Cut and Forward: Safe and Secure Communication for FPGA System on Chips

Francesco Restuccia and Ryan Kastner
University of California San Diego

***Abstract*—Modern FPGA system on chips use complex multi-manager, multi-subordinate on-chip communication networks. Processor cores, hardware accelerators, DMA engines, and other manager components actively access subordinate components like off-chip DRAM, on-chip memories, caches, and I/Os. On-chip communication networks are designed for high bandwidth and low latency. They use simple, fast transactions that largely assumes the managers cooperate. For example, it does not describe default mechanisms to ensure the safe behaviors of the managers using the on-chip interconnect. This lack of specification can lead to unpredictable behaviors: a single misbehaving, misconfigured, or malicious component can cause denial of service of shared resources. Clearly, this issue is critical in systems with safety and security constraints. Cut and Forward is a novel switching method for multi-component communication architectures on FPGA SoCs. Cut and Forward leverages the programmability of FPGA SoCs to enable safe and secure bus access and is carefully designed to minimize its impact on performance and resource usage. Experiments show that Cut and Forward ensures safety and security in realistic applications deployed on a commercial FPGA SoC from Xilinx including a popular DNN accelerator.**

***Index Terms*—On-chip communication, mission-critical systems, hardware security, FPGA System-on-Chip.**

I. Introduction

FPGA Systems on Chips (SoCs) are popular platforms for applications with complex energy, performance, real-time, and safety constraints [1]–[5]. Safety- and security-critical systems must undergo a rigorous verification process. Any failure, misbehaviour, or a security attack can cause dramatic consequences if not properly managed. Timing predictability is crucial in safety- and security-critical systems. This requires that a processing unit executes a critical task within a deadline [6]. Other safety and security requirements involve isolation of resources [7], noninterference [8], and authentication [9].

FPGA SoCs commonly use high-throughput and low-latency on-chip protocols to communicate between the various components of the system. Processors, accelerators, hardware root of trusts, and other hardware components act as *managers* to initiate transactions. Off-chip DRAM, on-chip memories, caches, and I/Os work as *subordinates* to deliver data requested by the managers. AMBA AXI [10] is a common on-chip communication protocol, which we use as our example throughout this article. However, the techniques are generally applicable to other on-chip communication protocols.

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2022 and appears as part of the ESWEEK-TCAD special issue.

AXI provides flexible and high-throughput bus transfers. *However, it does not provide a standard mechanism to supervise the behaviors (or misbehaviors) of the managers.* This lack of specification, combined with the standard switching method typically deployed in on-chip communication, permits *a misbehaving component to delay or deny access to a shared resource from all the other components*. Misbehaving/malicious components can exploit this lack of specification to perform a *denial of service (DoS)* of a critical component to the shared resource. This is particularly critical in safety- and security-critical systems, e.g., a DoS could stop a control algorithm from processing data coming from a critical sensor. Moreover, misbehaving components can cause unpredictable behaviors in the bus components (e.g., interconnects and buffers) within the processing system and the FPGA fabric. This would require the system to deploy challenging detection, mitigation, and restore features. However, such features must be compatible with the requirements of the safety- and security-critical system. For example, resetting a component is generally not permitted in safety-critical systems dealing with timing constraints.

The threat becomes even more significant in systems integrating third-party, closed-source IP cores and other components. Trusting the IP vendor may not be feasible; IP cores could implement malicious functionalities exploiting this lack of specification to perform a denial of service attack [11]–[13]. Moreover, it may be an unintentional flaw of the hardware component, e.g., caused by misconfiguration, improper access to the bus, or hardware design error.

Understanding whether a component is safely and securely integrated requires a joint verification of the entire integrated system, involving the hardware accelerators, the interconnect, and any other system components [14]. Some solutions have been proposed to prevent this issue [15], [16]. Unfortunately, they have key limitations for system integration: they lack flexibility or have a strong impact on the system performance. Such limitations make it impractical for mixed-critical applications, leaving them potentially affected by delays or denial of service caused by misbehaving, faulty, or malicious hardware components.

Contribution: Cut and Forward is a customizable switching methodology targeting safety and security for FPGA SoCs. Cut and Forward uses intelligent buffering to maintain operation even in the presence of one or more misbehaving or malicious hardware components. Cut and forward is easily integrated into the AXI interconnect in place of the standard cut-through buffers; no other modifications are required to the components or the underlying FPGA SoC platform. This work makes the following contributions:

- Identifying the sources of stalls in AXI-based FPGA SoCs and the limitations of existing solutions in addressing them (Section II).
- Creating Cut and Forward: a parameterizable switching methodology targeting safety and security (Section III).
- Analyzing the area and performance impact of Cut and Forward and the other similar methods (Section IV).
- Developing a straightforward method to implement Cut and Forward buffers while meeting performance and area requirements of a target application (Section V).
- Demonstrating the applicability and the advantages of Cut and Forward on a commercial FPGA SoC platform and deploying a realistic mixed-critical application involving a commercial deep neural network (DNN) accelerator (Section VI).

The rest of the paper is organized as follows: Section II introduces the threat model, identifies the source of AXI stalls, and examines the limitations of existing solutions. Section III describes Cut and Forward switching and defines its behavior on bus transactions. Section IV analytically compares the impact on performance and area consumption of Cut and Forward with respect to other available switching methods. Section V proposes a method based on the results proposed in Section IV to target the Cut and Forward buffers on the performance and area requirements of a target application. Section VI reports the experimental results, evaluated on realistic, multi-accelerator architectures deployed on commercial FPGA SoC platforms from Xilinx including a popular hardware component for DNN acceleration. The paper ends with related works (Section VII) and conclusions (Section VIII).

II. Motivation and Background

This section describes the threat model and shows how misbehaving hardware components can sabotage on-chip communications. We identify the core behaviors causing the stalls. And we discuss the limitations of available solutions in addressing this potential safety and security vulnerability.

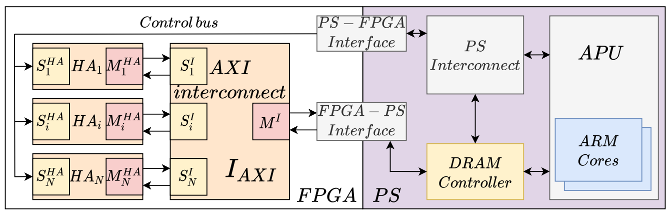


Fig. 1. A sample multi-accelerators architecture implemented on an FPGA SoC platform.

A. Threat Model

We consider the AMBA AXI protocol for on-chip communications [10]. AXI is deployed in many of the modern commercial FPGA SoC platforms. We consider a generic SoC FPGA architecture that models many the modern FPGA SoC commercial architectures [17]–[19]; We consider the default commercial AXI interconnect for the platform under analysis and provided by the FPGA vendor [20]. We focus our threat model on the AXI bus stall that is commonly introduced by FPGA SoC components during normal execution. For example, we experimentally validated the presence of AXI stalls during the nominal execution of the Xilinx DPU [21] (a DNN acceleration component on Xilinx FPGA SoC platforms) and of the Xilinx central DMA controller [22] (a standard module for data movement from/to the programmable logic). This paper focuses on providing a solution to solve the issue of AXI stalls presented in Section II. It is worth mentioning that previous works focused on solving different issues on the same platforms related to access control [23] and unfair bandwidth distribution [24]. We assume that such solutions are correctly integrated in the platforms to mitigate these problems. All of these solutions are compatible with Cut and Forward proposed in this paper.

The threat model assumes that the hardware components are fully compliant with the AXI standard. We do not require any internal knowledge about the structure of the hardware components. Therefore, the proposed threat model holds both for hardware components developed in-house (for which some internal knowledge may be available to the system integrator) and outsourced, third-party hardware components (e.g., closed-source encrypted modules).

The hardware components are connected to the *processing system (PS)*, and therefore, the DRAM memory controller, through an AXI-compliant interconnect. It is assumed that the routing functionalities of the AXI interconnect, processing system, memory controllers, and memories are trustworthy and implemented correctly. It is assumed that an access control system is in place and correctly configured to restrict the access of the hardware accelerators to the only allowed memory regions [23]. We do not make any specific assumption on the technology deployed for implementing the access control system. Therefore, this can be implemented in the AXI interconnect [20], in the peripherals [18], or using custom solutions. We assume that the access control system is properly configured and its functionalities are trustworthy, i.e., the access control system is able to stop any illegal access attempted by hardware components even when one or more of them are misbehaving or have been compromised.

