AXI-IC$^{\mathrm{RT}}$: Towards a Real-Time AXI-Interconnect for Highly Integrated SoCs

Zhe Jiang, Kecheng Yang, Nathan Fisher, Ian Gray, Neil Audsley, Zheng Dong§

Abstract—In modern real-time heterogeneous System-on-Chips (SoCs), ensuring the predictability of interconnects is becoming increasingly important. Most of the existing interconnects are mainly designed to achieve high throughput, with their micro-architectures usually based on FIFO queues. The FIFO-based design prevents transaction prioritization based on importance and leads to occurrences of physical priority inversion. Such problems lead to difficulties in ensuring transaction predictability, especially when the system scales to a large number of elements. In this paper, we introduce AXI-Interconnect$^{\mathrm{RT}}$ (*AXI-IC*$^{\mathrm{RT}}$, for short) – 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 through compositional scheduling. This hardware-software co-design approach provides predictable and scalable real-time performance for highly integrated SoCs.

Index Terms—Real-time Systems, Many-core Systems, Interconnect, Schedulability, Scalability.

1 Introduction

In real-time systems, the complexity of SoCs increases dramatically, as a result of 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 from different vendors typically have different architectures, *heterogeneity* is always the key to more functionality [3]–[5]. That is, the SoCs couple processing units with different architectures, including hardware accelerators (HAs), on the same chip, *e.g.*, Tesla's FSD Chip [6] integrates CPUs with GPUs and a neural processing unit to accelerate image processing and machine learning related applications.

As '*bridges*' between different system elements, *interconnect*s become a dominant factor when determining the real-time performance of heterogeneous SoCs [7]. It is impractical to manage the traffic flow of an interconnect solely from the system software level, since the *Primaries* (*e.g.*, processors) in heterogeneous SoCs are usually designed with different instruction architectures (ISAs). This makes managing interconnect transactions at the software level unreliable, with an extremely high overhead, as frequent inter-Primary communication and translation are required [7], [8]. Therefore, it is crucial to guarantee *predictability* and *throughput* of an interconnect at the *hardware level*.

The ARM Advanced Microcontroller Bus Architecture Advanced eXtensible Interface (AMBA AXI) [9] is the most widely used *de-facto* standard interface for interconnects, which is used by billions of SoCs each year. A number of industrial interconnects are based on this protocol, *e.g.*, Xilinx's AXI-InterConnect [10] and AXI-SmartConnect [11]. However, most of these were not designed for real-time application scenarios. Within the context of real-time systems, some prototype interconnects have been developed. These include, Restuccia *et al.* [12], Pagani *et al.* [13], and Gomony *et al.* [14]. Existing interconnect designs are usually based on *FIFO queues*, which prevent transaction 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 elements in heterogeneous SoCs usually execute *independently*. Moreover, the FIFO-based design also allows a low-priority transaction to block 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. This phenomenon is also called *physical priority inversion*, which brings additional unpredictability to the interconnect. Even worse, these problems are further magnified when the system scales with more hardware elements, since extra resource contention/blocking is introduced.

Contributions. We present AXI-Interconnect$^{\mathrm{RT}}$ (*AXI-IC*$^{\mathrm{RT}}$) to provide *guaranteed* real-time performance in highly integrated SoCs. Specifically, we present

- A novel *micro-architecture*, enabling random accesses of buffered transactions and allowing transaction prioritization based on importance.

- *Zhe Jiang is with Central Engineering, ARM, United Kingdom, S1 4LW; and Computer Science Department, University of Cambridge, CB3 0FD.*
- *Kecheng Yang is with the Department of Computer Science, Texas State University, San Marcos, TX 78666.*
- *Ian Gray is with Computer Science Department, University of York, United Kingdom, YO10 5GH.*
- *Neil Audsley is with Department of Computer Science, City, University of London, EC1V 0HB.*
- *Zheng Dong and Nathan Fisher are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202.*

This work was supported in part by the U.S. National Science Foundation under Grants CNS-2103604, CNS-2140346, CNS 2113817, CNS-2038609, IIS-1724227, CCF-2118202 and CNS-2104181, in part by a start-up Grant from Wayne State University, in part by start-up and REP grants from Texas State University.

§. Corresponding author, Zheng Dong (dong@wayne.edu).

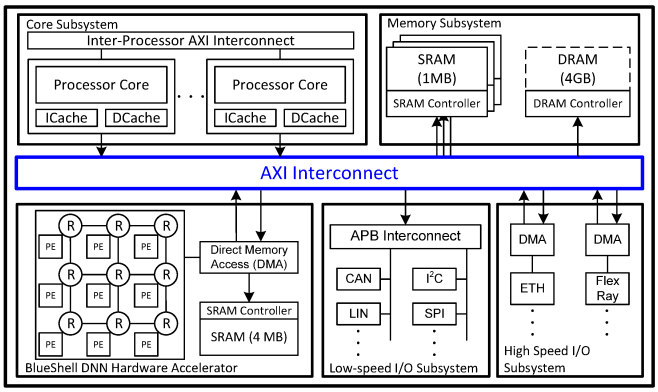


Fig. 1. Hardware architecture of a heterogeneous SoC (R: Router/Arbiter; PE: Processing element).

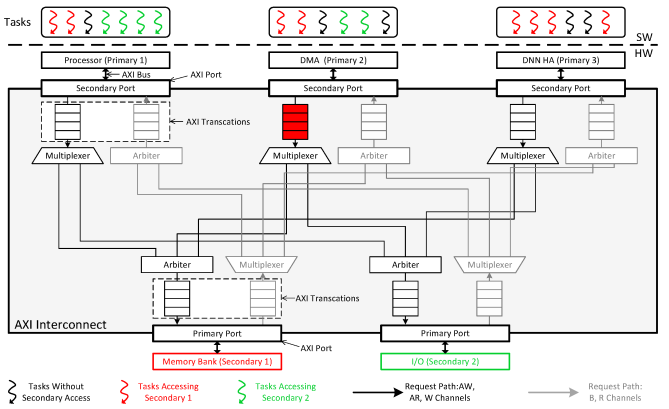


Fig. 2. Generalized hardware micro-architecture of conventional AXI interconnect. FIFO queues in the conventional AXI interconnect prevent *prioritization* of transactions based on their importance. Hence, a low-priority transaction may *physically* block high-priority transactions issued either from the same port or different ports.

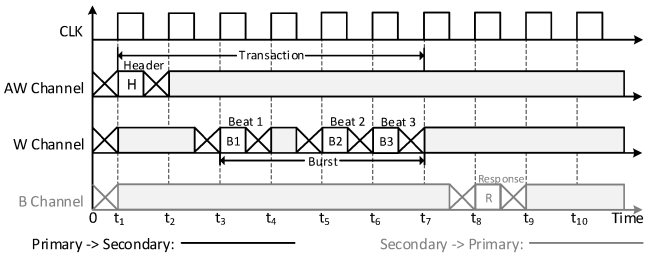


Fig. 3. A write transaction in AMBA AXI involves AW, W and B channels.

- A scalable *two-layer compositional scheduler* for *AXI-IC*$^{\text{RT}}$, allocating the access time of the Secondaries to the Primaries in a hierarchical manner, with guaranteed real-time performance.
- A limited preemptive task model, describing execution behavior of real-time jobs under the new architecture.
- A pseudo-polynomial time algorithm to address the problem of selecting an optimal period and budget for components consisting of sporadic task systems under compositional scheduling. In coping with the limited preemptivity of real-time tasks, a critical observation (from Fig. 11) is provided to quantify the limited preemptive scheduling induced utilization loss at the server task level.
- A comprehensive *experimental evaluation*, including a real-world use case, examining overhead, predictability and performance of *AXI-IC*$^{\text{RT}}$ compared with state-of-the-art interconnects.

The rest of the paper is organized as follows, Sec. 2 gives the background to the architecture of heterogeneous SoC and ARM AMBA-AXI. Sec. 3 explains the real-time issues associated with existing AXI interconnects. Sec. 4 and Sec. 5 present the design of the real-time AXI interconnect (*AXI-IC*$^{\text{RT}}$), which resolves the revealed architectural issues. Sec. 6 introduces an interface selection algorithm for *AXI-IC*$^{\text{RT}}$'s run-time configurations, further optimizing the real-time performance of *AXI-IC*$^{\text{RT}}$. Sec. 7 evaluates the *AXI-IC*$^{\text{RT}}$ and Sec. 8 concludes the paper.

2 Preliminaries

In this section, we introduce the top-level architecture of a heterogeneous SoC (our platform) and the essential concepts of an AXI interconnect.

2.1 Modern Heterogeneous SoC

It is difficult to introduce a unified hardware architecture for different SoCs, because hardware architectures vary depending on the practical demands. However, Fig. 1 gives a top-level overview of a prototype heterogeneous SoC [7], satisfying commonly required functionalities in modern safety-critical systems, which is also used in the later evaluation. The introduced SoC is built on a Xilinx VC709 evaluation board and contains four major subsystems:

Core subsystem. The core subsystem is responsible for the execution of general-purpose software applications and operating systems (OSs). It contains configurable numbers of MicroBlaze processors [15] with instruction and data caches.
Memory subsystem. The memory subsystem manages memory resources shared between different system elements, including both on-chip Static RAMs (SRAMs) and off-chip Dynamic RAMs (DRAMs). The SRAMs are smaller than the DRAMs, but enable faster memory access.
BlueShell Deep Neural Network (DNN) HA. The DNN HA is modified using an open-source Network-on-Chip (NoC), containing nine Multiply-And-Accumulate (MAC) Processing Elements (PEs) arranged in a 3×3 array [16]. The DNN HA accelerates the execution of DNN inferences by enabling parallel computation of different DNN blocks.
I/O subsystems. I/O subsystems contain shared I/O peripherals. Based on common features of the I/O peripherals, we split the subsystems into two specialized domains, one for low-speed I/Os and the other for high-speed I/Os.

