Tampere University

Aleksei Gimbitskii

INTERCONNECT DESIGN FOR THE EDGE COMPUTING SYSTEM-ON-CHIP

Master of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Examiners: Prof. Timo Hämäläinen
Sakari Lahti, M.Sc
April 2022

ABSTRACT

Aleksei Gimbitskii: Interconnect design for the edge computing system-on-chip
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
April 2022

Nowadays the majority of system-on-chips are designed by placing various IP blocks such as CPUs, memories and accelerators on the same chip. With the advantage of silicon manufacturing technologies, it has become possible to place hundreds of CPU cores and other design blocks on the same chip. A communication system that transfers data between chip components largely affects overall chip performance, computational speed and response time for external events.

Firstly, this thesis studies the main on-chip interconnect design paradigms. According to the presented research, various architectures may be chosen for an interconnect design depending on the required complexity and number of subsystems. The shared and hybrid bus interconnects are one of the oldest means of on-chip communication. They are efficient for small systems with no more than ten IP blocks. The crossbars or bus matrix interconnects can help to build on-chip communication systems which can efficiently interconnect dozens of system-on-chip modules. The networks-on-chip can provide a communication solution for large scale chip designs with hundreds of IP blocks.

The second part of this thesis focuses on the novel Ballast chip implementation and its interconnect design. The Ballast is a heterogeneous multiprocessor chip designed for edge computing and general-purpose computing applications. In this thesis Ballast interconnect was designed from scratch by using a cascaded crossbar approach by connecting three open-sourced AXI protocol bus matrices. The designed interconnect allows to efficiently connect 6 bus masters with 9 slaves and provides up to 9,6 GB/s bandwidth for the most productive CPU subsystem.

Keywords: interconnect, bus, crossbar, network-on-chip, SoC hub, Ballast chip

PREFACE

This thesis was written as part of SoC Hub project led by Tampere University and received funding from the Business Finland Veturi program.

I would like to thank my examiners professor Timo Hämäläinen and M.Sc. Sakari Lahti for their guidance while writing my thesis. Also, I want to thank the whole SoC Hub development team for their efforts in designing of the Ballast system-on-chip. Special thanks to Jarkko Passi and Mohamed Ibrahim for their contributions to the Ballast interconnect testing and verification and Antti Rautakoura and Tero Lehtinen for their feedback and help in the Ballast interconnect development.

Tampere, 9th April 2022

Aleksei Gimbitskii

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application-specific Integrated Circuit
AXI	Advanced eXtensible Interface
CDC	Clock-domain crossing
CDMA	Code division multiple access
ConfigICN	Configurational Interconnect
CPU	Central Processing Unit
DDR	Double Data Rate
DFT	Design for test
DLC	Data link control
DMA	Direct Memory Access
DSP	Digital signal processor
FIFO	First-in First-out
FPGA	Field-programmable gate array
GALS	Globally asynchronous locally synchronous
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HIBI	Heterogeneous IP Block Interconnection
HPC	High Performance Computing
HPICN	High-Performance Interconnect
HWPE	Hardware processing engine
I/O	Input-Output
IC	Integrated Circuit
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors

JTAG	Joint Test Action Group
LOC	Lines of code
LPICN	Low-Performance Interconnect
LUT	look-up table
MAC	Media access control
ML	Machine Learning
MMU	Memory management unit
MPC	Medium Performance Computing
MPEG	Moving Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
NoC	Network-on-Chip
NVIDLA	NVIDIA Deep Learning Accelerator
PE	Processing element
PLL	Phase-locked loop
PTW	Page table walker
RAM	Random Access Memory
ROM	Read-only Memory
RTL	Register-transfer logic
SDMA	Space division multiple access
SIMD	Single instruction multiple data
SoC	System-on-Chip
SPI	Serial peripheral interface
SRAM	Static Random-access Memory
TDMA	Time division multiple access
TLB	Look-aside buffer
TTA	Transport Triggered Architecture
UVM	Universal Verification Methodology

1. INTRODUCTION

In recent decades, advances in silicon design technologies made it possible to manufacture very large and complex computing systems on a single chip commonly named as System-on-Chip (SoC). Nowadays single SoC may contain dozens of billions of transistors and feature up to hundreds of heterogeneous components such as CPUs, GPUs, specific computing accelerators, on-chip memories, modems, radio antennas and input-output interfaces. All that complexity requires adequate communication architecture that provides reliable, scalable, power-efficient, high throughput and minimal latency interconnection between chip components.