Nevertheless, the functionalities provided by the access control system are not intended to stop hardware components to propagate stalls on the common bus able that can generate a DoS to shared resources. Hardware components may introduce unintentional stalls on the shared bus, e.g., caused by the internal structure of the specific hardware accelerator, by the structure of the issued AXI transactions (i.e., maximum number of outstanding transactions, burst lengths, etc.), or by a combination of these. Such stalls may be unintentionally introduced by high-performance hardware components leveraging speculative bus access (e.g., booking the bus for writing data before they are ready to be propagated). We experimentally evaluated that the Xilinx DPU hardware accelerator [21] using the Vitis AI framework for acceleration of deep neural networks on Xilinx FPGA SoC platforms [25] introduces such stalls. However, the lack of specification of stalls on

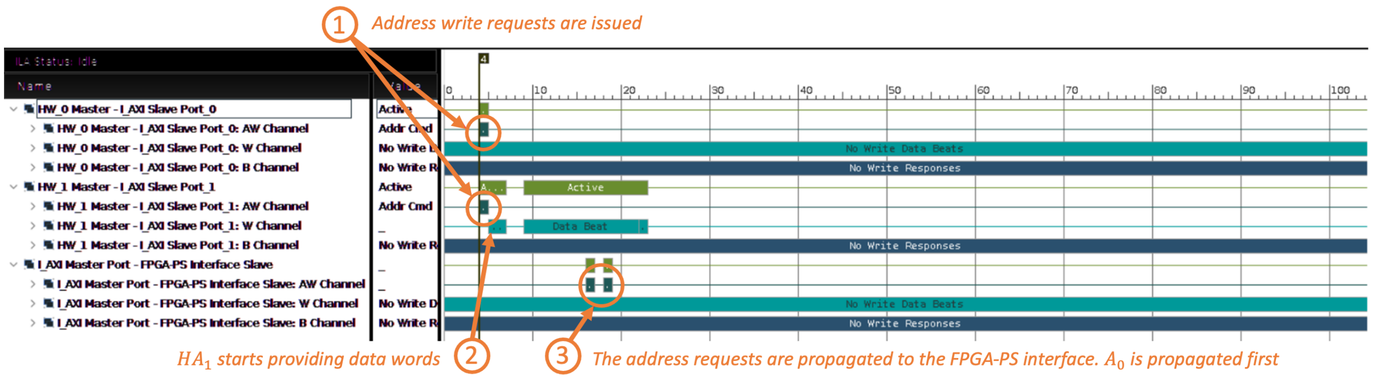

1 Address write requests are issued
ILA Status: Idle
Name
Value
HW_0 Master - I_AXI Slave Port_0
HW_0 Master - I_AXI Slave Port_0: AW Channel
HW_0 Master - I_AXI Slave Port_0: W Channel
HW_0 Master - I_AXI Slave Port_0: B Channel
HW_1 Master - I_AXI Slave Port_1
HW_1 Master - I_AXI Slave Port_1: AW Channel
HW_1 Master - I_AXI Slave Port_1: W Channel
HW_1 Master - I_AXI Slave Port_1: B Channel
I_AXI Master Port - FPGA-PS Interface Slave
I_AXI Master Port - FPGA-PS Interface Slave: AW Channel
I_AXI Master Port - FPGA-PS Interface Slave: W Channel
I_AXI Master Port - FPGA-PS Interface Slave: B Channel
Active
Addr Cmd
No Write D
No Write R
No Write Data Beats
No Write Responses
Active
Data Beat
HA_1 starts providing data words 2
3 The address requests are propagated to the FPGA-PS interface. A_0 is propagated first

Fig. 2. ILA waveform track for the test-case under analysis. HA_0 delays the provisioning of the data after booking the shared bus to the DRAM memory acting a denial of service for HA_1.

the common bus introduces a mechanism that, if properly exploited, enables a malicious/compromised hardware component to perform a DoC of one or multiple shared components from all of the other hardware components in the system, thus compromising the system availability.

From the previous considerations, the security threats addressed in this work relate to the availability of the shared resources, e.g., a single faulty/malicious/misbehaving hardware component can introduce a stall on the shared bus causing a DoS attack of one or multiple shared resources (e.g., memories and peripherals).

B. AXI Stalls on FPGA SoC Platforms

Figure 1 illustrates a typical hardware accelerator architecture implemented on a FPGA SoC platform. On the left is the *programmable logic (PL)* where the hardware accelerator components reside. The Processing System (PS) is on the right. *N hardware accelerators (HAs)* are connected to one of the subordinate ports available at the *FPGA-PS interface*, which is typically used to reach a shared DRAM memory controller (MC) placed in the PS. An *AXI interconnect* I_{AXI} arbitrates the requests issued by the HAs. Conflicts are solved using round-robin arbitration. The generic manager hardware accelerator HA_i is connected to one of the N subordinate input interfaces of the AXI interconnect S_i^I. The output manager port of I_{AXI} is connected to the FPGA-PS interface.

To demonstrate the bus stalling issue, we deployed multi-accelerators architectures on two FPGA-based SoCs: Xilinx ZYNQ-7000 and ZYNQ Ultrascale+. We built the bus architecture using AXI SmartConnect [20], which is the state-of-the-art AXI interconnect for Xilinx platforms. The test architecture is composed of two hardware accelerators HA_0 and HA_1 accessing the DRAM memory through the AXI SmartConnect I_{AXI} (i.e., Figure 1 with $N = 2$). Each hardware accelerator issues a single write request. The following considerations also apply when the hardware accelerators issue multiple transactions. We deployed an *integrated logical analyzer (ILA)* in the FPGA fabric to monitor the manager interfaces of each hardware accelerator and the manager interface of I_{AXI}. Figure 2 shows the recorded ILA track for ZYNQ Ultrascale+. We observed similar results on the ZYNQ-7000, which are not reported for brevity. The relevant transactions are:

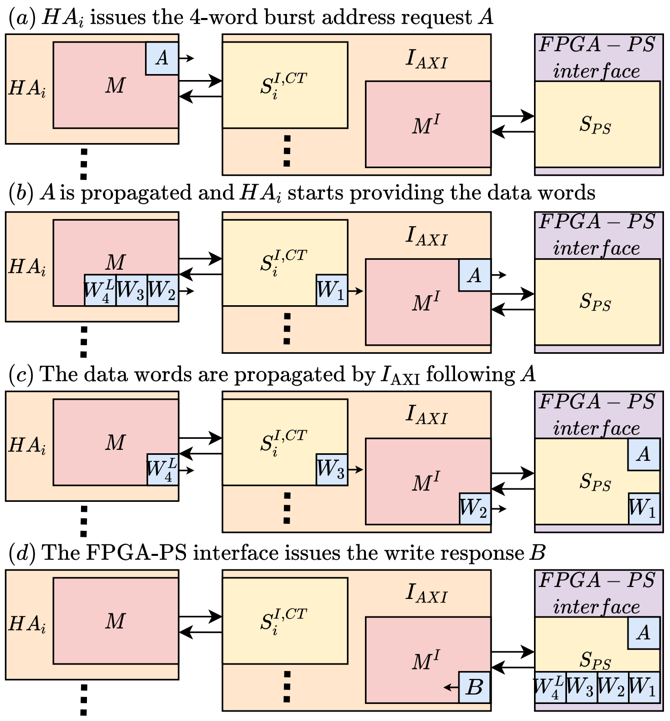
(1) HA_0 issues its write request A_0. A_0 is sampled by I_{AXI}. Similarly, HA_1 issues its write request A_1.
(2) HA_1 starts providing the data W_1 (corresponding to A_1) to I_{AXI}. HA_0 delays the provisioning of the corresponding data W_0 to I_{AXI}.
(3) I_{AXI} propagates the requests A_0 and A_1 to the FPGA-PS interface. The round-robin arbitration is won by A_0, i.e., the round-robin arbiter decides to propagate the write request A_0 before A_1. The data corresponding to A_1 has already been sampled by I_{AXI}. However, interleaving of write transactions is forbidden [26]. Thus, even if already buffered, the data corresponding to A_1 is not propagated by I_{AXI} to the FPGA-PS interface *until the data corresponding to A_0 is sampled by I_{AXI} and propagated to the FPGA-PS interface.*

By delaying data provisioning, HA_0 can directly affect the availability of the output port M^I of I_{AXI} to the other hardware accelerators in the system. The delay is fully compliant with the standard specification – according to AXI, HA_0 is free to delay the data provision for an unbounded time. Thus, HA_1 cannot access the data write channel until the data corresponding to A_0 is provided by HA_0.

C. The Source of Bus Stalls: Cut-Through Switching

Many commercial AXI interconnects, including the Xilinx AXI SmartConnect [20], use *cut-through switching (CT)*. In the following, we describe the behavior of I_{AXI} when implementing CT subordinate buffers. Subsequently, we describe how cut-through switching fails in properly handling stalls generated by hardware accelerators.