The key trend of modern heterogeneous SoCs is to couple increasingly more hardware elements (with different architectures) on the same chip, leading to the hardware becoming highly integrated [3]–[5]. These hardware elements are usually connected using an ***AXI interconnect***.

2.2 ARM AMBA-AXI

Industrial and academic interconnects designed in compliance with AMBA AXI, can be found, *e.g.*, [11] and [12];

however, the design details of these interconnects are not always publicly available. Fig. 2 shows a generalized AXI interconnect based on the official protocol [9] and existing IP documentation. Using this diagram, we introduce the essential elements of an AXI interconnect and AXI transactions.

AXI bus. The AMBA AXI protocol defines a *Primary-Secondary* interface, which allows simultaneous, bi-directional data exchange. The AXI protocol 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 [9].

AXI transactions. In AMBA AXI, a *transaction* is always initialized by a *Primary* (*e.g.*, a processor). To issue a read/write transaction, the Primary first sends a *header* packet containing the information necessary for the transaction (*e.g.*, Secondary address) to a Secondary using the AR/AW channel. In *read* procedures, response data is transferred back to the Primary using the R channel. In *write* procedures, write data is routed to a Secondary via the W channel, and the Secondary uses the B channel to acknowledge the transmission from the Primary. Fig. 3 shows an example of a write transaction at the hardware level, where the black lines show the transmissions from Primaries to Secondaries (*i.e.*, request paths) and the lighter grey lines show the transmissions from Secondaries to Primaries (*i.e.*, response paths). Following the AXI protocol [9], we call the data payload of a transaction a *burst*, and the packets of a burst *beat*s. In Fig. 3, the Primary issues a *write burst* with *3 beats* in a transaction.

AXI port. The AMBA AXI protocol introduces two types of ports: *Primary* and *Secondary port*s. As regulated by AMBA AXI 5.0 [9], Primary ports connect Secondaries, and the Secondary ports connect the Primaries. Such connectivity is established using the AXI bus. Corresponding to the AXI bus, AXI ports also contain five communication channels.

AXI interconnect. An AXI interconnect has two responsibilities: (i) receiving the requests/responses sent from a Primary/Secondary and then routing them to the corresponding destinations; (ii) organizing the transaction order when a Secondary/Primary receives multiple requests/responses.

To this end, an AXI interconnect introduces a group of FIFO queues and multiplexers for each Secondary port in the *request path*. During run-time, the FIFO queues buffer the requests in the AR, AW, and W channels respectively, and the multiplexers select the destinations of these requests. Simultaneously, a group of FIFO queues and arbiters are connected to each Primary port. These arbiters are entirely independent of each other, and decide the access order of requests sent to the connected Secondaries. In most existing work and commercial IPs (*e.g.*, [3], [8], [9]), a round-robin scheduling policy is adopted in the arbiters. *In the response path*, symmetric structures of the R and B channels are implemented.

3 Research Challenges and Related Work

3.1 Research Challenges and Motivation

Conventional AXI interconnects cannot ensure the *time-predictability* of transactions, because they are usually designed using FIFO queues. The FIFO-based design serves transactions according to their arrival order, preventing the *prioritization* of transactions based on their importance and leaving the predictability of the transactions entirely to the Primaries and Secondaries. However, Primaries and Secondaries in heterogeneous SoCs usually execute independently, leading to frequent contentions in the interconnect, and so significantly damaging the system *predictability*.

Moreover, the FIFO-based design leads to occurrences of ***physical priority inversion***. 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. Such blocking occurs in two scenarios:

Secondary-port blocking. Secondary-port blocking occurs between requests/responses issued from the same port. Taking Fig. 1 as an example, tasks executed on the same processor can cause frequent Secondary-port blocking when they keep accessing the interconnect concurrently.

Primary-port blocking. Primary-port blocking occurs between the requests/responses issued from different ports, but sent to the same destination. Taking Fig. 1 as an example, the processors in the core subsystem and PEs in the DDN HA can suffer frequent Primary-port blocking when they keep reading the *same* memory bank simultaneously.

As an increasing number of hardware elements are integrated into the modern heterogeneous SoCs, to solve the above real-time issues, both *dependency* and *scalability* must be taken into account, as they could further magnify the issues in the AXI interconnects.

Dependency. As shown in Fig. 2, an AXI interconnect *fully* connects the Primaries and Secondaries in the system. Therefore, blocking which occurs in one transaction path can cause or magnify blocking 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 [12], [17].

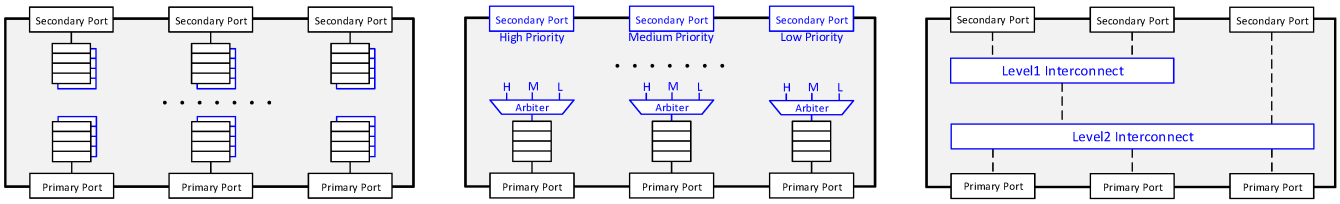
Scalability. With the increase in system complexity, modern SoCs always introduce additional Primaries and Secondaries, creating more transaction paths in the interconnect and bringing more data transferred between the Primaries and Secondaries. This adds significantly more resource contention/blocking in the interconnect and further magnifies the introduced issues [18].

These issues lead to challenges in designing a real-time AXI interconnect for modern heterogeneous SoCs.

3.2 Related Work

Existing work focusing on interconnect real-time performance can be mainly classified as *duplicated channels*, *bandwidth reservation*, and *hierarchical connections*. Note that communication protocols are not restricted in the review, as ideally, all these methods are compatible with AMBA AXI.

Duplicated channels (Fig. 4(a)). A straightforward way to improve interconnect real-time performance is by duplicating the communication channels. For example, Liao *et al.* [19] and Loh *et al.* [20] implement "virtual channels" for the interconnect transaction paths. In industrial patents, duplicated channels are also created for specific transactions, *e.g.*, secure messages [21], video processes [22], and I/O communication [23]. As evidenced in the evaluation [19],



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

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

[20], 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 Primary port, which further magnifies the problems reviewed in Sec. 3.1. At the same time, this method also significantly increases *hardware consumption*.

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 priority to each Primary (or Secondary port) and then allocates them a certain bandwidth based on their priorities. For instance, Restuccia *et al.* [24] introduce an AXI burst equalizer to re-organize transactions, ensuring that all Secondary ports are able to use the same bandwidth. Hebbache *et al.* [25] and Pagani *et al.* [13] propose a dedicated AXI controller to reserve bandwidth for high priority Primaries. Bandwidth reservation ensures throughput and predictability of certain Primaries. The method reduces *design flexibility* and *interconnect utilization*, since the interconnect must accurately achieve transaction information *before* run-time and always preserve sufficient bandwidth for the high-priority Primaries *at* run-time.

Hierarchical connections (Fig. 4(c)). There are also interconnects with multiple hierarchies. Specifically, Secondary ports are grouped into different partitions, and the Secondary ports in the same partition are connected to a local interconnect. At the same time, a global interconnect connects these local interconnects and Primary ports, enabling communication between the Primaries and Secondaries. For example, Audsley [26] introduces a tree-like structure to support multiple-level connections between local interconnects. Wang *et al.* [27] further extend this structure [26] to support 128 Primaries. This method brings partial optimizations to the interconnect. However, like the other methods, it is not able to solve the fundamental problems in Sec. 3.1.

4 *AXI-IC*$^{\mathrm{RT}}$: Overview

To guarantee the time-predictability of transactions, we present a new real-time AXI interconnect (*AXI-IC*$^{\mathrm{RT}}$), employing Random Access Queues (RAQs) and Transaction Control Units (TCUs) to buffer and schedule the transactions, respectively (again, Conventional AXI interconnects are designed using FIFO queues). The RAQs support random accesses of buffered transactions, prioritizing transactions based on their importance. The TCUs enable compositional scheduling at the hardware level, simultaneously ensuring the transactions' predictability and performance. Associated with the new hardware design, we present an interface selection algorithm (Sec. 6) and the associated analysis framework to find the optimized configurations of the *AXI-IC*$^{\mathrm{RT}}$, further optimizing its real-time performance.

4.1 Context

Based on the latest AMBA protocol (AMBA 5.0) [9], we assume that (i) each Primary has a unique ID (*i.e.*, PID); and, (ii) a single source of timing is used through the entire system, ensuring global synchronization. Here, as an example, we describe the design and guarantees for the write-related channels, i.e., AW, W, and B channels. This is because the protocol regulates independent Write and Read channels. In practice, we also built and analyzed the AR (R) channel using the same method as the AW (W) channel.

4.2 Design Concepts

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

DC 1: RAQ-based micro-architecture. *AXI-IC*$^{\mathrm{RT}}$ removes the FIFO queues used in conventional interconnects, presenting new *RAQ*s to buffer transactions. Unlike FIFO queues, RAQs support *random accesses* of buffered transactions, avoiding physical priority inversion and enabling transaction prioritization.

DC 2: Online monitoring and grouping. *AXI-IC*$^{\mathrm{RT}}$ monitors and decodes incoming transactions to extract the associated parameters. Based on these parameters, *AXI-IC*$^{\mathrm{RT}}$ groups transactions sent to the same destinations and schedules each group using a dedicated TCU. This prevents interference between transactions sent to different destinations.

DC 3: Real-time compositional scheduling. In light of the new architecture, a compositional scheduling policy can be implemented to ensure the predictability and scalability of this highly integrated system. Given the parameters of the task set executed by a Primary, the near-optimal transaction time of the corresponding TCU shared by the Primary will be calculated independently, and the transaction time shared by the other Primaries will not be altered.