According to the International Technology Roadmap for Semiconductors (ITRS) 2.0 [1], an interconnect design remains to be one of the key challenges in modern chips productization. ITRS identifies a set of key issues related to the on-chip interconnect design. Many of them are related to the material properties, chip manufacturing process and metrology. However, this thesis will concentrate on the second set of interconnect design challenges related to the components integration, logic design and architectural decisions and will not consider design aspects related to the manufacturing process.

There is a group of interconnect performance metrics and properties, that are most important for the evaluation of on-chip interconnect solution:

- The throughput of an interconnect heavily affects overall system performance and computing speed. An interconnect with limited throughput may become a bottleneck for the whole system because many modern computing applications generate heavy data traffic between execution units and memories.
- The interconnect latency may become an important performance and safety factor when SoC is supposed to be used in real-time embedded applications with strong requirements for external event response time.
- The interconnect power consumption may become a limiting factor for mobile and embedded applications. High power consumption may lead to short operational time, extensive heating and product cost increase.
- The logic utilization and chip area affect overall chip cost and power consumption.
- The scalability property of on-chip interconnect shows how easily an interconnect

architecture can be reused for the new chip production and how easily new components can be connected with existing interconnect with minimal impact on system performance.

- The number of reliability properties include routing algorithm robustness, ability to recover from errors and deadlock conditions.

An interconnect design requires careful analysis and compromise decisions because many of mentioned properties conflict with each other. For example, rising operational frequency to achieve higher throughput inevitably will lead to higher power consumption. Complex routing algorithms with additional reliability features will require a larger area of logic and will produce a longer critical path with lower clock frequency etc.

The first goal of this thesis is to study and analyse various interconnect design paradigms, architectures and their relation to the chip properties and performance. Chapter 2 will consider and analyse shared and hybrid buses, crossbar based interconnects and compare different bus arbitration algorithms. Chapter 3 will study the basics of the network-on-chip (NoC) design paradigm, NoC router structure and on-chip network topologies.

The second goal of this thesis is to use obtained theoretical knowledge to develop an interconnect solution for the novel Ballast chip. The Ballast chip is designed by the SoC Hub community[2] for edge computing and research applications. Due to the novelty of the Ballast chip, there are no specific requirements for the future interconnect performance, area or power consumption. The design goal is simply to achieve the best possible results. Chapter 4 will outline the general structure of the Ballast SoC and the functionality of its subsystems. Chapter 5 will describe the Ballast interconnect design process. Firstly, we will analyse expected interconnect data traffic and will use it as a design specification with some additional constraints. Then we will present the achieved interconnect architecture with some implementation details. Then Chapter 5 will describe methods used for Ballast interconnect verification. Lastly, Chapter 5 will outline some performance metric calculations such as achieved interconnect throughput and latencies. Chapter 6 will summarize the content of this thesis and will give a review of the achieved results.

2. BUS BASED INTERCONNECTS

2.1 Single and hierarchical bus architectures

Buses are one of the oldest and widely used means of communication between system-on-chip components. They provide very simple and efficient way of data transferring and have remained as preferred interconnection technology until these days.A bus connects several on-chip components with a single shared channel. Physically this channel may be implemented as one wire (serial bus) or a set of multiple wires (parallel bus). The parallel bus is the most typical choice for on-chip communication architectures.

The essence of any bus is a set of rules by which data moves between system components. Such rules are usually named bus protocol. It defines the duration and sequence of exchanged messages, data acknowledgement mechanisms, transferred data size etc. Bus-based architectures usually consist of one or more shared buses and logic components that implement the rules of used bus protocol.

Components that initiate and control data transfers are referred to as masters (for example CPU or digital signal processor). Every master component is connected to the bus using a set of signals which collectively form a master port. The components that simply respond to data transfer and cannot initiate transfers themselves are called slaves and have corresponding slave ports (for example on-chip RAM). Some components can have both master and slave ports. For instance, the direct memory access (DMA) component.