1) Sample Write Transaction: Figure 3 reports a schematic representation of the behavior of a generic buffer implementing cut-through switching ($S_i^{I,CT}$) on a sample AXI write transaction. For simplicity, only the generic hardware accelerator under evaluation HA_i is reported. A multi-accelerator architecture will have N hardware accelerators, as in Figure 1. The description is as follows: **(a)** HA_i starts the write transaction issuing the address request A. In this example, A is a four-word burst request. **(b)** A is sampled by $S_i^{I,CT}$ and immediately routed to M^I. HA_i starts providing the data words corresponding to A. The data words $W = W_1, \ldots, W_4^L$



(a) HA_i issues the 4-word burst address request A
(b) A is propagated and HA_i starts providing the data words
(c) The data words are propagated by I_{AXI} following A
(d) The FPGA-PS interface issues the write response B
HA_i
M
$S_i^{I,CT}$
I_{AXI}
M^I
$FPGA-PS$ $interface$
S_{PS}
A
B
W_4^L W_3 W_2 W_1

Fig. 3. Sample write transaction under cut-through switching.

are consecutively sampled by $S_i^{I,CT}$. The last word of the burst is marked by HA_i as *last (L)* word, hence named W_4^L. **(c)** W are routed through the network following the address request A, until the last word W_4^L is sampled by the FPGA-PS interface S_{PS}. **(d)** After a service delay, S_{PS} replies with a write response B to acknowledge HA_i about the completion of the transaction. B is routed to HA_i by I_{AXI}. The write transaction is completed.

2) The Stalling Problem: In step **(b)**, $S_i^{I,CT}$ propagates A to M^I regardless of the provisioning of the corresponding data words W by HA_i. According to the AXI standard, the propagation of A to M^I books the M^I's write channel to HA_i, which should provide the write data W corresponding to A. However, the AXI standard does not specify any timing constraint on HA_i to provide W after A has been propagated to M^I. HA_i can arbitrarily delay the provisioning of W! M^I is the shared port among all the hardware accelerators in the system to access the shared subordinate resource (the memory). Therefore, as also demonstrated in the real example of Section II-C, once A is propagated to M^I and HA_i does not provide W, M^I is stalled and no other hardware accelerators in the system can access it. This means that the cut-through buffer $S_i^{I,CT}$ implicitly trusts HA_i that once A is propagated to M^I, HA_i will complete the transaction as soon as possible providing W and then leaving the access of M^I to the other hardware accelerators in the system.

Therefore, a single malicious/misbehaving hardware accelerator delaying the provisioning of the data can delay the access to the shared resource from the other hardware accelerators in the system, endangering the availability of the shared memory and all of the other shared peripherals of the PS subsystem, thus generating a DoS for all the other hardware accelerators in the system. At this point, since AXI requests cannot be aborted, the only method to restore a safe condition is a full reset of the network. However, a full reset can have dramatic consequences on a safety-critical system and is generally not acceptable.

D. Existing Solutions and Their Limitations

The deployed switching method has a critical impact on the safety and security of the system. Two methods are available to prevent the bus stalling issue. We briefly introduce them in the following.

1) AXI Stall Monitor: The *AXI Stall Monitor (ASM)* monitors the behavior of a supervised HA and intervenes when the HA stall endangers the schedulability of the system [15]. In such scenarios, ASM aborts the execution of the stalling hardware accelerator, safely solves the stall condition, and makes the shared bus available to the other hardware accelerators in the system. ASMs are configured leveraging a worst-case analysis applying only to fully disclosed and strictly periodic bus workloads.

2) Store and Forward: Store-and-forward switching (S&F) is a conservative switching method alternative to the cut-through switching [16]. Store and forward stores address write requests and waits to propagate them (i.e., booking the bus) until all of the corresponding data words have been received by the interconnect. Thus, each request and the corresponding data can be consequently propagated to M^I without any stalls.

Limitations: Due to the nature of its worst-case analysis, ASM cannot be applied to applications generating non-periodic bus workloads. This means that ASM is not applicable to popular safety- and security-critical systems. Moreover, experimentally many commercial IPs introduce bus stalls as part of their normal execution using speculative bus accesses [27], [28]. In such cases, ASM could generate a false positive affecting the system performance. On the other hand, while S&F is applicable regardless of the bus workload, it introduces a long, not configurable, buffering phase that impacts the transaction's latency and mandates higher resource consumption to deploy data buffers. This makes it unpalatable for many high-performance applications. The limitations of ASM and store and forward leave applications characterized by non-periodic workload with strict latency and/or area constraints completely uncovered, and therefore potentially affected by DoS attacks. In the following section, we describe the cut-and-forward approach that overcomes these limitations. Section IV provides a complete comparison of Cut and Forward with ASM and S&F.

III. The Cut-and-Forward switching

Cut and Forward (C&F) is a configurable and easy-to-integrate one-to-one substitute for traditional switching methods. Cut and forward enforces safe bus accesses while making no assumption on the workload generated by the hardware components. Moreover, it is configurable to match the performance and resource usage requirements. Cut and forward combines data buffering with a configurable, low-latency chunking machine. It introduces a pipeline between data buffering and data propagation. The cut-and-forward buffer $S_i^{I,\text{C\&F}}$ is characterized by the pipeline parameter C, corresponding to the size of the cut-and-forward data buffer. C affects the performance and resource usage. Thus, it can be tuned to provide a tradeoff between performance and resource utilization to match the requirements of a target application (see Section IV for more details). In the following, we describe the cut-and-forward buffering process. For simplicity, the description is reported considering a single hardware accelerator issuing a single request. Since each hardware accelerator has exclusive access to its cut-and-forward buffer, the same considerations hold when multiple hardware components issue transactions in parallel.

Consider the situation where a hardware accelerator HA_i initiates an address write burst request A for β_A words. The C&F buffer $S_i^{I,\text{C\&F}}$ adheres to the following rules:

Rule 1: A is buffered by $S_i^{I,\text{C\&F}}$.

Rule 2: If $\beta_A \leq C$, $S_i^{I,\text{C\&F}}$ waits to receive all of the data words corresponding to A. Then, it propagates A and W without introducing stalls.

Rule 3: If $\beta_A > C$, $S_i^{I,\text{C\&F}}$ buffers the first C data words $W_1, W_2, \ldots, W_C$ of A before booking M^I. Then, $S_i^{I,\text{C\&F}}$ generates a C-word burst sub-request A_1 and propagates it to M^I. A_1 is immediately followed by $W_1, W_2, \ldots, W_C$. The C-th data word completes A_1 and is marked as *last* by $S_i^{I,\text{C\&F}}$. The data phase of A_1 is completed and M^I is released.

Rule 4: While propagating $W_1, W_2, \ldots, W_C$ to M^I, $S_i^{I,\text{C\&F}}$ frees internal space and starts buffering the next C words of data $W_{C+1}, \ldots, W_{2C}$ corresponding to A. This introduces a pipeline between buffering and propagation phases.

Rule 5: Once $W_1, W_2, \ldots, W_C$ have been propagated by $S_i^{I,\text{C\&F}}$ and the next C words $W_{C+1}, \ldots, W_{2C}$ have been internally buffered, $S_i^{I,\text{C\&F}}$ generates and propagates to M^I a new C-word sub-request A_2. A_2 is immediately followed by the corresponding data words $W_{C+1}, \ldots, W_{2C}$. W_{2C} is marked by $S_i^{I,\text{C\&F}}$ as the last word.

Rule 6: As in Rule 4, while propagating $W_{C+1}, \ldots, W_{2C}$, $S_i^{I,\text{C\&F}}$ starts buffering the next C data words. Rules 4 and 5 are repeated until all of the β_A data words corresponding to A have been propagated. $S_i^{I,\text{C\&F}}$ splits A into $Q = \lceil \beta_A / C \rceil$ transactions $A_1, A_2, \ldots, A_Q$. If the ratio β_A / C is not an integer, the last sub-request is shorter than C words and managed accordingly by $S_i^{I,\text{C\&F}}$.

Rule 7: $S_i^{I,\text{C\&F}}$ merges the Q write responses received at the B channel into a single response. This maintains the transparency to HA_i and the network.

Cut-and-forward buffers are AXI-compliant and fully transparent to the hardware accelerators and network components. Thus, they can be integrated into existing AXI interconnects without requiring any modification to the hardware accelerators nor the underlying FPGA SoC platform.

A. Formal Description of the Cut-and-Forward Switching

This section aims to describe the safety of our approach. We describe all of the possible scenarios a cut-and-forward buffer can face during execution.

Consider a sample hardware accelerator HA_i initiating an address write burst request A for β_A words to the C&F buffer $S_i^{I,\text{C\&F}}$. After the address write request A is sampled and buffered by $S_i^{I,\text{C\&F}}$ (Rule 1), two cases are possible: (i) $\beta_A \leq C$ or (ii) $\beta_A > C$.

In case (i), $C\&F$ works as a store-and-forward buffer (Rule 2). A is buffered internally – the C&F buffer waits to internally buffer all of the β_A words before propagating A. This means that the shared bus M^I is booked only when all of the corresponding β_A data words are internally buffered and ready to be propagated. At that point, the cut-and-forward buffer propagates the request A (booking the shared bus), fulfills the transaction, and leaves the bus for the other HAs in the system. The burden of fulfilling the transaction is removed from HA_i (which is considered untrusted) and given to the cut-and-forward buffer.