4.3 Top-level Micro-architecture

Fig. 5 illustrates the top-level micro-architecture of *AXI-IC*$^{\mathrm{RT}}$. In the interfaces, we retain standard AXI ports to ensure *compatibility* with existing systems designed for conventional AXI interconnects.

In *request* path, we introduce an *AXI-decoder* connected to each *Secondary port*; two RAQs (for AW and W channels) and a TCU connected to each *Primary port*. During run-time, AXI-decoders monitor the AW channels. Once an AXI-decoder captures a transaction header, the AXI-decoder

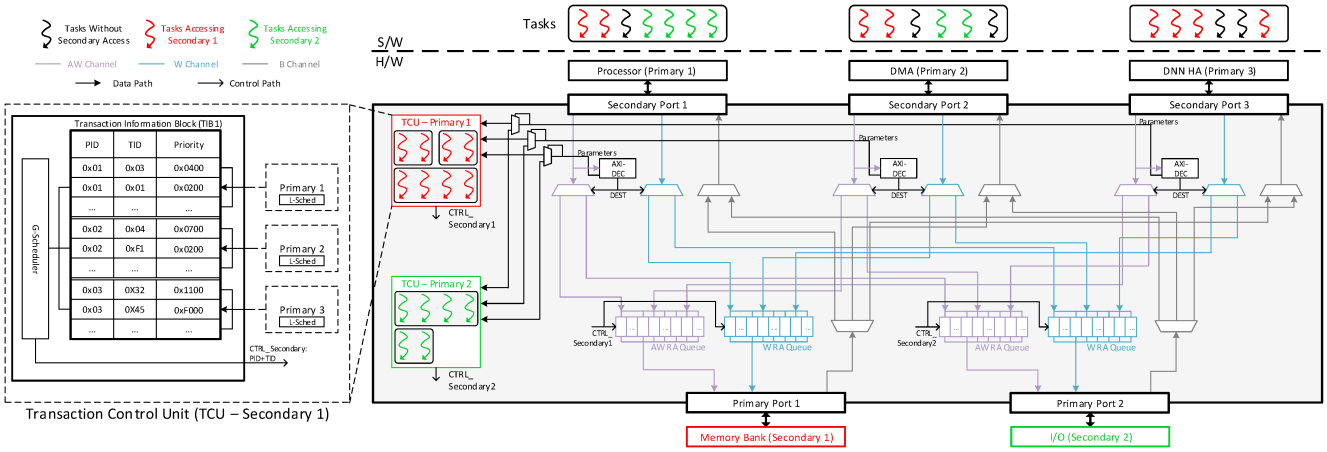


Fig. 5. Top-level micro-architecture of *AXI-IC*$^{\mathrm{RT}}$, showing AW, W and B channels (AXI-DEC: AXI-decoder; DEST: Destination; PID: Primary ID, TID: Transaction ID). AMBA AXI regulates independent write and read transactions; *AXI-IC*$^{\mathrm{RT}}$ contains a symmetric structure for AR(R) channels.

decomposes this header and extracts the transaction parameters. Based on the destination of the transaction, the AXI-decoder routes the transaction and its parameters to the corresponding RAQs and TCU, respectively. At the same time, the TCUs schedule the transactions for the Secondaries based on the transaction parameters.

In *response* path (*i.e.*, B channel), we adopt pass-through connections, because transactions always involve a *single-packet* response, and the Primaries (*e.g.*, processors) are faster than the Secondaries (*e.g.*, memory and I/Os). This leads to the transmission time in the B channel being negligible.

4.4 Scheduling Method: Compositional Scheduling

As modern computing systems are highly integrated, they need to be designed, implemented, and certified as several independent *components* (*i.e.*, Primaries), with each component having the "illusion" of executing on a dedicated *virtual* platform [28]–[30]. Thus, in a system using *AXI-IC*$^{\mathrm{RT}}$, transactions are scheduled in a hierarchical manner; a *Global Scheduler* (*G-Sched*) allocates the transaction time of the physical platform to each component and determines the characteristics of the virtual platform in each component. Each component then has a *Local scheduler* (*L-Sched*) to schedule that component's transactions on that virtual platform (VP).

In order to analyze the schedulability of each component, interfaces are required to characterize the *supply* provided by the VP, *i.e.*, the available time units for each transaction, obtained from the interconnect to support the transactions. For instance, the *periodic resource model* [31] is a fundamental interface which characterizes a VP using a pair of parameters (Π, Θ), with the interpretation that *at least* Θ time units of processor time are guaranteed to the supported task set every Π time units. The quotient Θ/Π is called the *bandwidth* of this VP. Given real-time tasks in the computing system, the optimal interface selection algorithm for each component will be provided in Sec. 6. In Sec. 5, we begin by showing how to implement the hierarchical scheduling method in the hardware.

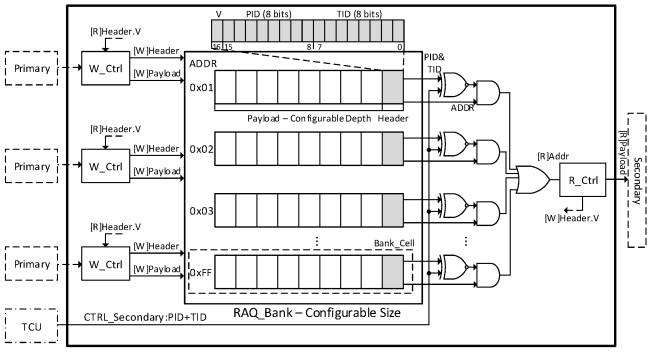


Fig. 6. Micro-architecture of a RAQ (TID: Transaction ID; W: write; R: read; W_Ctrl: write controller; R_Ctrl: read controller; V: Validness).

5 *AXI-IC*$^{\mathrm{RT}}$: Design

As discussed in Sec. 4, the *AXI-IC*$^{\mathrm{RT}}$ design is modularized, comprising three key elements: RAQs, AXI-decoders, and TCUs, which are further detailed in this section.

5.1 Random Access Queues (RAQs)

Memory is the most typical hardware element supporting random storage accesses. However, we cannot use memory for transaction buffers in an interconnect, as it has a significantly slower access speed with extra timing uncertainty compared to the FIFO queues in conventional interconnects. Thus, we designed RAQs to buffer transaction headers (in AW and AR channels) and bursts (in W and R channels). The design of the RAQ comprises a RAQ bank and multiple RAQ controllers (see Fig. 6).

RAQ bank. A *bank cell* is the essential element of a RAQ bank, with a unique address starting from 0x01.§ The bank cell design has two parts: a *payload FIFO* and a *cell header*. Specifically, the payload FIFO stores the transferred content (*i.e.*, a header or a burst of a transaction), and the cell header stores the transaction parameters, including TID (bits 0 - 7) and PID (bits 8 - 15). We also use the cell header's highest bit to indicate the validity of this bank cell.

§. We use the address 0x00, which indicates an invalid address.

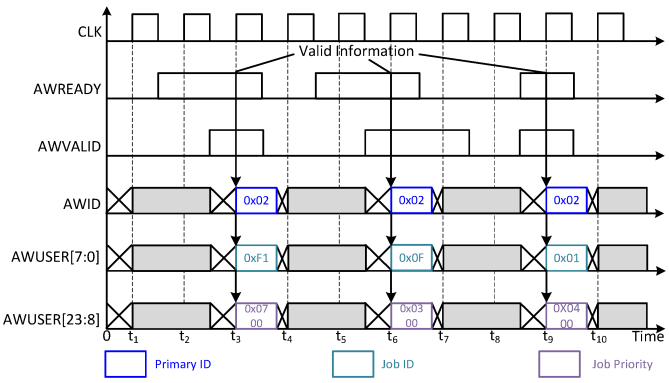


Fig. 7. Example of AW channel analysis in AMBA AXI.

The depth of a payload FIFO is configurable, providing flexibility for customization. For instance, we configure the payload FIFOs' depth as 1 for the AW and AR channels, as a transaction's header is always encapsulated in one packet.

RAQ controllers. We also present write/read controllers to store/fetch the content in the RAQ bank. A write/read controller contains two interfaces connected to the cell headers and payload FIFOs, respectively. In the *storing procedure*, the write controller first reads the cells' headers to check the cell validity (*i.e.*, *header.bit[16]*). If the controller finds an unused bank cell (*i.e.*, *header.bit[16]* == 0), it sets the cell *header.bit[16]* to 1 and then starts to *push* the transfer content to the payload FIFO. In the *fetching procedure*, the read controller pulls the content from a payload FIFO using its address, and then writes the cell's *header.bit[16]* to 0. As introduced in *DC-2* and *DC-3*, the *AXI-IC*$^{\text{RT}}$ relies on TCUs to schedule transactions, and the TCUs always return a transaction's TID and PID (see Sec. 5.3 for TCU design). Therefore, we also present a combinational logic circuit using the read controller and the RAQ bank to convert the IDs into the address of a specific bank cell in a fixed single clock cycle.

5.2 AXI-decoder

We first discuss online transaction monitoring and decomposition, which are the main concern of the AXI-decoders, followed by the design details of the AXI-decoder.

Online monitoring. Online monitoring captures each transferred transaction, and decomposition decodes the transaction's issuing and destination information.

As discussed in Sec. 2.2, an AXI transaction always initializes from the AW/AR channel, which presents the necessary information using a transaction header. Therefore, the AXI-decoder is only required to monitor these two channels. Here we give an example of monitoring and decomposition using the AW channel. The AW channel issues a transaction header by setting the AWVALID and AWREADY signals to 1, giving AWID validity to represent the PID and AWADDR the transaction destination.

To support the proposed scheduling method in Sec. 4.4, the TCU also requires the transaction TIDs and priority. However, the AMBA AXI protocol specification does not regulate specific signals for TIDs, and the only candidate signals for the priority (*i.e.*, AWQoS) contain only 4 bits – supporting up to 16 priority levels. Therefore, we use AWUSER to represent the TID bits $0-7$) and priority bits $8-23$). Note that, AMBA AXI 5.0 reserves AW/ARUSER signals for customization [9]. The example shown in Fig. 7 shows the Primary (PID: 0x02) initializing three transactions, with TID 0xF1, 0x0F, and 0x01. The transaction priorities are 0x0700, 0x0300, and 0x0400.