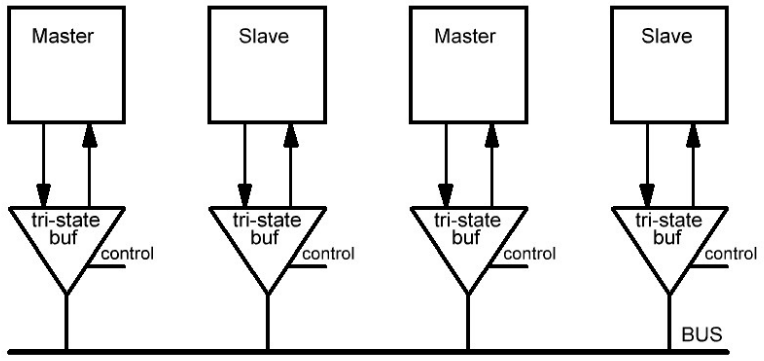
Bus signals can be classified into three main categories: address, data and control [3, p.20-21]. Some protocols have shared address bus for read and write transactions. However, some systems may have separate address buses for reading and writing. Multiple address buses improve concurrency because read and write transactions can occur in parallel but consume more area and power. Data signals are used to transmit data from master to slave during writes and opposite direction during reading transactions. Similar to address buses, interconnect architecture may use shared data bus or separate data buses for read and write transfers. Control signals are used to transmit protocol-specific data: request and acknowledge, data size and byte enable signals, burst length, master ID etc.

Besides wires, bus-based interconnect consists of logic components such as address decoders, arbiters and bridges [3, p.18-19]. Address decoder parses transaction address initiated by a master and selects a corresponding slave associated with this address. An arbiter is responsible to determine what master will be granted access to a bus if multiple masters request access at the same moment. An arbiter may implement some sort of priority mechanism, for example it can grant access in a round-robin manner or based on a time division scheme. Generally, decoding and arbitration can be implemented using centralized or distributed approach. A bridge is a logic component that connects separate buses. It can have fairly simple implementation if connected buses operate at the same frequency and follow the same protocol. Bus bridge includes protocol converters if connected buses use different transfer protocols and clock domain crossing (CDC) components if connected buses use different clocks.

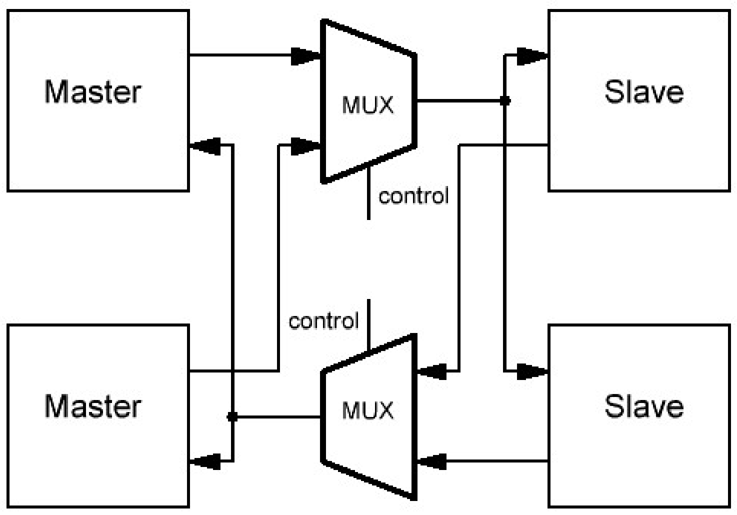
Earlier bus implementations used tri-state buffers that drive bidirectional lines Figure 2.1(a). The advantage of tri-state bidirectional buses is that they take up fewer wires and have a smaller area footprint. However, due to higher power consumption, higher delay (which can limit performance), and problems with debugging tri-state buffers, their use is restricted in modern bus-based on-chip communication architectures. Modern on-chip buses are implemented with more efficient solutions[3, p.21-22], such as MUX based busses Figure 2.1(b) and AND-OR buses Figure 2.1(c). In MUX and AND-OR implementations, a bus arbiter drives control and select signals in such a way that only one master can access a bus at the current clock cycle and only one slave is able to send response signals.

Bus-based interconnects can have several types of topology arrangements. The simplest scheme is the single bus topology shown in Figure 2.2. Single bus topology is one of the most familiar and widely used bus designs. It features low area utilization and is easy to debug [4]. Shared buses have only one communication link between all chip components. Such property inevitably limits bandwidth per component as more components are added to the system. Another property that negatively affects the scalability of traditional buses is long wiring [4][5][6]. Long global wires cause high signal delay and degrade the clock frequency, require repeaters in integrated circuits (ICs) and many switches on field-programmable gate array (FPGA) fabric. Furthermore, if bus protocol requires a large set of parallel wires the routing becomes problematic due to the growth of wiring complexity. A large count of parallel wires may limit clock frequency because combinatorial logic may require the use of fan-out buffers to drive control signals for a larger set of MUX or AND-OR arrays. There are estimations that traditional bus topology can be used in a system with up to 5 CPUs and no more than 10 bus masters [6][4].