In case (ii), a sub-request for C-words is generated. The sub-request is kept internally by the cut-and-forward buffer and not propagated until C words have been internally buffered by $S_i^{I,\text{C\&F}}$ (Rule 3). Following the AXI standard, this means that even if HA_i stalls while providing the C words corresponding to the sub-request, the shared bus M^I has not been booked yet, and therefore the stall is internally managed by the cut-and-forward buffer and not propagated to M^I. After the C words have been provided by HA_i and buffered inside $S_i^{I,\text{C\&F}}$, the subrequest can be completed. Once again, the burden of completing the transaction is removed from HA_i and left to $S_i^{I,\text{C\&F}}$. The data propagation to the shared bus M^I and the sample of the new incoming data provided by the HA (Rule 4) are two separate processes managed by $S_i^{I,\text{C\&F}}$. Thus, they do not influence each other. The following steps (reported in Rule 5 and Rule 6) are performed by $S_i^{I,\text{C\&F}}$ in Rule 4 and demonstrated before. The shared bus is never booked before all of the data corresponding to a sub-request have been sampled internally by $S_i^{I,\text{C\&F}}$ and therefore are ready to be propagated without stalls. Finally the write responses are merged according to the length of the original transaction (Rule 7). This remains in compliance with the AXI protocol with the request issued by HA_i and makes cut and forward completely transparent to both the hardware accelerator and the interconnect.

B. Cut and Forward Example

Figure 4 shows the behavior of the cut-and-forward buffer $S_i^{I,\text{C\&F}}$ on an AXI write transaction. In this example, $S_i^{I,\text{C\&F}}$ is configured with $C = 2$. The transaction proceeds as follows: (a) HA_i issues a four-word burst address write request A. (b) A is internally buffered by $S_i^{I,\text{C\&F}}$. Subsequently, HA_i starts providing the data words $W = W_1, \ldots, W_4^L$ corresponding to A. (c) $S_i^{I,\text{C\&F}}$ has buffered $C = 2$ data words associated with A (W_1 and W_2). $S_i^{I,\text{C\&F}}$ generates a two-word burst subrequest A_1 and propagates it to M^I. M^I is booked for two data words. W_1 and W_2 have already been buffered by $S_i^{I,\text{C\&F}}$ and can be propagated immediately after A_1. W_2 is converted by $S_i^{I,\text{C\&F}}$ into a last data word W_2^L. In parallel with the propagation of W_1 and W_2, $S_i^{I,\text{C\&F}}$ buffers W_3 and W_4^L. (d) W_3 and W_4^L have been buffered into $S_i^{I,\text{C\&F}}$. $S_i^{I,\text{C\&F}}$ generates a second

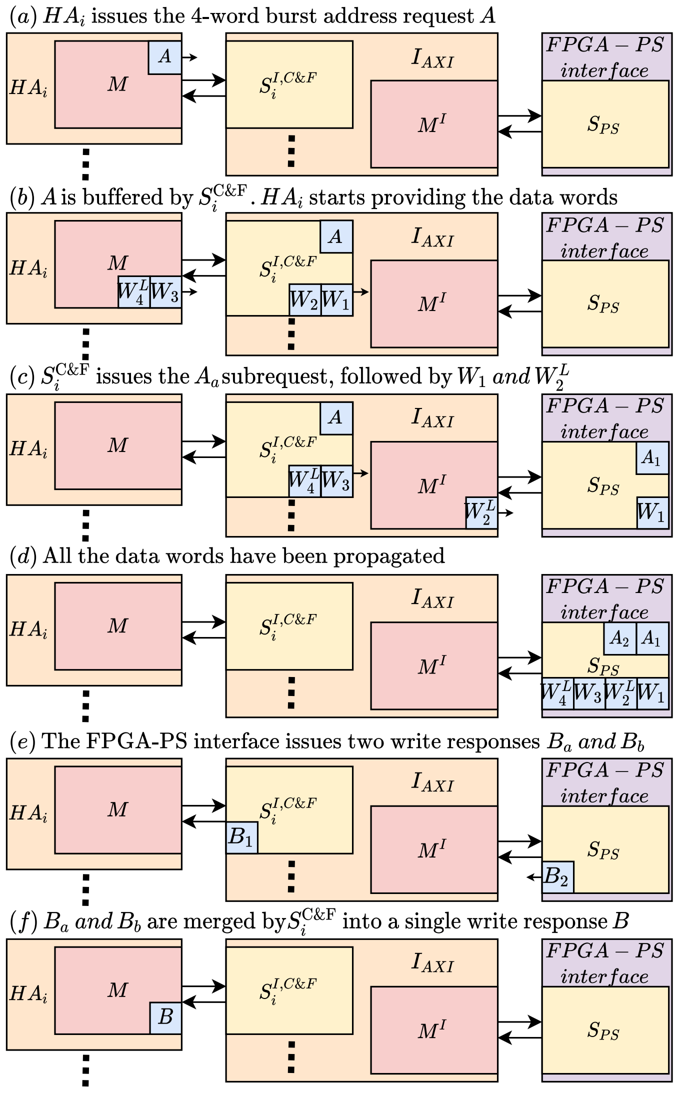

(a) HA_i issues the 4-word burst address request A
(b) A is buffered by $S_i^{\text{C\&F}}$. HA_i starts providing the data words
(c) $S_i^{\text{C\&F}}$ issues the A_a subrequest, followed by W_1 and W_2^L
(d) All the data words have been propagated
(e) The FPGA-PS interface issues two write responses B_a and B_b
(f) B_a and B_b are merged by $S_i^{\text{C\&F}}$ into a single write response B
HA_i
M
$S_i^{I,C\&F}$
I_{AXI}
M^I
$FPGA-PS$ $interface$
S_{PS}

Fig. 4. Sample write transaction under Cut-and-Forward switching.

sub-request A_2 for $C = 2$ data words. The address of A_2 is calculated according to C. A_2 is propagated to M^I. M^I is booked for two data words. W_3 and W_4^L are propagated to M^I immediately after A_2. (e) After a service delay, the FPGA-PS interface replies with the two write responses B_1 and B_2 corresponding to A_1 and A_2, respectively. B_1 is the response for A_1 and it is first one to reach I_{AXI}. $S_i^{I,\text{C\&F}}$ buffers B_1 while waiting for B_2. (f) B_2 has reached $S_i^{I,\text{C\&F}}$. $S_i^{I,\text{C\&F}}$ merges B_1 and B_2 into a single response B for the original request A. B is propagated to HA_i. The write transaction is concluded.

C. Cut and Forward Scalability

For the sake of simplicity, the considerations made in the previous sections focus on a single misbehaving hardware module in the system. In this section, we extend our discussion to the case where multiple manager hardware components in the system concurrently misbehave. Consider the sample architecture reported in Figure 1. Following the AXI standard, each of the manager hardware modules in the system is connected to a private secondary interface at the AXI interconnect. This means that each hardware manager interfaces exclusively with its private buffer integrated into the AXI interconnect. When integrating cut-and-forward buffers in the AXI interconnect, each buffer is independent of the others, meaning that each cut-and-forward buffer manages and contains the stalls introduced by the hardware components is connected to independently of the behavior or state of the other hardware components and cut-and-forward buffers in the system. Thus, the hardware components cannot influence each other before the transactions are validated by the cut-and-forward buffers; multiple stalls are contained in parallel, with no interactions among cut-and-forward buffers. From the previous considerations, our methodology is intrinsically able, with no modifications, to shield against bus stalls even when stalls are introduced concurrently by multiple modules, following the same methodology described in Section III.

IV. Analysis and Features comparison

In this section, we analyze the features of cut-and-forward buffers against the other existing solutions. We leverage this formulation later in Section V to propose a method for configuring the cut-and-forward buffers for a target application.

A. Latency

We start formulating a worst-case analysis bounding the response time of a write transaction under cut-through switching.

