Brief Industry Paper: AXI-Interconnect$^{\mathrm{RT}}$: Towards a Real-Time AXI-Interconnect for System-on-Chips

Zhe Jiang*‖, Neil Audsley*, Dayu Shi‖, Kecheng Yang†, Nathan Fisher‡, Zheng Dong‡
‖ARM Ltd, United Kingdom,*University of York, United Kingdom,†Texas State University, USA, ‡Wayne State University, USA

***Abstract*—In modern, real-time heterogeneous systems, ensuring the predictability of interconnects is becoming increasingly important. Existing interconnects are mainly designed to achieve high throughput, with their micro-architectures usually based on FIFO queues. This FIFO-based design prevents prioritization of transactions based on their importance, leading to difficulties in ensuring transaction predictability, especially in a system with a large number of system components. In this paper, we introduce *AXI-Interconnect*$^{\mathrm{RT}}$ – a real-time AXI interconnect for heterogeneous SoCs, which redefines the micro-architecture of interconnects by enabling random accesses of buffered transactions and organizing transactions using dedicated hardware units. With the new micro-architecture, *AXI-Interconnect*$^{\mathrm{RT}}$ can manage transactions based on their importance, guaranteeing their predictability.**

I. Introduction

The complexity of today's System-on-Chips (SoCs) is increasing dramatically, mainly driven by the diverse functionalities required by modern embedded computing (*e.g.*, image recognition in automated driving [1]) and the rapid evolution of manufacturing processes in the semiconductor industry (*e.g.*, the ability to produce $5nm$ ASICs [2]). Although modern SoCs developed by different vendors usually have different architectures, *heterogeneity* is always the key to more functionality [3], [4]. That is, the SoCs couple processing units with different architectures, including hardware accelerators (HAs), on the same chip. For instance, Apple's M1 [5] integrates CPUs with the GPU and a neural engine to accelerate image processing with machine learning (ML) related applications.

As a '*bridge*' between system components, an interconnect becomes a dominant factor when determining the real-time performance of heterogeneous SoCs. It is impractical to manage interconnect traffic flows solely from the system software level [6], as the masters in heterogeneous SoCs are usually designed with different instruction sets. This makes managing interconnect transactions at the software level unpredictable, with an extremely high overhead, as frequent inter-master communication and translation is required [3]. Therefore, it is crucial to guarantee *predictability* and *throughput* of an interconnect at the *hardware level*.

ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI) [6] is the most widely used *de-facto* standard interface for interconnects, and is used by billions of SoCs each year. A number of industrial interconnects are based on this protocol, *e.g.*, Xilinx's AXI-InterConnect [7] and AXI-SmartConnect [8]. However, most of these were not designed for real-time application scenarios. Some prototype interconnects which consider real-time performance include Restuccia *et al.*'s AXI-HyperConnect [9] and Garside *et al.*'s BlueTree [10]. These interconnects usually adopt different mechanisms (*e.g.*, bandwidth reservation) to ensure specific transaction path predictability. The designs of the interconnects are usually based on *FIFO queues*, which prevent prioritization based on importance and leave the real-time performance of an interconnect entirely to scheduling from the software level. However, as mentioned above, it is difficult to ensure the predictability of an interconnect from the software level, as the system components in heterogeneous SoCs usually execute independently.

In this paper, we propose a real-time AXI interconnect for heterogeneous SoCs (*AXI-Interconnect*$^{\mathrm{RT}}$), which redefines the micro-architecture of the interconnects. *AXI-Interconnect*$^{\mathrm{RT}}$ enables random accesses of buffered transactions and introduces dedicated Transaction Control Units (TCUs) to schedule them. With the new features, *AXI-Interconnect*$^{\mathrm{RT}}$ can prioritize transactions based on their importance, ensuring their predictability.