Transactions' priority. In practice, a software job may release multiple outstanding transactions continuously. To maintain a consistent transfer order for each software job's transactions, we introduce a 16-bit format for transaction priorities, with the high 8 bits inheriting the priority of the job; and the low 8 bits representing the transaction's offset. Using this format, we can unify the transfer order of transactions sent from the same Primary: (i) for transactions released by *different* jobs, the highest priority job's transaction is served first; (ii) for transactions released by the *same* job, the earliest released transaction is first.

AXI-decoder design (Fig. 8). The design of an AXI-decoder contains an address decoder, a register bank, and some combinational circuits.

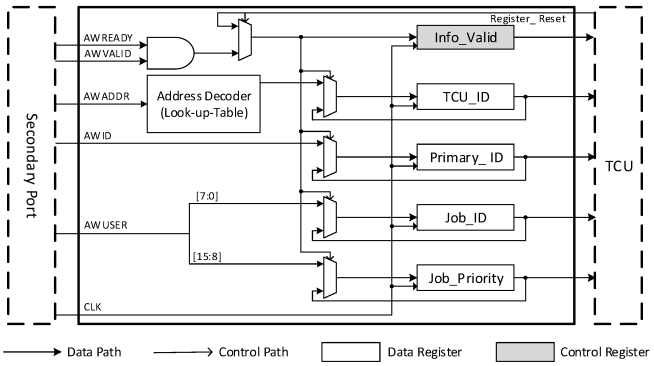


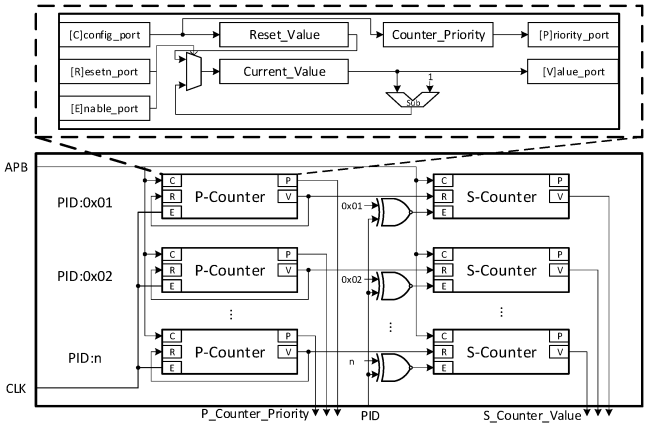
Fig. 8. Micro-architecture of an AXI-decoder.

The address decoder (implemented using a look-up-table) and the combinational circuits convert the AXI signals into transaction information using the method introduced above. Since we implement these modules without sequential logic, the AXI-decoder can always complete each decomposition in a fixed single clock cycle. The register bank contains four data registers and one control register. The data registers store the decomposed transfer information, and the control register determines when to load/reset the data registers.

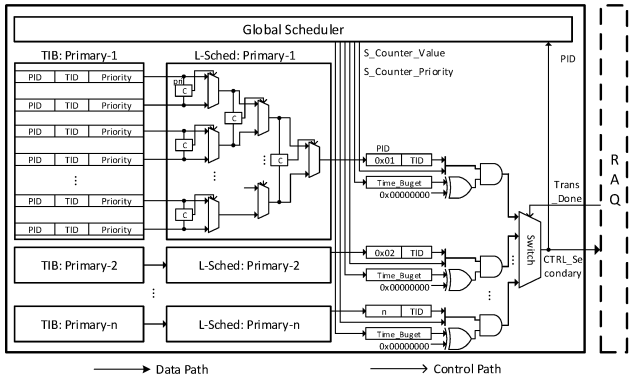
5.3 Transaction Control Unit (TCU)

TCUs are the brains of *AXI-IC*$^{\text{RT}}$ and determine the transaction order of the Secondaries they manage using the compositional scheduling method described in Sec. 4.4. The *global scheduler* (*G-Sched*) and the *local schedulers* (*L-Scheds*) are the main components of the TCU.

G-Sched (Fig. 9(a)). The G-Sched manages scheduling at the global level. Specifically, we associate two countdown counters to each Primary (i), the Server Counter (S-Counter) and the Period Counter (P-Counter). The S-Counter stores the Primary's transaction budget (Θ_i) and the P-Counter manages its refresh period (Π_i). We introduce the same micro-architecture for all counters; a counter has three registers which store the counter's reset value, current value and



(a) Global scheduler
(P_Counter: period counter; S_Counter: server counter).



(b) Local scheduler and other logic (C: priority comparator).

Fig. 9. Micro-architecture of a TCU.

priority. The counter *interfaces*, have three input ports and two output ports. The input ports are used to configure, reset, and enable the counter, and the output ports return the counter's current value and priority. Note that, the P-Counters' priority ports are only used to present the Primary priorities and tie off the S-Counter priority ports. During run-time, the current value of a counter is reset when its reset port equals 0 (*i.e.*, *active low*). It reduces the current value by one when its enable port meets a *rising edge*. To reset the counters for Primary i every Π_i, the P-Counter's value output is connected to its reset ports and its associated S-counter. Moreover, a counter's reset value and priority can also be updated online using its configure port.

L-Scheds (Fig 9(b)). To support scheduling in the local level, we introduce a dedicated Transaction Information Block (TIB) and an L-Sched to each Primary. Specifically, the TIB records the transaction parameters decomposed by the AXI-decoder, including PID, TID and priority; the L-Sched compares the transaction priorities and always returns the TID of the transaction with the highest priority. We designed the L-Sched using only combinational logic, ensuring scheduling in the local layer is always completed in a fixed single clock cycle.

Finally, we present a switch to collect the scheduling results from G-Sched and L-Scheds. The switch returns the PID and TID of the transaction with the highest priority which was transferred by the Primary with enough transaction budget.

Above we discuss the *AXI-IC*$^{\text{RT}}$ design. In the next section, we introduce a pseudo-polynomial time algorithm for the problem of selecting both a period (Π_i) and budget (Θ_i) for components consisting of sporadic task systems under compositional scheduling.

6 An Interface Selection Algorithm

Our two-layer scheduler is designed to allocate free time slots to the transactions in a hierarchical manner, which is shown in Fig. 5. In the global layer, each Primary issues a sequence of transactions. These transactions are processed in the available time slots are allocated to n Primaries. We assume that each time slot's duration equals the length of a transaction. The worst-case processing time of a transaction is denoted by q. To facilitate the hierarchical allocation in our architecture, each Primary i ($1 \leq i \leq n$) is supported by a periodic server task $\Gamma_i = (\Pi_i, \Theta_i)$ with the interpretation that the server task is invoked every Π_i time slots and receives at least Θ_i time slots between consecutive invocations. We assume that both Π_i and Θ_i parameters are multiples of q. The transactions from Primary i will be executed using the time slots received by Primary i. The transactions are modeled by a set of sporadic tasks, each of which is denoted $\tau_h = (T_h, C_h, D_h)$. τ_h releases a sequence of *jobs*, with minimum separation of T_h time slots, where each job consists of a batch of consecutive transactions and completes within C_h time slots of execution. Each job has a deadline at D_h time slots after it is released. According to the hardware implementation of the local scheduler, it performs the transactions one-by-one. Thus, *the earliest time instant when a released job can be scheduled for execution is the latest time instant when the local scheduler performs a new transaction.* We assume *constrained deadlines*, *i.e.*, $\forall h, D_h \leq T_h$. Let $\mathcal{T}_i$ denote the task set in Primary i, *i.e.*, $\tau_h \in \mathcal{T}_i$ means task τ_h is in Primary i. A job of any task can only be preempted between transactions. The utilization of task τ_h is defined by $u_h = C_h/T_h$, and we let U_i denote the total utilization of the task set in Primary i, *i.e.*, $U_i = \sum_{\tau_h \in \mathcal{T}_i} u_h$. The *demand bound function* $\mathsf{dbf}_h(t)$ gives the maximum *demand* of task τ_h in any time interval of length t, where the demand is defined by the total work required to complete jobs that have both release times and deadlines within the time interval, and can be calculated as follows [32]:

$$\mathsf{dbf}_h(t) = \max\left\{\left(\left\lfloor \frac{t - D_h}{T_h} \right\rfloor + 1\right) C_h, 0\right\}. \quad (1)$$

We also denote the total demand of all tasks in Primary i by

$$\mathsf{DBF}_i(t) = \sum_{\tau_h \in \mathcal{T}_i} \mathsf{dbf}_h(t). \quad (2)$$

In light of the server task model, the parameters Π_i and Θ_i are respectively referred to as the period and capacity of the server task Γ_i. The ratio of the budget and server period represents the interface *bandwidth* of the server task, denoted as $w_i = \Theta_i/\Pi_i$. A system-level scheduling algorithm allocates the transaction time among the different periodic server tasks that share the same Secondary, such that each server task receives (for every period) aggregate transaction time equivalent to its capacity. If the system-level scheduling

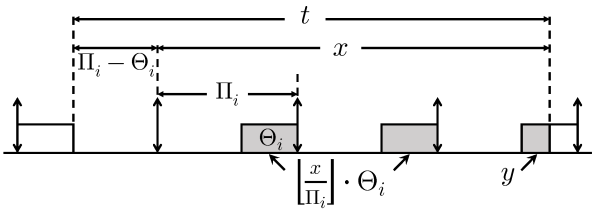


Fig. 10. Worst-case supply of a periodic resource.

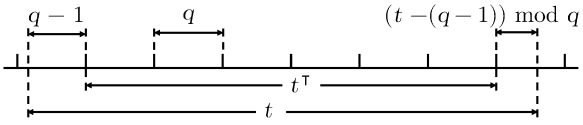


Fig. 11. Maximum interval that consists of integral transactions.

algorithm is Earliest-Deadline-First (EDF), then it is known (e.g., see [33]) that periodic server tasks $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$ can successfully guarantee the capacity parameters to their respective components, if and only if, the total system bandwidth does not exceed one, *i.e.*, $\sum_{i=1}^{m} w_i \leq 1$.