1) Cut-Through Switching: Consider the example from Section II-C. A write transaction issued by HA_i begins by issuing an address write request A, which is sampled by I_{AXI}. The hold time for A is t_{A}. In cut-through switching, A is immediately propagated through I_{AXI} after being received. The propagation delay experienced by A is d_I^A. The write data W are sequentially propagated by HA_i with A through I_{AXI}, one word after the other. As W follows A, the propagation latency introduced by I_{AXI} on W is covered by d_I^A. The hold data time for each word of data is t_{W}. The generic burst length of the transaction under analysis is β_i words (any AXI-compliant burst length is admitted). After traversing I_{AXI}, A and W reach the FPGA-PS interface. A and W are then routed to the DRAM memory controller. A write response B is issued by the MC after at most d_{PS}^{DRAM} time units. We opted for a safe coarse-grain model for the MC that considers the limited information available for the MC deployed on commercial platforms [29], [30]). Finally, B is propagated through I_{AXI} to HA_i after a propagation latency d_I^B. The hold time for B is t_B. The memory access time d_W^{CT} for a single write transaction under cut-through switching is:

$$d_W^{CT} = t_{\text{A}} + d_I^A + \beta_i \cdot t_W + d_{PS}^{\text{DRAM}} + t_{\text{B}} + d_I^{\text{B}} \quad \text{cycles} \tag{1}$$

2) General Formulation: The formulation introduced for cut-through switching can be extended to cover S&F and C&F buffers by introducing an extra term accounting for the buffering delay. The memory access time d_W^* of a single write transaction for the generic switching policy has an upper bound of:

$$d_{\text{W}}^{*} = d_{\text{W}}^{CT} + H_{\text{buff}}^{*} \times d_{\text{word}}^{\text{buff}} \qquad \text{cycles} \tag{2}$$

where $d_{\text{word}}^{\text{buff}}$ is the buffering delay for a single data word (generally equal to 1 cycle) and H_{buff}^{*} is the maximum, per-transaction, number of data words required to be buffered. Thus, $d_{\text{buff}}^{*} = H_{\text{buff}}^{*} \times d_{\text{word}}^{\text{buff}}$ is the worst-case, per-transaction, additional buffering latency introduced by the switching policy. While d_{W}^{CT} and $d_{\text{word}}^{\text{buff}}$ are constant, H_{buff}^{*} depends on the specific switching method as follows:

$$H_{\text{buff}}^{\text{CT}} = 0; \quad H_{\text{buff}}^{\text{S\&F}} = 256; \quad H_{\text{buff}}^{\text{C\&F}} = C. \tag{3}$$

Equations 3 show how the buffering phase of S&F introduces a worst-case structural buffering delay $H_{\text{buff}}^{\text{S\&F}}$ equal to the maximum burst length allowed by the AXI standard (256 words). This buffering latency introduced by store and forward can increase more than 90% the write access time with respect to cut-through switching (see Section VI-B). Clearly, store and forward is not a viable solution for applications having low-latency requirements. The pipelining effect introduced by cut-and-forward buffers enables a customizable impact on the maximum buffering latency via the configuration parameter C (see Section V).

B. Resource Consumption

The buffering stage of cut-and-forward and store-and-forward buffers requires a minimum amount of internal buffering slots for data words to work properly. This impacts the area consumption of the AXI interconnect. We propose a simple model for the minimum resource consumption required by a switching method. We start considering generic resources. Then, we target the resources required on FPGA SoC platforms. The amount of required resources r_W^* for the generic single subordinate input buffer $S_i^{I,*}$ is modeled as:

$$r_W^* = r_{\text{logic}} + H_{\text{data}}^{*} \times r_{\text{word}}; \quad \text{generic resources} \tag{4}$$

where r_{logic} is the cost for the logic of the buffer, r_{word} is the resource cost of a single-word data buffer and H_{data}^{*} is the minimum required number of single-word data buffers for the generic solution to work properly. Thus, $H_{\text{data}}^{*} \times r_{\text{word}}$ is the overall resource cost of a generic data buffer. We expand the generic formulation to consider FPGA resources of lookup tables (LUTs) and flip-flops (FFs)[1]:

$$\begin{cases} r_{W,\text{LUT}}^{*} &= r_{\text{logic}}^{\text{LUT}} + H_{\text{data}}^{*} \times r_{\text{word}}^{\text{LUT}} \quad \text{LUTs} \\ r_{W,\text{FF}}^{*} &= r_{\text{logic}}^{\text{FF}} + H_{\text{data}}^{*} \times r_{\text{word}}^{\text{FF}} \quad \text{FFs} \end{cases} \tag{5}$$

where $r_{W,\text{LUT}}^{*}$, $r_{\text{logic}}^{\text{LUT}}$, and $r_{\text{word}}^{\text{LUT}}$ are the LUT costs and $r_{W,\text{FF}}^{*}$, $r_{\text{logic}}^{\text{FF}}$, and $r_{\text{word}}^{\text{FF}}$ are the FF costs. Since CT, S&F, and C&F have similar logic complexity, $r_{\text{logic}}^{\text{LUT}}$ and $r_{\text{logic}}^{\text{FF}}$ are comparable between the switching methods. While $r_{\text{word}}^{\text{LUT}}$ and $r_{\text{word}}^{\text{FF}}$ are independent of the switching method, H_{data} depends on the specific solution:

$$H_{\text{data}}^{\text{CT}} = 0; \quad H_{\text{data}}^{\text{S\&F}} = 256; \quad H_{\text{data}}^{\text{C\&F}} = C; \tag{6}$$

[1]DSPs or BRAM blocks were not required for the implementation.

Approach	Features				
	Safe	Latency impact	Area impact	Contains bounded stalls	Supported workloads
CT	No	Low	Low	No	All
CT+ASM	Yes	Low	Low	No	Periodic
S&F	Yes	High	High	Yes	All
C&F	**Yes**	**Config.**	**Config.**	**Yes**	**All**

TABLE I
SUMMARY OF THE FEATURES COMPARISON.

To comply with the AXI specification, store-and-forward buffers require room for buffering at least one entire maximum length AXI transaction (256 words). In our experimentation, such buffering resources double the resource cost of the interconnect in a three hardware accelerator architecture (see Sections VI-D). Thus, store and forward does not scale well and can be unfeasible for area-constrained applications. The area impact of cut-and-forward buffers depends on the parameter C and is customizable towards the target application (see Section V).

C. Summary of Features

Table I summarize the features of the methodologies. ASM is able to prevent unbounded AXI stalls with no impact on resources and latency. However, as discussed in Section II-D, ASM show three major limitations: 1) It can be applied only to applications characterized by strictly periodic bus workload. 2) The number of transactions issued by each HA (per-period) must be known a-priori; 3) It is not compatible with hardware accelerators that introduce long stalls as part of their normal execution behavior (e.g., for speculative bus access). However, some safety-critical applications, such as mixed-critical, are characterized by mixed or partially unknown bus workload. In such scenarios, store and forward is the only solution available to enforce safe and secure bus access. Nevertheless, the strong impact on response times and resource consumption of store and forward may not be suitable for area-constrained and/or low-latency applications.

Cut and forward has a configurable impact on area and performance. Moreover, cut and forward makes the shared bus M^I independent from the behavior/misbehavior of the hardware accelerators, shielding M^I from any kind of stall (bounded or unbounded) without requiring any assumption on the bus workload.

V. Configuring Cut-and-Forward Buffers

This section proposes a straightforward method to find the configuration parameter C for cut-and-forward buffers. To be as general as possible, we suppose a typical mixed-critical scenario where a subset $\Gamma_{\text{RT}} = \{HA_1^{\text{RT}}, \ldots, HA_P^{\text{RT}}\}$ of P hardware accelerators are periodic and critical, and therefore their execution must be guaranteed within strict timing constraints. We leverage such timing constraints to parameterize the cut-and-forward buffers. The other $R = N - P$ hardware accelerators are non-critical and execute using a best-effort approach. The inputs considered for the analysis are:

1) Timing constraints for the write transactions: the $HA_i^{\text{RT}} \in \Gamma_{\text{RT}}$ must be able to complete a write transaction within a deadline D_i^W.

2) The overall resource available to implement the N subordinate C&F buffers on the platform R_{overall}, split in $R^{\text{LUT}}_{\text{overall}}$ and $R^{\text{FF}}_{\text{overall}}$.

For the sake of simplicity, we assume that all the cut-and-forward buffers implemented in I_{AXI} share the same parameter C.