II. Preliminaries

A. Modern Heterogeneous SoC

We illustrate a prototype heterogeneous SoC architecture in Fig. 1, satisfying commonly required functionalities in modern safety-critical systems. The introduced SoC is built on a Xilinx VC709 evaluation board and contains four major sub-systems: a core subsystem, a memory subsystem, a BlueShell DNN HA, and I/O subsystems.

Core subsystem. The core subsystem is responsible for execution of general-purpose software applications and operating systems (OSs). It contains two dual-core RSIC-V processors with instruction caches, data caches, and local static RAMs (SRAMs).

Memory subsystem. The memory subsystem manages memory resources shared between different system components. This subsystem

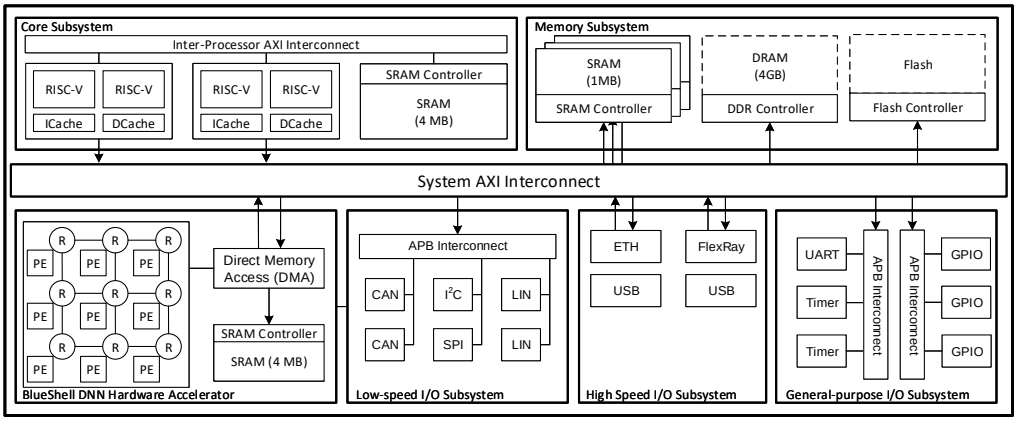


Fig. 1. Hardware architecture of heterogeneous SoC built on Xilinx VC709 (R: Router/Arbiter; PE: Processing element; GPIO: General-purpose Input/Output).

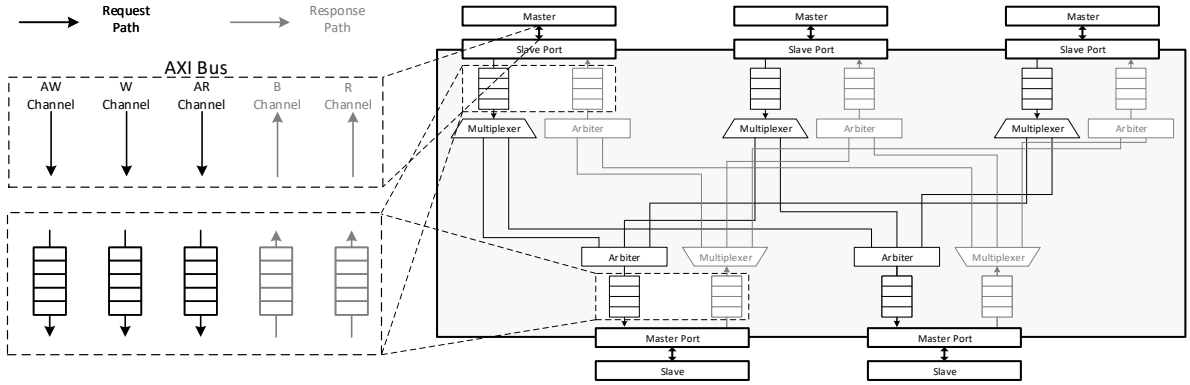


Fig. 2. Generalized hardware micro-architecture of conventional AMBA AXI framework (SCH: scheduler), containing AXI bus, ports, Interconnect.