6.1 Schedulability Test

In this subsection, we provide a schedulability test that determines whether a task set $\mathcal{T}_i$ is schedulable for a given resource period Π_i and a given budget Θ_i.

The *supply bound function (SBF)* of a periodic server task, denoted $\mathsf{sbf}(t)$, indicates the *minimum* interconnect time this periodic server task can supply during any time interval of length t. Shin and Lee [33] have shown that $\mathsf{sbf}(\Theta_i, \Pi_i, t)$ can be calculated by

$$\mathsf{sbf}(\Theta_i, \Pi_i, t) = \begin{cases} 0 & \text{if } x < 0 \\ \left\lfloor \frac{x}{\Pi_i} \right\rfloor \cdot \Theta_i + y & \text{if } x \geq 0 \end{cases}$$

where x and y are for notational simplicity and are defined by

$$x = t - (\Pi_i - \Theta_i),$$
$$y = \max\left\{ x - \left\lfloor \frac{x}{\Pi_i} \right\rfloor \Pi_i - (\Pi_i - \Theta_i), 0 \right\}.$$

This definition reflects the worst-case scenario illustrated in Fig. 10. In *AXI-IC*$^{\mathrm{RT}}$, the execution is transaction based and therefore partial transactions do not contribute to the supply of a time interval. Consequently, the worst-case supply for any time interval of length t in *AXI-IC*$^{\mathrm{RT}}$ is calculated as follows, the scenario for which is illustrated in Fig. 11.

$$\mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t) = \mathsf{sbf}(\Theta_i, \Pi_i, t^\mathsf{T}),$$

where

$$t^\mathsf{T} = t - (q-1) - \Big((t-(q-1)) \mod q \Big)$$
$$= \left\lfloor \frac{t-(q-1)}{q} \right\rfloor q = \left(\left\lfloor \frac{t+1}{q} \right\rfloor - 1 \right) q.$$

Thus, we can derive the following SBF that takes the effect of transactions in account.

$$\mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t) = \begin{cases} 0 & \text{if } x^\mathsf{T} < 0 \\ \left\lfloor \frac{x^\mathsf{T}}{\Pi_i} \right\rfloor \cdot \Theta_i + y^\mathsf{T} & \text{if } x^\mathsf{T} \geq 0 \end{cases} \quad (3)$$

where x^T and y^T are for notational simplicity and defined by

$$x^\mathsf{T} = \left(\left\lfloor \frac{t+1}{q} \right\rfloor - 1 \right) q - (\Pi_i - \Theta_i),$$
$$y^\mathsf{T} = \max\left\{ x^\mathsf{T} - \left\lfloor \frac{x^\mathsf{T}}{\Pi_i} \right\rfloor \Pi_i - (\Pi_i - \Theta_i), 0 \right\}.$$

Theorem 1. *All tasks in Primary i must meet their deadlines if and only if*

$$\forall t, \mathsf{DBF}_i(t) \leq \mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t). \quad (4)$$

Proof. We prove the equivalent statement: some deadline is missed if and only if $\exists t$ such that $\mathsf{DBF}_i(t) > \mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t)$. Let t_d denote the earliest deadline that is missed. Let Ψ denote the set of jobs with deadline at or before t_d. Let t_0 denote the latest time instant before t_d such that all jobs in Ψ released at or before t_0 have finished by t_0. As a result, a job in Ψ must be released at t_0 and at any time instant during $[t_0, t_d]$, there must be some pending job in Ψ. Therefore, during $[t_0, t_d]$, the completed work for jobs in Ψ is $\mathsf{sbf}^\mathsf{T}(t_d - t_0)$ even in the worst case. On the other hand, by the definition of t_0, such work must be from jobs released at or after t_0 and therefore the amount of such work is up to $\mathsf{DBF}(t_d - t_0)$. Therefore, the deadline at t_d being missed if and only if $\mathsf{DBF}(t_d - t_0) > \mathsf{sbf}^\mathsf{T}(t_d - t_0)$. Letting $t = t_d - t_0$, the theorem follows. □

We next show an upper-bound on the t to be examined for applying Thm. 1 and thereby derive a schedulability test that runs in pseudo-polynomial time.

Lemma 1. *If $U_i < w_i$, then $\forall t \geq \frac{\Upsilon_i U_i + 2(q-1+\Pi_i-\Theta_i)w_i}{w_i - U_i}$, $\mathsf{DBF}_i(t) \leq \mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t)$, where $\Upsilon_i = \max_{\tau_h \in \mathcal{T}_i}\{T_h - D_h\}$.*

Proof. We denote task set $\mathcal{T}_i'(t) = \{\tau_h \mid \tau_h \in \mathcal{T}_i \wedge D_h \leq t\}$. It is clear that $\forall t, \mathcal{T}_i'(t) \subseteq \mathcal{T}_i$. Then, we have

$$\begin{aligned} \mathsf{DBF}_i(t) &= \sum_{\tau_h \in \mathcal{T}_i^*(t)} \left(\left\lfloor \frac{t - D_h}{T_h} \right\rfloor + 1 \right) C_h + \sum_{\tau_h \in \mathcal{T}_i \setminus \mathcal{T}_i^*(t)} 0 \\ &\leq \sum_{\tau_h \in \mathcal{T}_i'(t)} \left(\frac{t - D_h}{T_h} + 1 \right) C_h \\ &\leq (t + \max_{\tau_h \in \mathcal{T}_i'(t)}\{T_h - D_h\}) \times \sum_{\tau_h \in \mathcal{T}_i'(t)} u_h \\ &\leq (t + \max_{\tau_h \in \mathcal{T}_i}\{T_h - D_h\}) \times \sum_{\tau_h \in \mathcal{T}_i} u_h \\ &= (t + \Upsilon_i) U_i. \end{aligned}$$

On the other hand, by Eq. (3), we have

$$\begin{aligned} \mathsf{sbf}^\mathsf{T}(\Theta_i, \Pi_i, t) &\geq \Big(t^\mathsf{T} - 2(\Pi_i - \Theta_i) \Big) w_i \\ &\geq \Big(t - 2(q-1) - 2(\Pi_i - \Theta_i) \Big) w_i, \end{aligned}$$

where the second "$\geq$" takes "$=$" when $t = \ell q + (q-2), \ell \in \mathbb{N}$. Therefore,

$$\begin{aligned}
&\mathsf{DBF}_i(t) \leq \mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, t) \\
\Leftarrow\ &(t + \Upsilon_i)U_i \leq \Big(t - 2(q-1) - 2(\Pi_i - \Theta_i)\Big)w_i \\
\Leftarrow\ &U_i < w_i \wedge t \geq \frac{\Upsilon_i U_i + 2(q - 1 + \Pi_i - \Theta_i)w_i}{w_i - U_i}.
\end{aligned}$$

The lemma follows. □

Lemma 2. *$U_i < w_i$ is necessary for* (4) *to be true.*

Proof. We let $H = \mathrm{lcm}_{\tau_h \in \mathcal{T}_i}\{T_h\}$ denote the hyper-period of all tasks in Primary i. Then, due to constrained deadlines, $\mathsf{DBF}_i(H) = H \times U_i$. On the other hand, by (3) (or, by observing Figs. 10 and 11), we have $\forall t, \mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, t) < t \times w_i$, which implies $\mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, H) < H \times w_i$. Thus, $U_i < w_i$, or $H \times U_i < H \times w_i$, is necessary for $\mathsf{DBF}_i(H) \leq \mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, H)$, which is necessary for (4). □

By Thm. 1 and Lems. 1 and 2, the following theorem holds, which implies a pseudo-polynomial-time schedulability test.

Theorem 2. *All tasks in Primary i must meet their deadlines if and only if $U_i < w_i$ and*

$$\forall t < \mathcal{Z}, \mathsf{DBF}_i(t) \leq \mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, t),$$

where $\mathcal{Z} = \frac{\Upsilon_i U_i + 2(q-1+\Pi_i-\Theta_i)w_i}{w_i - U_i}$ and $\Upsilon_i = \max_{\tau_h \in \mathcal{T}_i}\{T_h - D_h\}$.

6.2 Selection Range for Feasible Periods

In this subsection, we provide a bounded range for choosing the resource period Π_i for each server task Γ_i. This is a basis for our next step to determine the interface for each Primary i.

Lemma 3. *The period of the periodic resource server task Γ_i for task set $\mathcal{T}_i$ in Primary i must be upper bounded by*

$$\frac{q}{1 - \sum_{j \neq i} U_j} \leq \Pi_i \leq \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{2\sum_{j \neq i} U_j}$$

Proof. It is clear that, for each Primary j, the bandwidth of its periodic resource server must be at least the total utilization of the task set in it, *i.e.*, $\forall j, w_j \geq U_j$. On the other hand, in order to be feasible, the total system bandwidth of all periodic resource servers cannot exceed one, *i.e.*, $\sum_j w_j \leq 1$. Therefore, it is *necessary* that

$$w_i \leq 1 - \sum_{j \neq i} w_j \leq 1 - \sum_{j \neq i} U_j.$$

Then, focusing on Primary i, according to the periodic resource model, it may have time intervals up to length of $2(\Pi_i - \Theta_i) = 2\Pi_i(1 - w_i)$ that provide no budget to task set $\mathcal{T}_i$. Therefore, to be schedulable in the worst case, it is *necessary* for each task $\tau_h \in \mathcal{T}_i$ that $2\Pi_i(1 - w_i) + C_h \leq D_h$, *i.e.*, $2\Pi_i(1 - w_i) \leq \min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}$. Thus,

$$\Pi_i \leq \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{2(1 - w_i)} \leq \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{2\sum_{j \neq i} U_j}.$$

On the other hand, because Θ_i must be multiple of transactions, *i.e.*, multiple of q, we have $\Theta_i \geq q$. Due to $\frac{\Theta_i}{\Pi_i} = w_i \leq 1 - \sum_{j \neq i} U_j$ as we shown earlier in this proof, we have

$$\Pi_i \geq \frac{\Theta_i}{1 - \sum_{j \neq i} U_j} \geq \frac{q}{1 - \sum_{j \neq i} U_j}.$$

Thus, the lemma follows. □

Corollary 1. *The resource period Π_i for Primary i must be selected as $\Pi_i = \ell \times q$ for some integer ℓ that is within the following range*

$$\left\lceil \frac{1}{1 - \sum_{j \neq i} U_j} \right\rceil \leq \ell \leq \left\lfloor \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{q \times 2\sum_{j \neq i} U_j} \right\rfloor.$$

Proof. This corollary directly follows from Lem. 3, given that Π_i must be a multiple of q. $\lceil a/q \rceil \times q$ is the smallest multiple of q that is greater than or equal to a, and $\lfloor b/q \rfloor \times q$ is the largest multiple of q that is less than or equal to b. □

6.3 Interface Selection

In this subsection, we present our interface selection algorithm that provides the pair of (Π_i, Θ_i) to each Primary i.