A. Analysis

As the first step, we evaluate the maximum response time of a single write transaction A_i issued by $HA_i^{RT} \in \Gamma_{\text{RT}}$, considering the worst-case interference generated by the other hardware components in the system. Due to the round-robin arbitration implemented in I_{AXI}, in the worst-case scenario, A_i is delayed by $N-1$ interfering address requests, each issued by one of the $N-1$ interfering hardware accelerators and propagated before A_i by I_{AXI} [30]. Since we are considering C&F buffers, no stalls can happen on M^I. Also, the buffering time of interfering transactions does not delay A_i. From the point of view of HA_i^{RT}, interfering transactions can delay A_i only after they reach M^I. This means that the buffering time of interfering transactions is only experienced by the issuer HA, and not by HA_i^{RT}). Thus, each of the $N-1$ interfering requests can delay the service of A_i by at most d_W^{CT} cycles. Summing up the contributions from Equations 2 and 3, the worst-case overall response time for A_i is equal to

$$\delta_W^{\text{C\&F}} = (N-1) \times d_W^{\text{CT}} + d_W^{\text{C\&F}} = N \times d_W^{\text{CT}} + H_{\text{buff}}^{\text{C\&F}} \times d_{\text{word}} \quad (7)$$

where $(N-1) \times d_W^{\text{CT}}$ is the worst-case interference time, while $d_W^{\text{C\&F}}$ is the worst-case service time for A_i. The round-robin arbitration of I_{AXI} makes all the hardware components in the system experience the same worst-case interference. Therefore, $\delta_W^{\text{C\&F}}$ bounds the response time of a generic write transaction issued by any component. Consider now that the most critical componet is the one having the strictest deadline, equal to $D^W = \min_{HA_i \in \Gamma_{\text{RT}}} D_i^W$. Therefore, if $\delta_W^{\text{C\&F}} \leq D^W$ the deadline is guaranteed for any component in the system. Considering now the area requirements, each cut-and-forward buffer requires the same amount of resources (C is shared). Thus, R_{overall} is equally split among the N buffers. The single cut-and-forward buffer $S_i^{I,\text{C\&F}}$ can occupy at most $R^{\text{LUT}}_{\text{max}} = R^{\text{LUT}}_{\text{overall}}/N$ LUTs and $R^{\text{FF}}_{\text{max}} = R^{\text{FF}}_{\text{overall}}/N$ FFs. From the previous considerations, Equation 7, and Equation 5, we derive the following system ($H^{\text{C\&F}}_{\text{buff}} = H^{\text{C\&F}}_{\text{data}} = C$):

$$\begin{cases} \delta_W^{\text{C\&F}} = N \times d_W^{\text{CT}} + C \times d_{\text{word}} & \leq D^W; \\ r^{\text{C\&F}}_{W,\text{LUT}} = r^{\text{LUT}}_{\text{logic}} + C \times r^{\text{LUT}}_{\text{word}} & \leq R^{\text{LUT}}_{\text{max}}; \\ r^{\text{C\&F}}_{W,\text{FF}} = r^{\text{FF}}_{\text{logic}} + C \times r^{\text{FF}}_{\text{word}} & \leq R^{\text{FF}}_{\text{max}}; \end{cases} \quad (8)$$

Isolating C in the equations and considering that $C \in \mathbb{N}_{>0}$:

$$\begin{cases} C \leq \left\lfloor \frac{D^W - N \times d_{\text{W}}^{CT}}{d^{\text{buff}}_{\text{word}}} \right\rfloor \\ C \leq \left\lfloor \frac{R^{\text{LUT}}_{\text{max}} - r^{\text{LUT}}_{\text{logic}}}{r^{\text{LUT}}_{\text{word}}} \right\rfloor \\ C \leq \left\lfloor \frac{R^{\text{FF}}_{\text{max}} - r^{\text{FF}}_{\text{logic}}}{r^{\text{FF}}_{\text{word}}} \right\rfloor \end{cases} \quad (9)$$

System 9 has multiple solutions. To find the best value of C, we note that DRAM memory controllers can implement policies favoring long and consecutive memory accesses aiming at optimizing the average throughput in high-performance applications [18], [19]. The chunking employed by cut and forward can interfere with such policies. The lower the value of C, the higher the potential impact on average performance. The solution minimizing such an effect is the maximum value satisfying System 9 is

$$C = \min\left(\left\lfloor \frac{D_W^{\max} - N \times d_{\text{W}}^{CT}}{d^{\text{buff}}_{\text{word}}} \right\rfloor, \left\lfloor \frac{R^{\text{LUT}}_{\max} - r^{\text{LUT}}_{\text{logic}}}{r^{\text{LUT}}_{\text{word}}} \right\rfloor, \left\lfloor \frac{R^{\text{FF}}_{\max} - r^{\text{FF}}_{\text{logic}}}{r^{\text{FF}}_{\text{word}}} \right\rfloor \right) \quad (10)$$

The interaction with average throughput is platform dependent, conditioned on the internals of the memory controller. Such details are unfortunately not generally disclosed by vendors. We experimentally analyzed the impact on average performance for the platforms under analysis in Section VI-C. Given our focus on critical systems, for our purposes a limited impact on average performance is an acceptable tradeoff if balanced with the safety and security features provided by cut and forward. We note that, in other scenarios, it is the responsibility of the system integrator to evaluate the best tradeoff for the application.

VI. Experimental Validation

The performance of cut and forward is evaluated on realistic multi-accelerator architectures deployed on commercial FPGA SoC platforms. The architectures involve multiple hardware accelerators sharing the main DRAM memory in the processing system. It is worth noting that same considerations apply to other popular memory configurations (e.g., accessing an external DRAM memory through the MIG IP in the FPGA [31]) as long as two or more hardware accelerators share the bus for the access. Such shared-bus configurations are popular in multi-accelerator architectures as typically peripherals and MCs export a single [31] or limited [18], [19] access ports. We developed a cut-and-forward buffer compliant with the rules described in Section III in VHDL. Subsequently, we integrated cut-and-forward buffers in the AXI HyperConnect (AXI HC) – an open-architecture AXI interconnect for safety-critical systems [32]. We deployed multiple setups to evaluate the effectiveness, performance, and area consumption of cut and forward on designs running on Zynq Ultrascale+ (ZCU102) and Zynq Z-7020 (PYNQ) platforms. The setups include commercial hardware accelerators, such as DMAs, an FIR filter, and a popular hardware accelerator for DNNs. Such setups perform different memory access patterns to evaluate the effectiveness of our solution on multiple AXI-compliant hardware accelerators. The results are compared against ASM [15], cut through, and store and forward.

A. Buffering Latency

This experiment compares the buffering latency experienced by hardware accelerators. The architecture is populated with three Xilinx DMA IPs [28], each performing a single write transaction to the DRAM (see Figure 1 with $N = 3$). We made this choice since DMAs are programmable and are able to mimic the behavior on the bus of any generic hardware

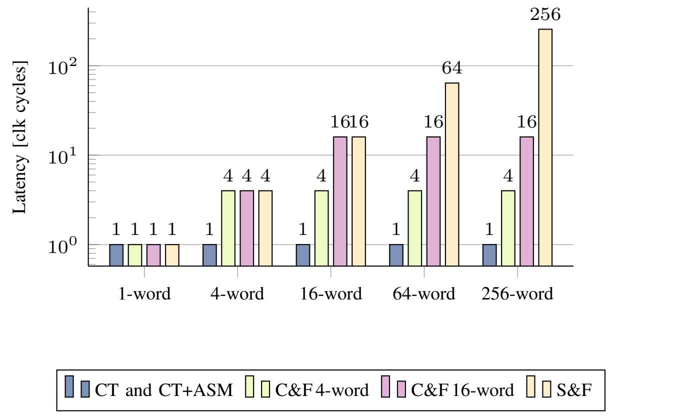


Fig. 5. Additional buffering latency on a single write transaction as a function of its burst length. Results are reported in log scale.

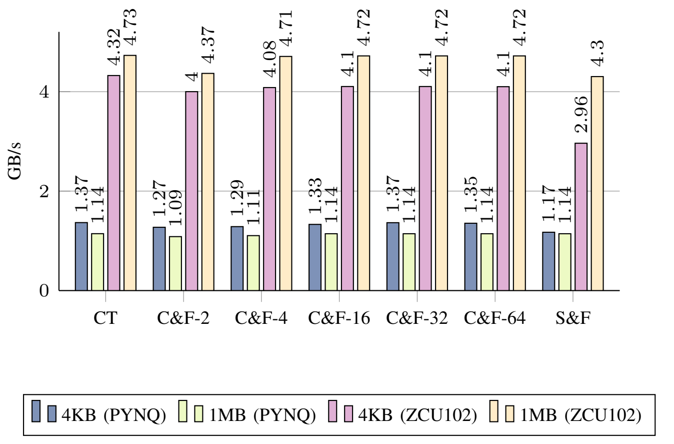


Fig. 7. Average throughput comparison under contention. The hardware accelerators are configured to access separate pages.

component. DMAs execute with no interference. We deployed a custom timer in the FPGA for accurate measurements. We implemented five designs. In each design, I_{AXI} implements: i) cut-through buffers, ii) cut-through buffers protected by ASM, iii) cut-and-forward buffers configured with $C = 4$ (C&F-4), iv) cut-and-forward buffers configured with $C = 16$ (C&F-16), and v) store-and-forward buffers. Figure 5 reports the measured buffering latency as a function of the transaction burst length. Results are the same for ZCU102 and PYNQ and confirm the model proposed in Section IV. In particular, 1) The worst-case buffering latency of store and forward is 256 clock cycles that occurs when the DMA accesses the bus in a greedy manner (256-word transactions). 2) C&F-4 and C&F-16 have a worst-case buffering latency of 4 and 16 cycles, respectively, which is independent of the burst length of the transaction. 3) C&F-4 and C&F-16 reduce the worst-case buffering latency by 98% and 93%, respectively, compared to store and forward. This confirms the effectiveness of the pipelining effect of cut and forward to reduce the buffering latency.