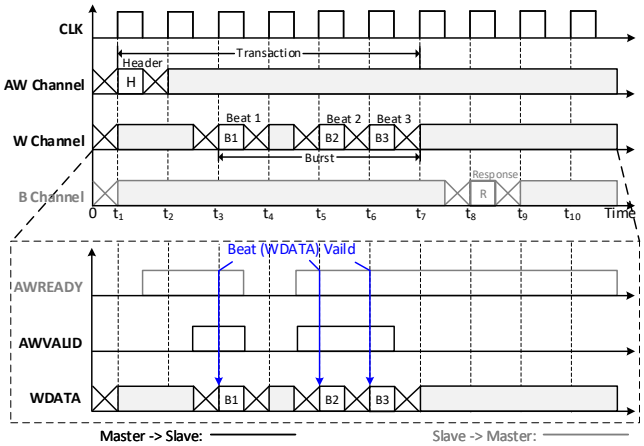


Fig. 3. A write transaction in AMBA AXI involves AW, W and B channels.

has three types of memory: off-chip flash and Dynamic RAMs (DRAMs), and on-chip SRAMs. The on-chip memory is smaller than the off-chip memory, but can provide faster memory accesses.

BlueShell DNN HA. BlueShell is a specialized DNN inference HA, containing 9 RSIC Processing Elements (PEs) arranged in a 3×3 array. The PEs are connected to the routers of a 2D mesh type open-source Network-on-Chip (NoC) [11]. The DNN HA accelerates the execution of DNNs by enabling parallel computation of different DNN blocks.

I/O subsystems. I/O subsystems manage shared I/O peripherals. Based on common features of the I/Os, we split the subsystems into three specialized domains for general-purpose I/Os, low-speed I/Os, and high-speed I/Os.

These illustrated sub-systems and their internal components are connected using a system AXI interconnect.

B. ARM Advanced Microcontroller Bus Architecture

There are industrial and academic interconnects designed in compliance with AMBA AXI, *e.g.*, [8] and [9]. However, the design details of these interconnects are not usually publicly disclosed. In Fig. 2, we summarize a generalized framework of AXI interconnect based on the official protocol [6] and existing IP documentation. We now introduce the essential components of the AXI interconnect and the AXI transactions.

AXI bus. The AMBA AXI protocol defines a *master-slave* interface, which allows simultaneous, bi-directional data exchange. An AXI interface introduces five independent communication channels: *Address Read* (*AR*), *Address Write* (*AW*), *Read Data* (*R*), *Write Data* (*W*), and *Write Response* (*B*). Each of these channels has a group of standard-defined signals [6].

AXI port. The AMBA AXI protocol introduces two types of port: *master port* and *slave port*. A master/slave port physically connects the slave/master via the AXI bus. Corresponding to the AXI bus, an AXI port also contains the five communication channels.

AXI interconnect. An AXI interconnect has two responsibilities: (i) receiving transactions sent from a master/slave and then routing them to the corresponding destinations. (ii) organizing the order of transactions when a master/slave receives multiple transactions.

To this end, an AXI interconnect introduces a group of FIFO queues and multiplexers for each slave port in the *request path*. During run-time, the FIFO queues respectively buffer the requests in the AR, AW, and W channels, and the multiplexers select the destinations of these requests. Simultaneously, a group of FIFO queues and arbiters are connected to each master port. These arbiters are entirely independent of each other, and decide the access order of requests sent to the connected slaves. In most existing work and commercial products (*e.g.*, [3], [6], [12]), a round-robin scheduling policy is adopted in the arbiters. *In the response path*, symmetric structures of the R and B channels are implemented.

AXI transactions. In AMBA AXI, a *transaction* is always initialized by a master. To issue a read/write transaction, a master first sends a *header* packet to a slave using the AR/AW channel, containing information necessary for the transaction, *e.g.*, length. In read procedures, response data is transferred back to the master via the R channel. In write procedures, write data is routed to a slave via the W channel, and the slave uses the B channel to acknowledge the transmission from the master. Fig. 3 demonstrates a complete write transaction.