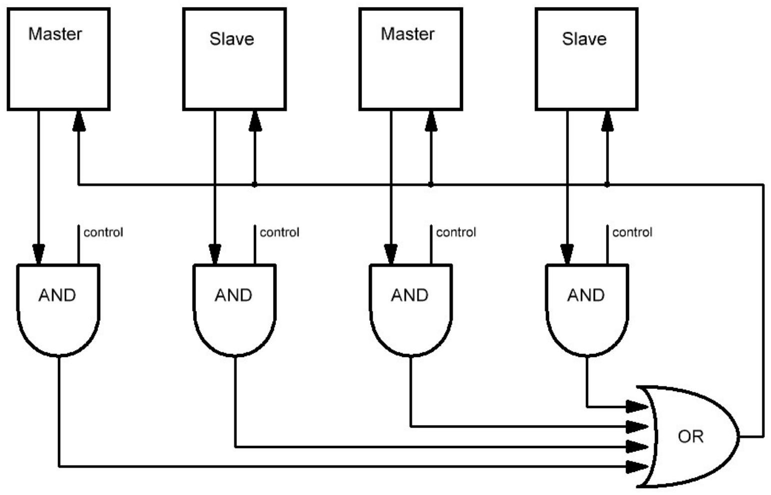
Hierarchical buses such as HIBI bus [7] split on-chip interconnection into several segments connected by bus bridges Figure 2.2. This solution increases bandwidth because masters and slaves can exchange data concurrently within each segment and allows the use of shorter communication wires overcoming some of traditional bus drawbacks[4][7].



(a) Bus based on tri-state buffers.

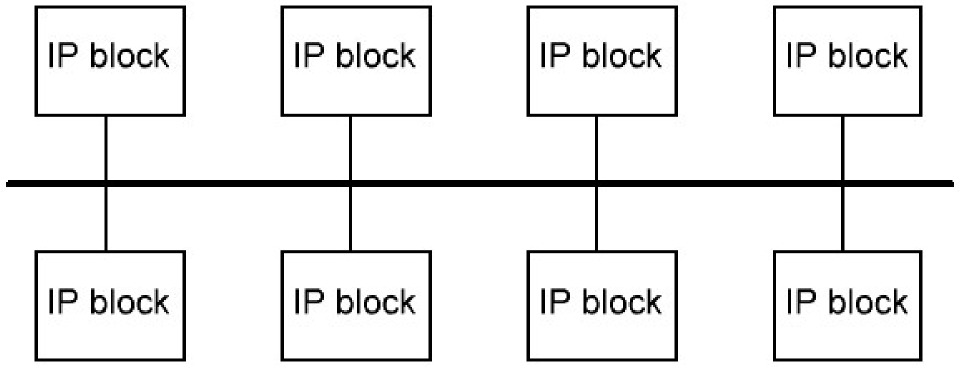


(b) Bus based on MUX components

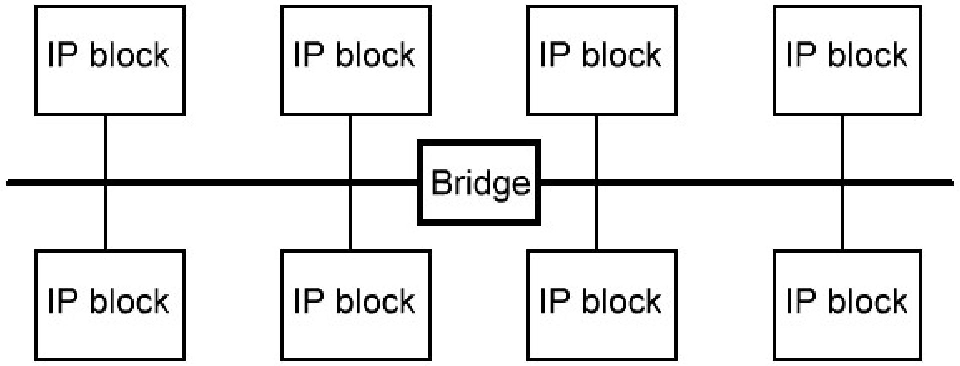


(c) Bus based on AND-OR components

Figure 2.1. *Shared bus implementations.*



(a) Single bus topology



(b) Hierarchical bus topology

Figure 2.2. *Bus interconnect topologies.*

For instance, Ryu and Mooney have presented an automated approach for generating application-specific hierarchical buses and reported 1.16 - 2.4 times of bandwidth increase and up to 37% of gate count reduction [8].