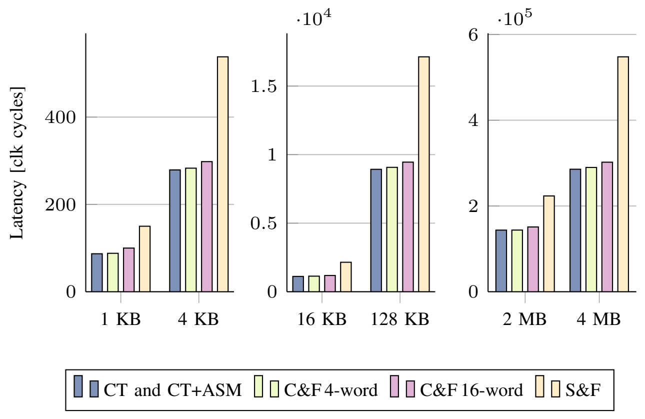


Fig. 6. Memory write access times as a function of accessed data.

B. Memory Write Access Time

In this second experiment, we evaluate the memory access time of the solutions on multiple transactions. The architecture under analysis is the same one as the previous section. However, in this experiment, the DMAs are configured to perform sequential write transactions to the memory. DMAs execute in isolation. Figure 6 reports the overall write access times for ZCU102. We obtained similar results on PYNQ, which are not reported for the sake of brevity. The results confirm that: 1) The buffering time of store and forward strongly impacts the write access time of the DMA. We measured an increase in the measured average write access time of more than 90% in all of the configurations with respect to (w.r.t.) CT. 2) Cut-and-forward buffers have a lower impact on performance. We measured an increase of the average write access time of the DMA around 3% (C&F-4) and around 7% (C&F-16) w.r.t. cut through in all of the configurations.

C. Impact on Average Throughput

Most of the DRAM memory controllers for high-performance implement policies favoring long consecutive accesses as introduced in Section V-A. The granularity of such policies is typically page-wise (e.g., see [19] p.302). The chunking performed by cut and forward could interfere with such policies and impact the average performance. Even though we are targeting critical systems and thus our focus is mainly on the maximum latency of the transactions, this experiment aims at evaluating the impact on the average performance of cut and forward on the platforms under analysis. We implemented a test setup composed of two hardware accelerators, each concurrently writing a private buffer sizing 4KB (one page) or 1MB (256 pages). To highlight the maximum performance impact, we placed the buffers on separate physical pages. We setup the hardware accelerators to request maximum allowable throughput to the DRAM memory controller, setting up for each platform: i) maximum allowed bus width (PYNQ: 64bit, ZCU102: 128bit); ii) maximum FPGA clock (PYNQ: 250Mhz. ZCU102: 330Mhz); iii) most greedy access from hardware accelerators, issuing maximum-size burst transactions.

We deployed seven designs to compare CT, store and forward, and 5 configurations of cut and forward. Figure 7

	HC CT	HC S&F	HC C&F-4	HC C&F-16	SC CT
LUT	2950	5902	3088	3099	3785
FF	1228	1696	1467	1479	7137

TABLE II
AREA COMPARISON FOR THE ZCU102 PLATFORM.

reports the average bandwidth measured on 1000 runs, obtained by dividing the overall, per-run exchanged data for the measured execution time. The considerations follow: 1) The maximum measured impact on average throughput for cut and forward is for C&F-2 and at most 8% w.r.t. cut through. 2) As expected, the impact decreases for higher values of C, e.g., C&F-16 impacts at most 5% the average throughput w.r.t. cut through. This effect is due to a better tradeoff for average throughput between increased signaling on the address channel and the pipelining effect introduced by cut and forward. It is worth noting that the best tradeoff for average throughput can depend on the optimizations policies implemented in the interconnect and peripherals of the platform under evaluation.[2] For the platform under analysis, all of the other tested configuration of cut and forward (except C&F-2) shows similar average performance. 3) The maximum impact of cut and forward on average performance is minor or comparable with store and forward. This confirms that, even though we target latency and resource consumption as major evaluation metrics, cut and forward has never demonstrated a higher impact on average performance with respect to store and forward. Such results are also confirmed by the limited performance impact measured in the real scenario reported in Section VI-E.

D. Resource Consumption

Table II reports the LUTs and FFs consumption for the AXI HyperConnect in the designs deployed in Section VI-B (ZCU102). The cost of the vendor infrastructure AXI SmartConnect deploying cut-through buffers is also reported for comparison (SC CT). The results show that: 1) Store-and-forward buffers require +100% LUTs and +38% FFs for the AXI HC with respect to cut-through buffers. The synthesis tool (Xilinx Vivado) uses more LUTs than FFs to deploy the larger store-and-forward buffer. 2) Cut-and-forward buffers have a lower impact on the footprint of AXI HC. 3) C&F-4 has an impact 48% and 14% lower with respect to S&F, on LUT and FF, respectively. The area impact on LUT and FF is 47% and 13% reduced with respect to S&F when considering C&F-16. This confirms that the area impact of cut-and-forward buffers is controllable and depends on the size of the input buffer, i.e., the parameter C. 4) AXI HC deploying cut-and-forward buffers show lower resource impact with respect to AXI SmartConnect in both of the tested configurations.

[2]This work focuses on latency-critical tasks and therefore addressing the best average performance is beyond the scope of this paper. This experiment has been designed to confirm the configurable impact on the average performance of our technique.

E. Mixed-Critical Application Case Study

This last experiment is a case study aiming at demonstrating the features and performance of cut-and-forward buffers on a typical mixed-critical application scenario. We populated the architecture with three popular commercial hardware accelerators: a Xilinx FIR filter IP [33] HA_1^{RT}, a general-purpose DMA HA_2^{DMA}, and the CHaiDNN hardware accelerator [27] HA_3^{DNN}. The hardware accelerators share the memory in the processing system to collaborate in the execution with the processors. Each HA is activated by a software task running in the processing system: HA_1^{RT} is activated by a periodic and critical software task SW^{RT}. When activated, HA_1^{RT} issues ten 256-word read burst transactions (4KB each), processes the data, and then writes the result to the shared DRAM issuing ten 256-word burst transactions. The entire read-process-write execution must complete within a given deadline D_1^{RT}. HA_1^{RT} is an accelerator used by SW^{RT} to process the data of a critical sensor. A deadline miss of HA_1^{RT} can have fatal consequences. HA_2^{DMA} is a general-purpose DMA that can be activated by non-critical SW-tasks. For instance, HA_2^{DMA} is used to move data by a non-critical audio/video processing software task. HA_3^{DNN} is leveraged by the CHaiDNN software task SW^{DNN} to obtain the maximum execution rates of a quantized DNN model for classification operating on images. We suppose that HA_3^{DNN} must process a minimum required amount of frames per second (FPS) to avoid dangerous conditions. HA_1^{RT} produces a periodic bus workload (activated periodically by SW^{RT}). However, the bus workload of HA_2^{DMA} is unknown and SW^{DNN} makes HA_3^{DNN} greedily adapt to the generated bus workload according to the instant bandwidth availability. This means that the overall bus workload is not fully periodic. This case study cannot be protected from denial of service using ASM. Thus, we implemented four designs, considering the tested configurations of the previous experiment except the one using ASMs: i) CT buffers, ii) C&F buffers, configured with $C = 4$ (C&F-4), iii) C&F buffers, configured with $C = 16$ (C&F-16), and iv) S&F buffers.

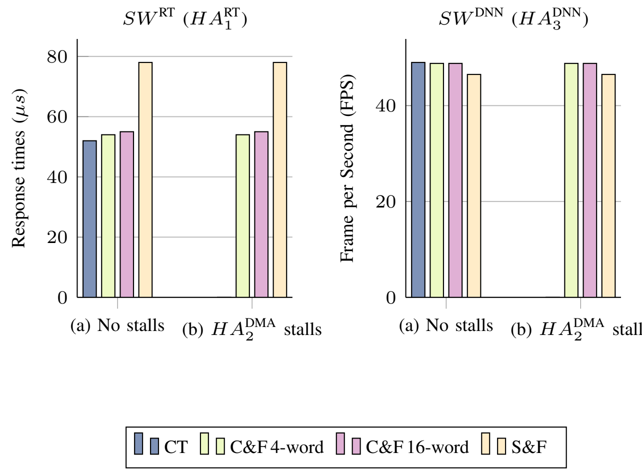


Fig. 8. Measured timing performance for the case-study.

1) Considered Scenarios: For each of the designs, we tested two scenarios: a) concurrent execution of SW^{RT} and SW^{DNN} (HA_2^{DMA} not in use and b) concurrent execution of SW^{RT} and SW^{DNN} while HA_2^{DMA} triggers a misbehaviour stalling the bus. The misbehaviour of HA_2^{DMA} may be due to various reasons – HA_2^{DMA} is assumed to be a generic faulty/malicious/misbehaving/misconfigured component. To showcase the worst-case scenario, we configured HA_2^{DMA} to stall the bus indefinitely. This corresponds to the case in which HA_2^{DMA} introduces a permanent fault in the system, e.g., due to bad development/testing or it is maliciously exploiting such lack of specification to attack the availability of the shared DRAM memory. It is worth noting that limited stalls have similar capabilities to threaten the safety and security of the system.