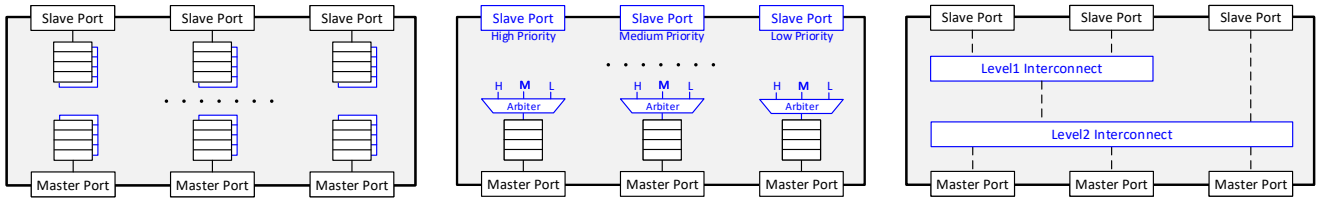
AXI provides two methods for data transfer: *single transaction mode* and *burst transaction mode*. A master can read/write up to 256 addresses in the burst transaction mode; whereas it can only read/write a single address in the single transaction mode. From a design perspective, the single transaction mode can be treated as a burst transaction mode with a single read/write address. Therefore, in this paper, we assume that all transactions are generated in the burst mode. Following the AXI protocol [6], we call the data payload of a transaction a *burst*, and the packets of a burst *beat*s. In Fig. 3, the the master issues a *write burst* with *3 beats* in a transaction.

As regulated in [6], a master *must* generate the header and bursts in the same order, and the transmission of a burst is *non-preemptive*.

III. Research Challenges and Related Work

A. Research Challenges

In heterogeneous SoCs (*e.g.*, Fig. 1), an AXI port and its associated FIFO queues are often shared between multiple masters/slaves. The implementation of FIFO queues prevents *prioritization* of transactions, and leaves the real-time performance of an AXI interconnect entirely to the arrival order of the transactions. However, masters



(a) Duplicated channels. (b) Reserved bandwidth. (c) Hierarchical structure.

Fig. 4. Hardware micro-architectures of real-time interconnects. The blue portions highlight the modifications compared to conventional interconnects.

and slaves in heterogeneous SoCs usually execute independently. This leads to frequent contentions in the interconnect, significantly damaging its *predictability* and *analyzability*.

More seriously, the FIFO-based structure leads to *random* occurrences of *physical priority inversion* at both slave and master ports. That is, a low-priority transaction blocks a high-priority transaction when both transactions are buffered in the same FIFO queue and the low-priority transaction arrives before the high-priority transaction. We detail this blocking below.

Slave port blocking. Slave port blocking occurs between masters using the *same* slave port, which is observed in FIFOs connected to slave ports. Taking Fig. 1 as an example, PEs in the DNN HA can cause frequent slave port blocking, when they keep accessing the interconnect concurrently.

Master port blocking. Master port blocking occurs when the masters connected to *different* slave ports send transactions to the same destination simultaneously. This blocking is observed at the FIFOs connected to the master ports. Taking Fig. 1 as an example, processors in the core subsystem and PEs in the DDN HA can suffer frequent master port blocking when they keep reading the same memory block at the same time.

Moreover, to solve these issues in a heterogeneous SoC, both *dependency* and *scalability* must be taken into account:

Dependency. As shown in Fig. 2, an AXI interconnect *fully* connects the masters and slaves in the system. Therefore, blocking which occurs in one transaction path can generate or magnify blocks in the other paths. More seriously, such interference usually occurs *repetitively* and *recursively* between the transaction paths, which largely magnifies the unpredictability of the interconnect.