2.2 Crossbar based interconnects

One of the major drawbacks of shared bus topologies is low scalability. As the number of masters grows the potentially available bandwidth decreases. To overcome this limitation, crossbar or bus matrix communication topologies became widely used and adopted. Crossbar interconnects were studied in academia [9][10][11], used in commercial IP's [12][13] and open-source interconnect designs [14].

In the full crossbar topology, every master is connected to every slave via a dedicated bus Figure 2.3. A large number of separated buses allows many transactions to be processed in parallel. Several studies showed that crossbar topology provides significantly higher throughput than a shared bus and hierarchical bus alternatives [15][16]. Typically, crossbar interconnection is employed in high-performance computing systems that re-

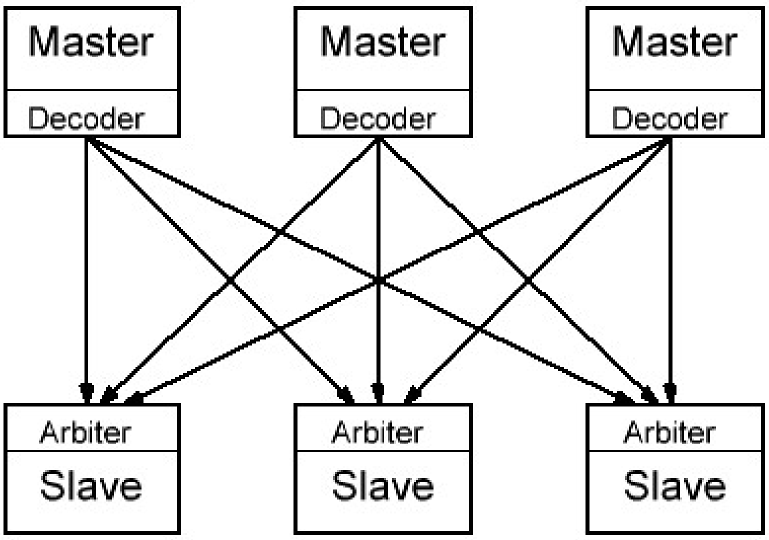
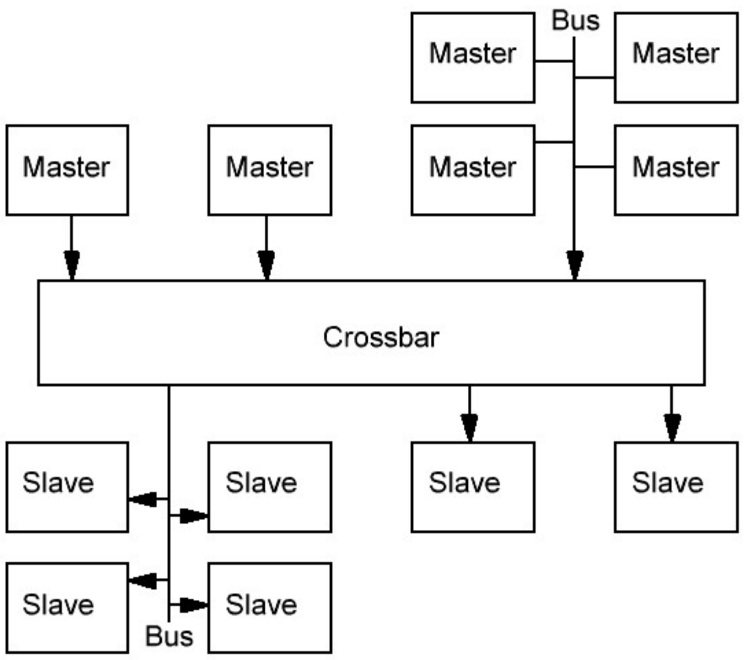