The average performance measured for ZCU102 on Scenario a) and b) are reported in Figure 8 as response times for SW^{RT} (lower is better) and FPS for SW^{DNN} (higher is better). We could not deploy this design on PYNQ due to limited availability of resources. The performance of SW^{RT} and SW^{DNN} strictly depends on the performance of the leveraged HA, respectively, HA_1^{RT} and HA_3^{DNN}. Considering Scenario a), the average measured response time of SW^{RT} associated with CT is 52 μs. The same response time associated with store and forward is 78 μs, corresponding to an increase of around 50% with respect to cut through. The measured increase confirms that the long buffering time introduced by store-and-forward buffers in the writing phase of HA_1^{RT} strongly impacts the performance of memory-intensive hardware accelerators. Cut-and-forward buffers show a significantly lower impact: we measured an average response time of SW^{RT} equal to 54 μs (C&F-4) and 55 μs (C&F-16), corresponding to an increase of just 4% and 6%, respectively, with respect to CT. Considering SW^{DNN}, we measured a smaller performance impact of store-and-forward buffers. This depends on the less memory intensity of HA_3^{DNN} with respect to HA_1^{RT} and confirms that store and forward has a higher performance impact on memory-intensive tasks. Also, such results confirm the limited impact on average performance introduced in Section VI-C on a realistic case-study.

Concerning Scenario b), Figure 8 does not report any result associated with cut-through buffers. This is because cut-through buffers are not able to contain the misbehavior of HA_2^{DMA} – the shared bus M^I is stalled and the availability of the shared memory is compromised for HA_1^{RT} and for HA_3^{DNN}. The response time of SW^{RT} is unbounded and SW^{DNN} executes 0 FPS – any timing requirement is broken a priori. Store and forward and cut and forward are able to keep the availability of the memory for HA_1^{RT} and HA_3^{DNN} and therefore the critical functionalities running. The measured performances are equal to the one measured in Scenario a), confirming the higher performance impact of store and forward and the ability of cut and forward of making any hardware component independent from other misbehaving hardware components. The bus access is predictable and HA_1^{RT} and HA_3^{DNN} can address their timing requirements regardless of the misbehavior of HA_2^{DMA}.

VII. Related works

The research community and the industry proposed several solutions specifically addressing safety and security on commercial FPGA SoC platforms. Pagani et al. [34] introduced a method to predictably redistribute and allocate the bus bandwidth to the hardware accelerators. Restuccia et al. proposed the AXI HyperConnect [32], a research AXI interconnect for the management of the hardware accelerators from hypervisor technologies and a method to guarantee fair bandwidth distribution among multiple hardware accelerators deployed in modern FPGA SoCs [24]. Rahmatian et al. [35] proposed a method to detect software intrusion through specific modules deployed in the FPGA logic. Kyung et al [36] and Moro et al. [37] proposed two solutions to monitor the AXI traffic generated by the hardware accelerators and eventually react to critical situations by performing a reset of the system network.

The research community also spent multiple efforts proposing arbitration policies for on-chip interconnects aiming at improving the throughput and predictability of the system interconnect [38]–[41]. Modern commercial FPGA SoC platforms integrate blocks for the management of the Quality-of-Service in the Processing System (see [18], p.375) and mechanisms for isolating bus managers (see [18], p. 366). The Xilinx AXI protocol firewall IP is a tool for decoupling and protecting portions of AXI interconnections. Since its version 1.1 [42], it can be configured to timeout on bus stalls, behaving similarly to ASM [15]. In safety-critical systems, this methodology requires the application of a worst-case analysis requiring the full knowledge of the bus traffic generated by the hardware accelerators in the system. This is generally not possible in systems integrating hardware accelerators generating best-effort traffic on the shared bus, as in the case of mixed-critical applications. Moreover, it is worth mentioning that stalls propagation impacts the system performance.

VIII. Conclusions

Commercial FPGA SoCs can be affected by unbounded stalls of the bus that can compromise the availability of the shared resources and cause denial of service. This is a critical issue for safety-critical and security-critical applications and prevents any execution timing guarantee. This paper proposed Cut and Forward, a novel configurable switching methodology targeting safety and security in FPGA SoCs. Cut-and-forward buffers shield the system against misbehaving/malicious hardware accelerators independently of the bus workload and keeping its impact on area and performance under control. The experiments demonstrated the applicability of cut-and-forward buffers in realistic scenarios deployed on commercial FPGA SoCs.

References

[1] M. Wirthlin, "High-reliability fpga-based systems: space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015.

[2] R. B. Atitallah, V. Viswanathan, N. Belanger, and J.-L. Dekeyser, "Fpga-centric design process for avionic simulation and test," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 3, pp. 1047–1065, 2017.

[3] G. Serra, P. Fara, G. Cicero, F. Restuccia, and A. Biondi, "Pac-pl: Enabling control-flow integrity with pointer authentication in fpga soc platforms," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 241–253.

[4] L. Qiao, G. Luo, W. Zhang, and M. Jiang, "Fpga-accelerated iterative reconstruction for transmission electron tomography," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 152–156.

[5] F. Restuccia and A. Biondi, "Time-predictable acceleration of deep neural networks on fpga soc platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 441–454.

[6] M. Damschen, L. Bauer, and J. Henkel, "Corq: Enabling runtime reconfiguration under wcet guarantees for real-time systems," *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 77–80, 2017.

[7] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, "Information flow isolation in i2c and usb," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 254–259.

[8] H. M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Networks on chip with provable security properties," *IEEE Micro*, vol. 34, no. 3, pp. 57–68, 2014.

[9] C. H. Gebotys and R. J. Gebotys, "A framework for security on noc technologies," in *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.* IEEE, 2003, pp. 113–117.

[10] *AMBA® AXI™ and ACE™ Protocol Specification*, ARM, IHI 0022D.

[11] A. Dhavlle, R. Hassan, M. Mittapalli, and S. M. P. Dinakarrao, "Design of hardware trojans and its impact on cps systems: A comprehensive survey," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[12] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, may 2016. [Online]. Available: https://doi.org/10.1145/2906147

[13] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a comprehensive and systematic classification of hardware trojans," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 1871–1874.

[14] A. Meza, F. Restuccia, and R. Kastner, "Safety verification of third-party hardware modules via information flow tracking," in *1st Real-time And intelliGent Edge computing workshop (RAGE) co-located with the 2022 59th Design Automation Conference (DAC).*

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

[16] S. Lam, "Store-and-forward buffer requirements in a packet switching network," *IEEE Transactions on Communications*, vol. 24, no. 4, pp. 394–403, 1976.

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

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

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

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

[21] *Zynq DPU Product Guide*, Xilinx-AMD, pG338.

[22] *AXI CDMA, LogiCORE IP Product Guide, Document number PG034*, Xilinx.

[23] F. Restuccia, A. Meza, and R. Kastner, "Aker: A design and verification framework for safe and secure soc access control," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

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

[25] *Vitis AI User Guide*, Xilinx, uG1414v1.3.

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

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

[28] *AXI CDMA, LogiCORE IP Product Guide, PG034*, Xilinx.

[29] H. Yun, R. Pellizzon, and P. K. Valsan, "Parallelism-aware memory interference delay analysis for cots multicore systems," in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015, pp. 184–195.

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

[31] *UltraScale Architecture-Based FPGAs Memory IP v1.4*, Xilinx, 2021, pG150.

[32] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[33] *FIR Compiler, LogiCORE IP Product Guide*, Xilinx Inc., 2018, pG149.

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

[35] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-assisted detection of malicious software in embedded systems," *IEEE Embedded Systems Letters*, vol. 4, no. 4, pp. 94–97, 2012.

[36] H.-m. Kyung, G.-h. Park, J. W. Kwak, T.-j. Kim, and S.-B. Park, "Design and implementation of performance analysis unit (pau) for axi-based multi-core system on chip (soc)," *microprocessors and microsystems*, vol. 34, no. 2-4, pp. 102–116, 2010.

[37] A. Moro, F. Federici, G. Valente, L. Pomante, M. Faccio, and V. Muttillo, "Hardware performance sniffers for embedded systems profiling," in *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*. IEEE, 2015, pp. 29–34.

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

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

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

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

[42] *AXI Protocol Firewall IP v1.1*, Xilinx, 2021, pG293.

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

Ryan Kastner is a professor in the Department of Computer Science and Engineering at UC San Diego. He received a PhD in Computer Science at UCLA, a masters degree in engineering and bachelor degrees in Electrical Engineering and Computer Engineering from Northwestern University. His current research interests include hardware acceleration, hardware security, and remote sensing.