An interconnect's functionality, as we know it, is routing the requests and responses between the Primaries and the Secondaries. To ensure the system's real-time performance, Algorithms 1-3 are proposed to determine the interface for each Primary, *i.e.*, the scheduling parameters for each server task. The pseudo-code is presented in Alg. 1, where the subroutine MinBudget is described in Alg. 2 in which the subroutine SchedTest is described in Alg. 3. In particular, by Lem. 3, it is assigned that $\Pi_{\min} = \left\lceil \frac{1}{1 - \sum_{j \neq i} U_j} \right\rceil \times q$ and $\Pi_{\max} = \left\lfloor \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{q \times 2\sum_{j \neq i} U_j} \right\rfloor \times q$ when calling the function InterfaceSelect for each Primary i.

The goal of Alg. 1 is to find an interface pair (Π_i, Θ_i) that guarantees the deadlines of all tasks in Primary i to be met while minimizing the bandwidth $w_i = \Theta_i/\Pi_i$. Specifically, Lines 2 – 11 iterate all multiples of q in the range $[\Pi_{\min}, \Pi_{\max}]$ for potential selection of Π_i, and Line 4 calls subroutine MinBudget, as described in Alg. 2, to find the minimum required (for guaranteeing schedulability) budget for each given Π_i selection in a binary-search manner. Please note that, the input Π_i for MinBudget (Alg. 2) is supposed to be a multiple of q, and it returns a multiple of q as well. SchedTest$(\mathcal{T}_i, \Pi_i, \Theta_i)$ is the schedulability test as proven in Thm. 2, for any given task set $\mathcal{T}_i$ and interface pair (Π_i, Θ_i).

Time Complexity. Although the range of Π_i given by Lem. 3 depends on the total utilization in other Primaries (U_j), we can further see that this range must be a sub-interval of $\left[q, \frac{\min_{\tau_h \in \mathcal{T}_i}\{D_h - C_h\}}{2c_1}\right]$ if there exists some positive constant $c_1 \leq U_j, \forall j$. That is, there are pseudo-polynomial number of iterations in InterfaceSelect (Alg. 1) for systems with at least two Primaries and the task utilization in each Primary is lower-bounded by some constant $c_1 > 0$ (*e.g.*, even $c_1 = 0.0001$ is fine). Due to the logarithmic complexity of binary search, MinBudget (Alg. 2) has polynomial number of iterations. SchedTest (Alg. 3) has a pseudo-polynomial

Algorithm 1: InterfaceSelect($\mathcal{T}_i, \Pi_{\min}, \Pi_{\max}$)

$w_i \leftarrow \infty$
$\pi \leftarrow \Pi_{\min}$
repeat
  $\theta = \mathsf{MinBudget}(\mathcal{T}_i, \pi)$
  if $\theta/\pi < w_i$ **then**
    $w_i \leftarrow \theta/\pi$
    $\Pi_i \leftarrow \pi$
    $\Theta_i \leftarrow \theta$
  end if
  $\pi \leftarrow \pi + q$
until $\pi > \Pi_{\max}$
if $w_i \neq \infty$ **then**
  return (Π_i, Θ_i)
else
  return FAILURE
end if

Algorithm 2: MinBudget($\mathcal{T}_i, \Pi_i$)

$\mathsf{lo} \leftarrow 1$
$\mathsf{hi} \leftarrow \Pi_i/q$
if $\mathsf{SchedTest}(\mathcal{T}_i, \Pi_i, \Pi_i) =$ **false then**
  return ∞
end if
while $\mathsf{lo} < \mathsf{hi}$ **do**
  $\mathsf{mid} \leftarrow \lfloor(\mathsf{lo} + \mathsf{hi})/2\rfloor$
  if $\mathsf{SchedTest}(\mathcal{T}_i, \Pi_i, \mathsf{mid} \times q) =$ **true then**
    $\mathsf{hi} \leftarrow \mathsf{mid}$
  else
    $\mathsf{lo} \leftarrow \mathsf{mid} + 1$
  end if
end while
return $\mathsf{hi} \times q$

Algorithm 3: SchedTest($\mathcal{T}_i, \Pi_i, \Theta_i$)

if $U_i \geq w_i$ **then**
  return false
end if
$\Upsilon_i \leftarrow \max_{\tau_h \in \mathcal{T}_i}\{T_h - D_h\}$
$\mathcal{Z} \leftarrow (\Upsilon_i U_i + 2(q - 1 + \Pi_i - \Theta_i)w_i)/(w_i - U_i)$
$t \leftarrow \min_{\tau_h \in \mathcal{T}_i}\{D_h\}$
repeat
  if $\mathsf{DBF}_i(t) > \mathsf{sbf}^{\mathsf{T}}(\Theta_i, \Pi_i, t)$ **then**
    return false
  end if
  $t \leftarrow t + 1$
until $t \geq \mathcal{Z}$
return true

time complexity for any Primary i such that $w_i - U_i$ is lower-bounded by some constant $c_2 > 0$ (*e.g.*, even $c_2 = 0.0001$ is fine). To sum up, the time complexity of the interface selection algorithm is the product of a pseudo-polynomial, a polynomial, and a pseudo-polynomial, which is still in pseudo-polynomial time.

7 Evaluation

AXI-IC$^{\mathrm{RT}}$ is examined through system implementations.

7.1 Experimental Platform

We built a heterogeneous SoC (introduced in Fig. 1) on a Xilinx VC709 evaluation board. For the core subsystem, we implemented 16 MicroBlaze processors [34], with FreeRTOS (v.10.4) as the OS kernel for all processors [35]. At the same time, we instantiated two DNN HAs, following the IPs' default settings to present 9 MAC PEs in each DNN HA.

We implemented the AXI interconnect in the SoC using a traditional design (**AXI-TD**), duplicated channels (**AXI-DC**), reserved bandwidth (**AXI-RB**), hierarchical connections (**AXI-HC**), and *AXI-IC*$^{\mathrm{RT}}$. For AXI-TD and *AXI-IC*$^{\mathrm{RT}}$, we used the design methods described in Sec. 2.2 and Sec. 5 respectively. In *AXI-IC*$^{\mathrm{RT}}$, the interface parameters were obtained using the algorithms provided in Sec. 6. For each of the other interconnects, different variants are demonstrated in the literature. Our implementations of the other interconnects looked to find common ground representing their key characteristics, as follows:

AXI-DC (Fig. 4(a)). A "virtual channel" to all the communication paths was implemented. A transaction was transferred using the channel with the fewest buffered transactions.

AXI-RB (Fig. 4(b)). Equal importance was assumed for Primaries; identical bandwidths were assigned to each Primary.

AXI-HC (Fig. 4(c)). The Primaries were grouped into four partitions and connected to a local interconnect. At the same time, the local interconnects and the Secondaries were connected to a global interconnect.

All interconnects were implemented using BlueSpec System Verilog [36] and compiled into Verilog. The hardware in the system was synthesized and deployed using Vivado (v2020.2) [15], and the software executing on the processors (OS kernels, drivers and user applications) was compiled using the Xilinx MicroBlaze GNU tool-chain [34].

7.2 Hardware Overhead

Experimental setup. We first configured the AXI interconnects to support 8/16 outstanding transactions for each Primary/Secondary and compared the interconnects' hardware overhead in terms of Look-Up-Tables (LUTs), registers, DSPs, RAMs, and power. Using this configuration, the traffic pressure was considered close to practical applications and able to facilitate *overhead* observation. We then compared the *AXI-IC*$^{\mathrm{RT}}$ against other hardware elements in the system, *i.e.*, general-purpose processors (MicroBlaze and RISC-V) and the entire SoC, to examine *AXI-IC*$^{\mathrm{RT}}$'s hardware overhead from a system perspective. The MicroBlaze was full-featured, enabling all performance related functionalities (e.g., pipeline and data cache). The RISC-V was implemented based on [37], supporting all functionalities of the MicroBlaze, as well as multi-branch, out-of-order processing and related functionalities (e.g., branch-prediction). The SoC is introduced in Sec. 2.1, excluding the AXI interconnect.

Obs 1. *AXI-IC*$^{\mathrm{RT}}$ consumed similar hardware resources compared to existing AXI interconnects.