Figure 2.3. *Full crossbar.*

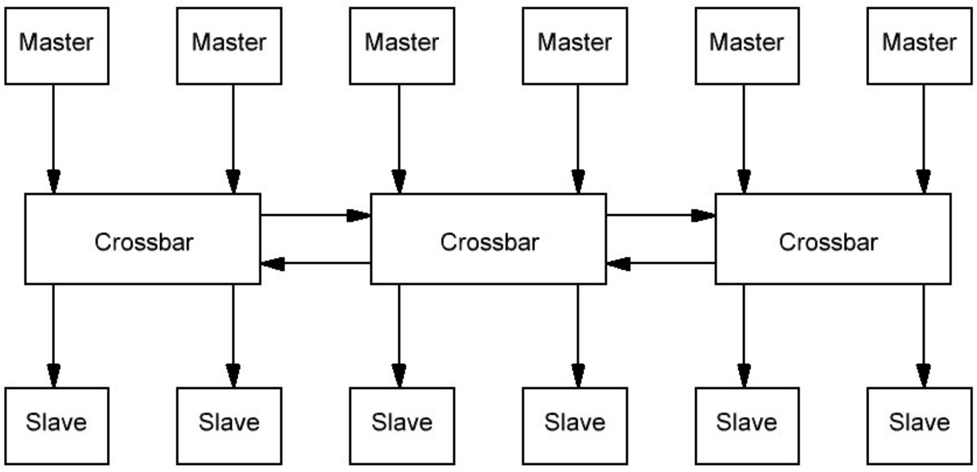
quire the highest possible throughput and data parallelism. However, due to advances in chip manufacturing technologies and transistor shrinking, crossbars became adopted in microcontrollers and edge computing systems as well [17][18].

While the full crossbar bus topology offers great parallelism, the excessive number of buses can take up a large chip area, increase power consumption, and make it practically impossible to achieve routing closure [19]. To overcome some of the most common crossbar limitations, several hybrid solutions can be used. One possible solution is a partial crossbar when a group of shared buses is connected with one or more crossbar blocks Figure 2.4(a). Another solution is to use cascaded crossbar approach, by hierarchically connecting several crossbars Figure 2.4(b).

Murali et al. compared shared bus, full crossbar and partial crossbar topology in [20]. The authors explored some older versions of STbus (the standardized and configurable interconnect design solution) wrapped around the cycle-accurate SystemC simulation environment to obtain performance metrics. The average reported latency is around 35 clock cycles for shared buses, 6 clock cycles for full crossbars and around 10 clock cycles for partial crossbars. Full crossbars on average utilized 10,5 times more area than shared buses, while partial crossbars used 4 times more area than shared buses. It is also worth saying that latency numbers can be used to evaluate potential throughput since all benchmarks employed a large number of CPU models executing in parallel. Of course reported numbers are very specific to the used workloads, test environment and author’s topology design method, but they clearly show that full crossbars provided much higher throughput at the expense of a larger area, while partial crossbar is a trade-off solution that saves around 60% of chip area at the expense average latency increase from 6 to 10 clock



(a) Hybrid crossbar topology



(b) Cascaded crossbar topology

Figure 2.4. *Crossbar based architectures.*

cycles.