Scalability. With the increase in system complexity, modern heterogeneous SoCs usually introduce additional masters and slaves, which creates more transaction paths in the interconnect. These additionally introduced transaction paths lead to extra resource contentions/blocking, further magnifying the above problems.

These issues lead to challenges in designing a real-time AXI interconnect for modern heterogeneous SoCs. In the next section, we examine the existing work contributing to this research scenario.

B. Related Work

Existing work focusing on the real-time performance of interconnects can be mainly classified as *duplicated channels*, *bandwidth reservation*, and *hierarchical connections*. Note, we do not restrict the communication protocols of the reviewed work. Ideally, all these methods are compatible with AMBA AXI.

Duplicated channels (Fig. 4(a)). A straightforward way to improve an interconnect's real-time performance is duplicating its communication channels. For example, Loh *et al.* [13] implemented "virtual channels" for all transaction paths of the interconnect. Burns [14] adopted a similar method, performing a detailed theoretical analysis to bound the worst-case behaviors of the interconnect. The same design concept has also been widely adopted in industry. In industrial patents, duplicated channels are usually created for specific transactions, *e.g.*, privileged/secure messages [15], video processes [16], and I/O communications [17].

As evidenced in both experimental and theoretical evaluation [13], [14], duplicating the communication channels considerably enhances system-level *throughput*. However, the additionally introduced channels bring extra resource contentions, *e.g.*, transactions buffered in different virtual channels can simultaneously access a master port, which further magnifies the problems reviewed in Section III-A. At the same time, this method also significantly increases *hardware consumption*, as the implementation of communication channels (*i.e.*, register chains) dominates the interconnect's hardware overhead. For instance, a dual-channel interconnect usually consumes nearly two times the hardware compared to a single-channel interconnect.

Bandwidth reservation (Fig. 4(b)). Bandwidth reservation is usually used to ensure the services of specific transaction paths. To achieve this, the interconnect first assigns a static priority to each master (or slave port) and then allocates them a certain bandwidth based on their associated priorities. For instance, Restuccia *et al.* [18] introduced an AXI Burst Equalizer (ABE) to re-organize transactions, which guarantees that all slave ports can use identical bandwidth; Pagani *et al.* [19] proposed a dedicated AXI controller to reserve bandwidth for high priority masters. In addition, some work on *run-time* bandwidth allocation optimizes arbiters' scheduling strategies. For instance, Hebbache *et al.* [20] introduced a dynamic arbitration scheme for the arbiters.

Bandwidth reservation effectively ensures throughput and predictability of high priority transactions. However, this method cannot fundamentally solve contentions and blocking in the interconnect. At the same time, it also reduces *design flexibility* and *interconnect utilization*, since the interconnect must accurately acquire transaction information before run-time and always ensure sufficient bandwidth for the high-priority masters at run-time.

Hierarchical connections (Fig. 4(c)). Different from the above work, relying on an independent arbiter to schedule transactions sent to the connected master port, there are also interconnects with multiple hierarchies. In a hierarchical structure, the slave ports are grouped into different partitions by the interconnect. Slave ports in the same group are connected to a local interconnect. At the same time, a global interconnect connects these local interconnects and master ports. For example, Restuccia *et al.* [12] implemented a 2-level AXI interconnect with the corresponding theoretical analysis. Audsley [21] introduced a tree-like structure to support multiple-level connections between local interconnects. Garside *et al.* [10] further extended this structure [21] to support 64 masters.

This method brings partial optimizations to the interconnect. However, like the other methods, it cannot solve contentions and blocking in an interconnect due to the lack of a global view.

IV. *AXI-Interconnect*$^{\mathrm{RT}}$ Design

In this section, we present the top-level design of a real-time AXI Interconnect (*AXI-Interconnect*$^{\mathrm{RT}}$), which could simultaneously guarantee transaction predictability and throughput.