TABLE 1
Hardware overhead (implemented on FPGA)

	LUTs	Registers	DSPs	RAMs(KB)	Power(mW)
Proposed	4,745	4,184	0	0	44
AXI-TD	3,785	4,839	0	0	39
AXI-DC	4,538	7,588	0	0	58
AXI-RB	4,342	4,795	0	0	47
AXI-HC	9,829	5,593	0	0	68
MicroBlazes	4,908	4,385	6	256	359
RISC-V	7,432	16,321	21	512	583
SoC	141,718	113,627	174	16,384	2,904

This observation is shown by comparing the interconnects' hardware overhead presented in Table 1. *AXI-IC*$^{\mathrm{RT}}$ required fewer registers than other AXI-interconnects. Specifically, it consumed 665 (13.5%) fewer registers than AXI-TD, 3,404 (44.9%) fewer than AXI-DC, 611 (12.7%) fewer than AXI-RB, and 1,409 (25.2%) fewer than AXI-HC. Such improvement benefited from deploying RAQs in *AXI-IC*$^{\mathrm{RT}}$. Unlike conventional interconnects implementing FIFO queues for each Primary and Secondary, *AXI-IC*$^{\mathrm{RT}}$ implements RAQs for transaction paths. This allows Primaries and Secondaries to share the RAQs in each transaction path. We found that *AXI-IC*$^{\mathrm{RT}}$ required slightly more LUTs than other interconnects, since *AXI-IC*$^{\mathrm{RT}}$ presents new AXI-decoders and TCUs, which are designed using combinational logic.

Obs 2. From a system perspective, the design of the *AXI-IC*$^{\mathrm{RT}}$ was resource-efficient.

As shown in Table 1, *AXI-IC*$^{\mathrm{RT}}$ consumed less hardware compared to the other main system elements. Specifically, compared to the MicroBlaze processor *AXI-IC*$^{\mathrm{RT}}$ required 163 LUTs, (3.3% of a MicroBlaze), 201 registers (4.6%), and 315 mW (87.7%). It also consumed 2,687 (36.2%) LUTs, 12,137 (74.4%) registers, and 539 (92.5%) mW less than the RISC-V processor. Over the entire SoC, the hardware required by *AXI-IC*$^{\mathrm{RT}}$ was less than 4% (for all metrics).

7.3 Synthetic Workloads: Transmission Efficiency and Real-time Performance

Experimental setup. We deployed 4/8/16 processors as transaction generators (Primaries) and implemented 4 dedicated transaction operators (Secondaries). We then connected them to the same interconnect. During experiments, a generator randomly generated 2-4 transactions for each operator and assigned a unique priority to each transaction. The operators acknowledged the Primaries for transactions without processing any data. The generator paused until it had no outstanding transactions and then started to reissue new transactions. Synthetic transaction workloads such as this provide traffic patterns close to practical applications and facilitate *behavior* observation. We examined interconnect transmission efficiency and real-time performance using the *propagation* and *blocking latency* of transactions. The propagation latency of a transaction records its response time from issue to completion. The blocking latency of a transaction indicates the duration of time it is blocked by transactions with lower priority. The experiments were executed 1,000 times.

Obs 3. *AXI-IC*$^{\mathrm{RT}}$ had the best *transmission efficiency* when workloads were not intensive. With an increase in workload intensity, this benefit decreased slightly.

This observation is shown in the comparison between the propagation latency of three experimental groups in Fig. 12. In experiments with 4 transaction generators, the transactions in *AXI-IC*$^{\mathrm{RT}}$ experienced the lowest propagation latency on average. To transmit a transaction through a certain communication channel, conventional AXI interconnects have to pass the transaction through the entire FIFO queue of this channel. The improvements benefited from employing RAQs in *AXI-IC*$^{\mathrm{RT}}$, enabling random accesses of transactions and providing shorter paths for the transactions. However, with increased workload intensity, the FIFO queues in the conventional AXI interconnects behave like a "pipeline", mitigating the drawbacks of the transmission efficiency.

Obs 4. *AXI-IC*$^{\mathrm{RT}}$ always achieved the best *real-time performance* when the system was scaled with different numbers of transaction generators.

This observation is summarized from two perspectives: (i) transactions in *AXI-IC*$^{\mathrm{RT}}$ always had the shortest blocking time. Such benefit was achieved by deploying RAQs in *AXI-IC*$^{\mathrm{RT}}$ (detailed in Sec. 5.1), which enabled prioritization of the transactions, avoiding a high-priority transaction being blocked by low-priority transactions. (ii) *AXI-IC*$^{\mathrm{RT}}$ always had the least experimental variance. This is because *AXI-IC*$^{\mathrm{RT}}$ deploys the real-time scheduler (TCU) at the hardware level (see Sec. 5.3), ensuring transactions are managed in a time-predicable manner.

7.4 Case Study

We now use a case study to examine the benefits of applying *AXI-IC*$^{\mathrm{RT}}$ in a heterogeneous SoC.

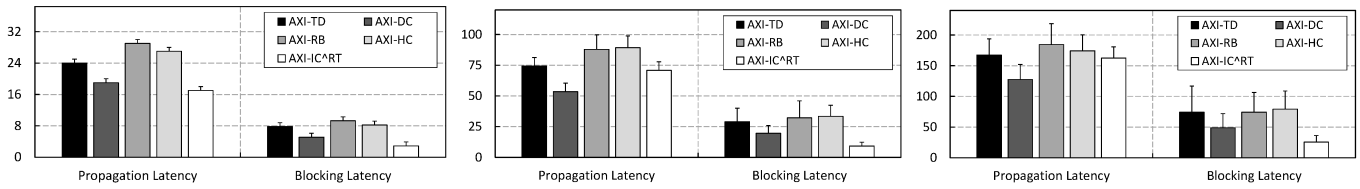
System configurations. We integrated the different AXI interconnects in the SoC (shown in Fig. 1). For each SoC, we executed real-world task sets using the core subsystem and DNN inference tasks using DNN HAs.

Task sets. We introduced three sets of real-world tasks for the core subsystem (20 tasks in each set):

- Safety tasks, selected from the Renesas automotive use case database [38], *e.g.*, CRC, RSA32, *etc.*
- Function tasks, selected from the EEMBC benchmark [39], *e.g.*, Fourier transform, speed calculation, *etc.*
- Synthetic workloads, selected from the EEMBC benchmark, which could be optionally added into the system to tune overall system utilization.

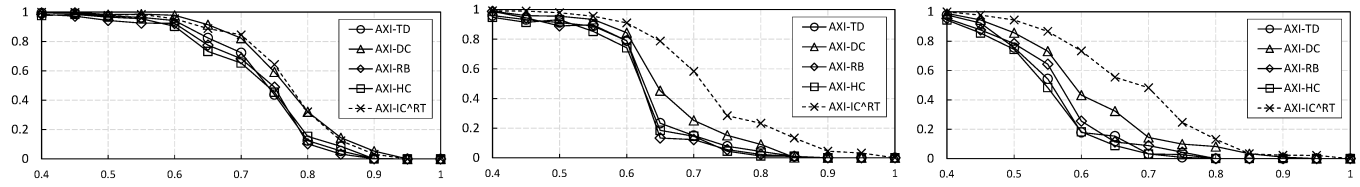
We employed a hybrid-measurement approach to obtain WCETs for all tasks [40]. The raw data processed by the 40 tasks was randomly generated off-chip and sent to the evaluated systems via two Ethernet controllers (1 Gbps). The results were sent back via a FlexRay controller (10 Mbps). Each task had a defined period and implicit deadline, with overall system utilization approximately 40%. Note, in practical systems, the execution time of a task is affected by diverse factors (e.g., cache miss rate); hence, adding synthetic workloads to a system only gives it a *target utilization*.

DNN inference tasks. We used two groups of DNN inference tasks for the DNN HAs, which were built on LeNet-5 [41] and AlexNet [42] architectures. Each group contained



(a) 4 generators and 4 operators. (b) 8 generators and 4 operators. (c) 16 generators and 4 operators.

Fig. 12. Synthetic workloads: average propagation and blocking latency (unit:μs). The error bar indicates the worst case in $1,000$ experiments.



(a) 4-core architecture. (b) 8-core architecture. (c) 16-core architecture.

Fig. 13. Case study: success ratios of different systems (x-axis: target utilization; y-axis: success ratio).

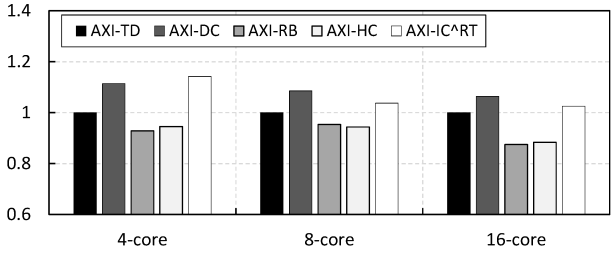


Fig. 14. Case study: average throughput of DNN HAs. The results are normalized by AXI-TD in each experimental group.

three tasks, which were trained using MNIST, EMNIST and CIFAR-10 training datasets. The testing datasets for task inferences were stored in off-chip DRAMs.

Experimental setup. We introduced three groups of experimental setups, which activated 4/8/16 processors in the core subsystem and executed the task sets and synthetic workloads. In each experimental group, we executed each examined system 1,000 times under varying target utilization from 40% to 100% (with an interval of 5%). Each execution lasted 150 seconds, which guaranteed that all tasks executed at least 300 times. At the same time, we also controlled the DNN HAs to continually fetch test datasets from DRAMs and execute the inference tasks in a *round-robin* manner. For fair comparison, we also ensured the data input to the examined systems was identical in each execution. We evaluated the examined systems using *success ratio* for the core subsystem and *throughput* for the DNN HAs. The *success ratio* recorded the percentage of trials that executed successfully (*i.e.*, without deadline misses of any safety or function software tasks) under a specified target utilization. This metric reveals system-level throughput because, as shown from synthetic benchmarks in section 7.3, as the success ratio increases more transactions can be correctly processed. This serves as an effective proxy for the system's overall predictability. The throughput evaluated the average execution times of the DNN tasks in each examined system.

Obs 5. With the increase in SoC complexity, interconnect became the dominant factor in the real-time performance.

This observation can be explained by the experimental results in Fig. 13. In the system with 4 processors (Fig. 13(a)), the success ratios of the examined systems were similar under each target utilization. In 8-core and 16-core systems (Fig. 13(b) and 13(c)), with more Primaries and Secondaries involved, the success ratios of the examined systems dropped significantly. As would be expected, this indicates that the interconnect dominated the SoC's real-time performance when the SoC's complexity increased.

