bus master units capable of initiating read and write transactions directed to slave memory modules and I/O peripherals. A wide range of schemes to arbitrate master's requests can be conceived, based on well know paradigms like Fixed Priority, Round Robin, and Time-Division Multiple Access (TDMA). In the research community, many efforts have been dedicated to improve throughput and predictability of on-chip interconnect with innovative solutions. Poletti et al. [14] presented a performance analysis of arbitration policies for SoCs platforms. Richardson et al. [15] and Burgio et al. [6] employed TDMA-based schemes with dynamic timeslot allocation to guarantee system predictability and provide good average-case performances. Shah et al. [16] proposed an arbitration scheme combining static priority based arbitration with TDMA. In [4] and [5] the authors propose a predictable bus arbitration scheme designed for multi-core architectures. Lahiri et al. [10] and Lin et al. [11] employed a statistical approach to solve the contention problem, in which the master arbitration is resolved using a ticket-based random selection. Yuan et al. [26] and Sousa et al. [17] presented reconfigurable bus arbiters employing different arbitration schemes depending on the application.

However, despite considerable research efforts, the AMBA AXI bus architecture remains the dominant solution used in FPGA and FPGA-SoC platforms to leverage the availability of IP cores compliant to the AXI standard. The AXI specification [1] does not mandate any specific arbitration protocol for the Interconnect which is left to the implementer. In the Xilinx's ecosystem, the interconnect functionalities are implemented by the *AXI Interconnect* [20] and the more recent *SmartConnect* [24] IPs. Both IPs implement the Round Robin arbitration policy but ignore the burst size while accepting incoming requests. As shown in this paper, this behavior can result in unfair bandwidth allocation when considering multiple masters with different burst sizes.

7 CONCLUSIONS

This paper showed that round-robin arbiters, quite common in state-of-the-art AXI Interconnect used in SoC FPGAs, can lead to a completely unfair bandwidth allocation whenever transactions with heterogeneous burst sizes are issued. Experimental evaluations on multiple state-of-the-art platforms highlighted that the memory bandwidth of a victim hardware accelerator can drop to 9% of the bandwidth expected under a fair allocation. In the general case including an arbitrary number of hardware accelerator, the bus bandwidth of a victim hardware accelerator can be arbitrarily dropped to 0%. The AXI Bus Equalizer (ABE) has been introduced to restore fairness in the bus arbitration. The ABE introduces just one cycle of additional latency, independently of the the burst size of the transactions. The effectiveness of the ABE has been demonstrated with experimental results targeting both traffic generators and hardware accelerators from the Xilinx's IP library. The impact on resource consumption of ABEs has also found to be very marginal (less than the 0.5% on a Zynq Ultrascale+). Future work will aim at realizing a powerful interconnect module that does not suffer of the issue identified in this work, implementing more advanced arbitration policies to better control the memory traffic.

REFERENCES

[1] ARM 2012. *AMBA AXI and ACE Protocol Specification.* ARM.
[2] Luca Benini and Giovanni De Micheli. 2002. Networks on chips: A new SoC paradigm. *IEEE Computer* 35, 1 (2002), 70–78.

[3] Alessandro Biondi, Alessio Balsini, Marco Pagani, Enrico Rossi, Mauro Marinoni, and Giorgio Buttazzo. 2016. A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

[4] Roman Bourgade, Christine Rochange, Marianne De Michiel, and Pascal Sainrat. 2010. MBBA: a multi-bandwidth bus arbiter for hard real-time. In *5th IntâĂŹl Conference on Embedded and Multimedia Computing (EMC)*.

[5] Roman Bourgade, Christine Rochange, and Pascal Sainrat. 2011. Predictable bus arbitration schemes for heterogeneous time-critical workloads running on multicore processors. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*. IEEE, 1–4.

[6] Paolo Burgio, Martino Ruggiero, Francesco Esposito, Mauro Marinoni, Giorgio Buttazzo, and Luca Benini. 2010. Adaptive TDMA bus allocation and elastic scheduling: A unified approach for enhancing robustness in multi-core RT systems. In *Computer Design (ICCD), 2010 IEEE International Conference on*. IEEE, 187–194.

[7] Jason Cong, Michael Gill, Yuchen Hao, Glenn Reinman, and Bo Yuan. 2015. On-chip interconnection network for accelerator-rich architectures. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 8.

[8] Abbas Eslami Kiasari, Zhonghai Lu, and Axel Jantsch. 2013. An analytical latency model for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21, 1 (2013), 113–123.

[9] Akash Kumar, Andreas Hansson, Jos Huisken, and Henk Corporaal. 2007. An FPGA design flow for reconfigurable network-based multi-processor systems on chip. In *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1–6.

[10] Kanishka Lahiri, Anand Raghunathan, and Ganesh Lakshminarayana. 2006. The LOTTERYBUS on-chip communication architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 6 (2006), 596–608.

[11] Bu-Ching Lin, Geeng-Wei Lee, Juinn-Dar Huang, and Jing-Yang Jou. 2007. A precise bandwidth control arbitration algorithm for hard real-time SoC buses. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*. IEEE Computer Society, 165–170.

[12] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, et al. 2016. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (2016), 1591–1604.

[13] Marco Pagani, Enrico Rossi, Alessandro Biondi, Mauro Marinoni, Giuseppe Lipari, and Giorgio Buttazzo. 2019. A Bandwidth Reservation Mechanism for AXI-Based Hardware Accelerators on FPGAs. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, Sophie Quinton (Ed.), Vol. 133. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 24:1–24:24. https://doi.org/10.4230/LIPIcs.ECRTS.2019.24

[14] Francesco Poletti, Davide Bertozzi, Luca Benini, and Alessandro Bogliolo. 2003. Performance analysis of arbitration policies for SoC communication architectures. *Design Automation for Embedded Systems* 8, 2-3 (2003), 189–210.

[15] Thomas D Richardson, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Yuan Xie, Chita Das, and Vijay Degalahal. 2006. A hybrid SoC interconnect with dynamic TDMA-based transaction-less buses and on-chip networks. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*. IEEE, 8–pp.

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

[17] Éricles Sousa, Deepak Gangadharan, Frank Hannig, and Juergen Teich. 2014. Runtime reconfigurable bus arbitration for concurrent applications on heterogeneous MPSoC architectures. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 74–81.

[18] Xilinx 2016. *Zynq-7000 All Programmable SoC - Reference Manual*. Xilinx. UG585.

[19] Xilinx 2017. *Zynq UltraScale+ Device - Reference Manual*. Xilinx. UG1085.

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

[21] Xilinx 2018. *Convolutional Encoder, LogiCORE IP Product Guide*. Xilinx. PG026.

[22] Xilinx 2018. *Fast Fourier Transform, LogiCORE IP Product Guide*. Xilinx. PG109.

[23] Xilinx 2018. *FIR Compiler, LogiCORE IP Product Guide*. Xilinx. PG149.

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

[25] Xilinx 2018. *Versal: Adaptive Compute Acceleration Platform*. Xilinx. WP505.

[26] Ching-Chien Yuan, Yu-Jung Huang, Shih-Jhe Lin, and Kai-hsiang Huang. 2008. A reconfigurable arbiter for SOC applications. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*. IEEE, 713–716.

[27] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. 2016. Memory bandwidth management for efficient performance isolation in multi-core platforms. *IEEE Trans. Comput.* 65, 2 (2016), 562–576.