A. Design Concepts

We present three design concepts (DCs) for *AXI-Interconnect*$^{\mathrm{RT}}$:

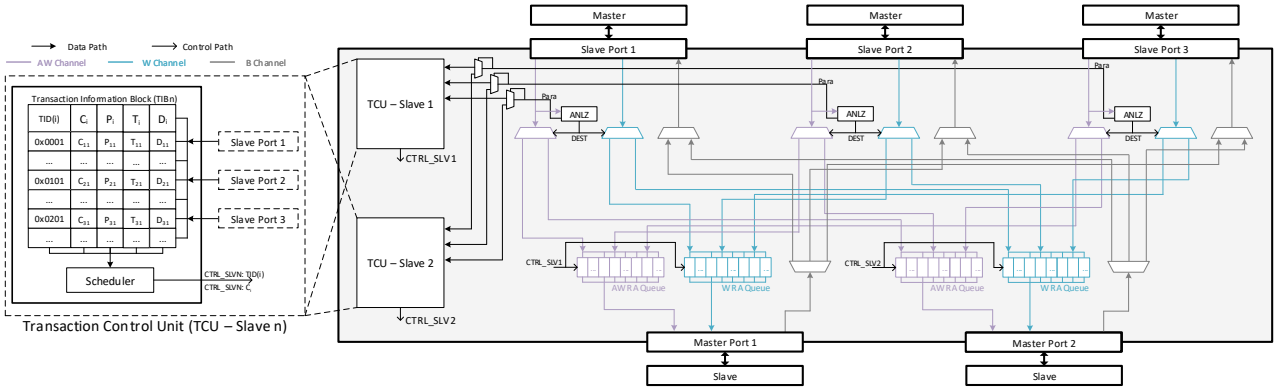


Fig. 5. Hardware micro-architecture of AW, W and B channels in *AXI-Interconnect*$^{\text{RT}}$ (ANLZ: Analyzer; DEST: Destination; Para: Parameter). AMBA AXI regulates independent write and read transactions; *AXI-Interconnect*$^{\text{RT}}$ also contains a symmetric structure for AR and R channels.

DC 1: FIFO-free connections. *AXI-Interconnect*$^{\text{RT}}$ replaces the FIFO queues with new *Random Access queues* (RA queues), enabling random accesses of buffered transactions. The new micro-architecture effectively avoids the *physical* priority inversion and provides the possibility of transaction prioritization.

DC 2: Multiple-Write-Single-Read RA queues. The RA queue introduces a Multiple-Write-Single-Read (MWSR) structure, allowing different masters to write to it and a single slave to read from it simultaneously. This structure prevents resource contentions between different masters.

DC 3: 1-Cycle decoding and grouping. *AXI-Interconnect*$^{\text{RT}}$ decodes buffered transactions to extract associated parameters, and then correspondingly re-groups them.

DC 4: Online scheduling. *AXI-Interconnect*$^{\text{RT}}$ introduces independent global scheduling for each slave, which prioritizes transactions based on the associated parameters.

B. Overall Miro-architecture

The top-level micro-architecture of *AXI-Interconnect*$^{\text{RT}}$ is shown in Fig. 5. We retain the standard slave and master ports at interfaces, ensuring compatibility with heterogeneous SoCs designed with conventional AXI interconnects.

For each *slave port*, we remove the connected FIFO queues (in the conventional interconnect) and introduce a transaction analyzer connected to both AW and W channels. The analyzer monitors and decomposes transaction headers in the AW channel, which (i) determines the destination of each header and its following burst; and, (ii) extracts the associated parameters of this transaction, *e.g.*, length, period, etc. For each *master port*, we introduce two RA queues and a Transaction Control Unit (TCU). The RA queues respectively buffer the header and send bursts to the connected slave, whilst the TCU organizes the transfer orders.

Moreover, we adopt response path pass-through connections (*i.e.*, B channel) because transactions always involve a single-packet response and the masters (e.g., processors) are usually hundreds of times faster than the slaves (e.g., memory and I/Os). This leads the B channel to be block-free.