Obs 6. In complex SoCs, applying *AXI-IC*$^{\mathrm{RT}}$ is beneficial.

As shown in Figs. 13(b) and 13(c), with the same configuration, the system with *AXI-IC*$^{\mathrm{RT}}$ always achieved higher success ratios compared to the baseline systems. Such improvements were acquired by: (i) introducing the new micro-architecture of *AXI-IC*$^{\mathrm{RT}}$ (see Sec. 5), enabling prioritization and real-time scheduling of transactions. (ii) deploying the interface selection algorithm (see Sec. 6), further optimizing the real-time performance of *AXI-IC*$^{\mathrm{RT}}$.

Obs 7. *AXI-IC*$^{\mathrm{RT}}$ improved the Primaries' throughput, although with a slight increase in SoC complexity.

This observation is shown in Fig. 14. In 4-core systems, DNN HAs executing in the system with *AXI-IC*$^{\mathrm{RT}}$ outperformed all other examined systems. In 8-core and 16-core systems, *AXI-IC*$^{\mathrm{RT}}$ ensured higher throughput of the DNN HAs compared to AXI-TD, AXI-RB, and AXI-HC, but was outperformed by AXI-DC. This observation aligns with experimental results using synthetic workloads, *i.e.*, Obs. 3.

8 Conclusion

This paper proposes a real-time interconnect (*AXI-IC*$^{\mathrm{RT}}$) for multi-/many-core heterogeneous SoCs. *AXI-IC*$^{\mathrm{RT}}$ introduces a novel micro-architecture, enabling random accesses of buffered transactions and prioritizing transactions using a dedicated two-layer compositional scheduler. To realize efficient compositional scheduling, we propose a pseudo-polynomial time algorithm to solve the problem of selecting both a period and capacity for components consisting of

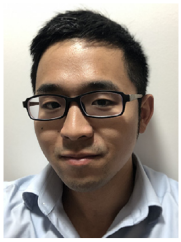
sporadic task systems. As shown in the evaluation, *AXI-IC*^{RT} outperforms the conventional interconnects, and the *AXI-IC*^{RT} design is resource-efficient.

9 Acknowledgment

We would like to thank the editors and anonymous reviewers for their helpful feedback. Especially, we would like to thank Dr. Dayu Shi for his kind and consistent help.

References

[1] I. ISO, "26262: Road vehicles-functional safety," 2018.
[2] S. Anthony, "Ibm unveils world's first 5nm chip," *Ars Technica*, 2017.
[3] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Modeling and analysis of bus contention for hardware accelerators in fpga socs," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
[4] D. Casini, P. Pazzaglia, A. Biondi, M. Di Natale, and G. Buttazzo, "Predictable memory-cpu co-scheduling with support for latency-sensitive tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[5] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Memory feasibility analysis of parallel tasks running on scratchpad-based architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 312–324.
[6] "Website: Tesla, FSD," https://www.tesla.com/en_GB/support/full-self-driving-computer.
[7] Z. Jiang, N. Audsley, D. Shill, K. Yang, N. Fisher, and Z. Dong, "Brief industry paper: Axi-interconnect rt: Towards a real-time axi-interconnect for system-on-chips," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 437–440.
[8] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 75–84.
[9] ARM, "AMBA AXI 5.0," https://developer.arm.com/architectures/system-architectures/amba/amba-5.
[10] "Xilinx Interconnect," https://www.xilinx.com/axi_interconnect.
[11] "Xilinx SmartConnect," https://www.xilinx.com/smart-connect.
[12] 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.
[13] 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 2019)*, 2019.
[14] M. D. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens, "A generic, scalable and globally arbitrated memory tree for shared dram access in real-time systems," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 193–198.
[15] "Website: Xilinx," https://www.xilinx.com/.
[16] G. Plumbridge, J. Whitham, and N. Audsley, "Blueshell: a platform for rapid prototyping of multiprocessor nocs and accelerators," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 107–117, 2014.
[17] D. Casini, A. Biondi, and G. Buttazzo, "Timing isolation and improved scheduling of deep neural networks for real-time systems," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1760–1777, 2020.
[18] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.
[19] X. Liao, J. Zhou, and X. Liu, "Exploring amba axi on-chip interconnection for tsv-based 3d socs," in *2011 IEEE International 3D Systems Integration Conference (3DIC), 2011 IEEE International*. IEEE, 2011, pp. 1–4.
[20] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, "Interconnect-memory challenges for multi-chip, silicon interposer systems," in *Proceedings of the 2015 international symposium on Memory Systems*, 2015, pp. 3–10.
[21] "Website: Arm trustzone," https://developer.arm.com/ip/trustzone.
[22] B. Parten, Y. Lee, B. Quach, L. Myers, W. Ray, and W. Maung, "Multi-channel peripheral interconnect supporting simultaneous video and bus protocols," Sep. 26 2017, uS Patent 9,772,965.
[23] P. R. Chandra, K. C. Kahn, E. Galil, E. Kugman, N. Zolotov, V. Yudovich, Y. Dishon, and E. Bagelman, "Multi-protocol tunneling over an i/o interconnect," Jul. 8 2014, uS Patent 8,775,713.
[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, pp. 1–22, 2019.
[25] F. Hebbache, M. Jan, F. Brandner, and L. Pautet, "Shedding the shackles of time-division multiplexing," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 456–468.
[26] N. Audsley, "Memory architecture for noc-based real-time mixed criticality systems," *Proc. WMC, RTSS*, pp. 37–42, 2013.
[27] H. Wang, N. C. Audsley, X. S. Hu, and W. Chang, "Meshed bluetree: Time-predictable multimemory interconnect for multi-core architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3787–3798, 2020.
[28] Z. Deng and J. Liu, "Scheduling real-time applications in an open environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, 1997, pp. 308–319.
[29] S. Dey, D. Sarkar, and A. Basu, "A tag machine based performance evaluation method for job-shop schedules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1028–1041, 2010.
[30] G. Gracioli and A. A. Fröhlich, "Two-phase colour-aware multicore real-time scheduler," *IET Computers & Digital Techniques*, vol. 11, no. 4, pp. 133–139, 2017.
[31] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, 2003, pp. 1–12.
[32] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *[1990] Proceedings 11th Real-Time Systems Symposium*. IEEE, 1990, pp. 182–190.
[33] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–39, 2008.
[34] Xilinx, "Microblaze," https://www.xilinx.com/products/microblaze.
[35] FreeRTOS, "FreeRTOS official website," http://www.freertos.org/.
[36] "Bluespec System Verilog," https://bluespec.com.
[37] S. Mashimo *et al.*, "An open source fpga-optimized out-of-order RISC-V soft processor," in *ICFPT*, 2019.
[38] R. Electronics, "Renesas: Automotive Use Cases," https://www.renesas.com/solutions/automotive.html.
[39] EEMBC, "EEMBC benchmark," https://www.eembc.org/autobench/.
[40] S. Law, M. Bennett, and Hutchesson, "Effective worst-case execution time analysis of DO178C level a software." *Ada User Journal*, 2015.
[41] Y. LeCun, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.



Zhe Jiang received his Ph.D. from the University of York (2019). He is currently working as a system design engineer in the Central Engineering Department at ARM Ltd and is a visiting research associate at the University of York. His research interests include safety-critical system, system architecture, and system micro-architecture. He can be reached at: zhe.jiang@arm.com or zhe.jiang@york.ac.uk.

Kecheng Yang received his BEng degree in computer science and technology from Hunan University in 2013, and his MSc and PhD degrees from the University of North Carolina at Chapel Hill in 2015 and 2018, respectively. He is an assistant professor in the Department of Computer Science at Texas State University. His research interests include real-time systems and scheduling algorithms. He received an Outstanding Paper Award and the Best Student Paper Award at the 40th IEEE RTSS, and an Oustanding Paper Award at the 26th RTNS.

Nathan Fisher received his Ph.D. from the University of North Carolina at Chapel Hill in 2007, his M.Sc. degree from Columbia University in 2002, and B.Sc. degree from the University of Minnesota in 1999, all in computer science. He is an Associate Professor with the Department of Computer Science, Wayne State University, Detroit, MI, USA. His research interests include real-time and embedded computer systems, sustainable computing, resource allocation, game-theory, and approximation algorithms. His current funded research projects are on composability of real-time applications, multiprocessor real-time scheduling theory, thermal-aware real-time system design, and algorithmic mechanism design in competitive real-time systems. Prof. Fisher was the recipient of the NSF CAREER Award in 2010 and has best paper awards from publication venues such as RTSS, ECRTS, and the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.

Ian Gray is an associate professor in the Department of Computer Science at the University of York. His research interests include real-time systems and their programming models, embedded systems, FPGAs and re-configurable computing, and many/multi-core distributed systems.

Neil C. Audsley is a professor in the Department of Computer Science at the University of York, where he leads a team researching Real-Time Embedded Systems. He is currently serving as the Head of Department of Computer Science. Specific areas of research include high performance real-time systems (including aspects of big data); real-time operating systems and their acceleration on FPGAs; real-time architectures, specifically memory hierarchies, Network-on-Chip and heterogeneous systems; scheduling, timing analysis and worst-case execution time; model-driven development. His research has been funded by a number of national (EPSRC) and European (EU) grants, including TEMPO, eMuCo, TouchHMore, MADES, JEOPARD, JUNIPER, T-CREST, DreamCloud and Phantom.

Zheng Dong received his BSc degree from Wuhan University, China, in 2007, MSc from the University of Science and Technology of China, in 2011, and PhD degree from the University of Texas at Dallas, USA, in 2019. He is an assistant professor with the Department of Computer Science, Wayne State University, Detroit, Michigan. His research interests are in real-time embedded computer systems and connected autonomous driving systems. His current research focus is on multiprocessor scheduling theory and hardware-software co-design for real-time applications. He received the Outstanding Paper Award at the 38th IEEE RTSS. He is a member of the IEEE Computer Society.