C. Run-time behaviors.

During run-time, analyzers continually monitor the AW channels. Once an analyzer captures a header transferred via its monitored AW channel, the analyzer decodes the header and sends the decoded destination and other associated parameters to the connected multiplexers and corresponding TCU respectively. The multiplexer then routes the received header and the following burst to the corresponding RA queues. Simultaneously, each TCU maintains the parameters of the transactions sent to its managed slave using a transmission information table (TIB), and organizes the transfer orders. Since the transfer of a burst is non-preemptive [6], a TCU only schedules the next transaction after the completion of each burst transaction.

As shown in Fig. 5, the design of the TCU is modularized, which allows system designers to customize the scheduling strategy.

V. Conclusion

In this paper, we identify and analyze predictability issues around AXI interconnects in heterogeneous SoCs, and summarize the work contributing to the research in this area. We also present *AXI-Interconnect*$^{\text{RT}}$, a real-time AXI interconnect, which can prioritize transactions at the hardware level and guarantee transaction predictability. We specifically introduce the design concepts and top-level micro-architecture of *AXI-Interconnect*$^{\text{RT}}$.

References

[1] I. ISO, "26262: Road vehicles-functional safety," 2018.
[2] S. Anthony, "Ibm unveils world's first 5nm chip," *Ars Technica*, 2017.
[3] Z. Jiang *et al.*, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *RTAS*, 2018.
[4] Z. Jiang, K. Yang, N. Fisher, N. Audsley, and Z. Dong, "Pythia-mcs: Enabling quarter-clairvoyance in i/o-driven mixed-criticality systems," in *2020 IEEE Real-Time Systems Symposium (RTSS)*.
[5] "Website: Apple m1," https://www.apple.com/uk/mac/m1/.
[6] ARM, "Amba axi 5.0," https://developer.arm.com/architectures/system-architectures/amba/amba-5.
[7] "Xilinx Interconnect," https://www.xilinx.com/axi_interconnect.
[8] "Xilinx SmartConnect," https://www.xilinx.com/smart-connect.
[9] F. Restuccia *et al.*, "Axi hyperconnect: a predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *DAC*, 2020.
[10] J. Garside *et al.*, "Wcet preserving hardware prefetch for many-core real-time systems," in *RTNS*, 2014.
[11] G. Plumbridge *et al.*, "Blueshell: a platform for rapid prototyping of multiprocessor nocs," *Computer Architecture News*, 2014.
[12] F. Restuccia *et al.*, "Modeling and analysis of bus contention for hardware accelerators in fpga socs," in *ECRTS*, 2020.
[13] G. H. Loh *et al.*, "Interconnect-memory challenges for multi-chip, silicon interposer systems," in *ISMS*, 2015.
[14] A. Burns *et al.*, "A novel flow control mechanism to avoid multi-point progressive blocking in hard real-time nocs," in *RTAS*, 2020.
[15] "Website: Arm trustzone," https://developer.arm.com/ip/trustzone.
[16] B. Parten *et al.*, "Multi-channel peripheral interconnect supporting simultaneous video and bus protocols," 2017, US Patent.
[17] P. R. Chandra *et al.*, "Multi-protocol tunneling over an i/o interconnect," 2014, US Patent 8,775,713.
[18] F. Restuccia *et al.*, "Is your bus arbiter really fair? restoring fairness in axi interconnects for fpga socs," *TECS*, 2019.
[19] M. Pagani *et al.*, "A bandwidth reservation mechanism for axi-based hardware accelerators on fpgas," 2020.
[20] F. Hebbache, M. Jan, F. Brandner, and L. Pautet, "Shedding the shackles of time-division multiplexing," in *RTSS 2018*.
[21] N. Audsley, "Memory architecture for noc-based real-time mixed criticality systems," *Proc. WMC, RTSS*, pp. 37–42, 2013.