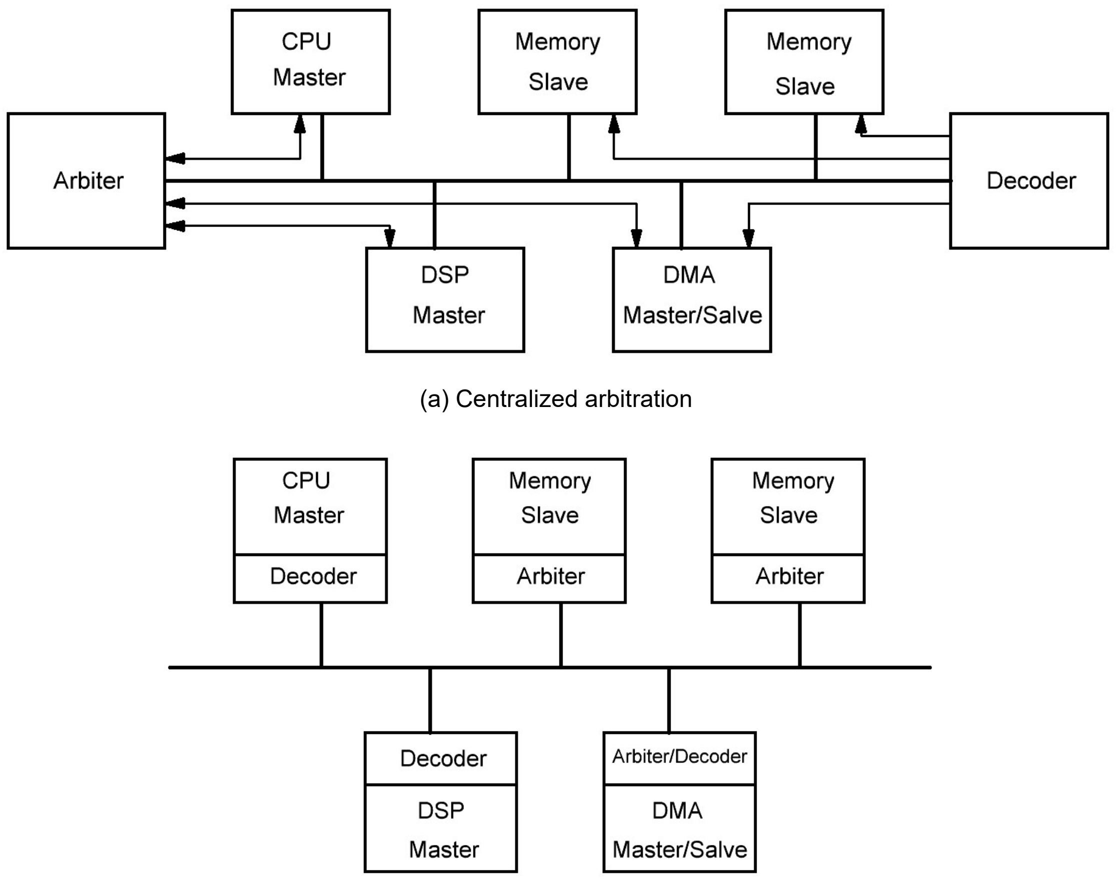
Pasricha et al. presented an automated approach to synthesise a partial crossbar interconnect in [9]. This solution takes into consideration the communication graph and minimal required throughput for each master and automatically generates interconnect topology targeting to minimize the number of generated buses. As a simulation environment, authors use the SystemC library of behavioural models and compare synthesized partial crossbar solution with a full crossbar design. This work reports up to 9 times component and area savings compared to the full crossbar, with the manual tuning of throughput constraints.

Jun et al. in the [10] presented the method to generate optimal cascaded crossbar topology. To increase maximum clock frequency, authors proposed to substitute a fully connected crossbar with several smaller crossbars interconnected hierarchically. This synthesis method takes the communication graph, required bandwidth, latency, clock frequency and area as inputs and tries to produce a topology that will satisfy all given constraints. With some results, this work reports up to 37.3% of clock frequency improvement at the expense of 43.6% area increase, and even up to 12% of area reduction with 21% of frequency increase compared to the single full crossbar. Later the same group of authors proposed some enhancements to their synthesis method in [21] and achieved more area reduction by up to 69% and 42.7% on average.

Yoo et al. in the [11] presented another study targeting to implement automatic synthesis flow to generate on-chip interconnect based on “cascaded bus matrix” topology. The authors do not compare cascaded crossbars with other topology alternatives, but rather compare different synthesis methods. Remarkably, this work compared up to 120 different bus architectures with 286 configurations and successfully synthesized cascaded crossbar interconnects for considerably large chip models with up to 31 masters and 71 slaves. The timing and area information was obtained by synthesizing the RTL code using Synopsys Design compiler using Samsung’s 90nm low-voltage ASIC process. Unfortunately, the authors did not share synthesis constraints such as targeting clock frequencies, area or desired bandwidths parameters.

2.3 Bus arbitration algorithms

The performance of bus-based interconnects depends on many factors such as architecture and topology, physical design constraints, number of masters and slaves etc. But one of the most crucial aspects that determine on-chip communication effectiveness is the arbitration algorithm. An arbitration process assigns priorities with which initiators are granted access to the shared communication resources. The increasing number of bus masters on single chips such as CPUs, DMA, DSPs and various accelerators can lead to severe contentions for communication resources, real-time constraints violations,



(a) Centralized arbitration

(b) Distributed arbitration

Figure 2.5. *Implementation schemes for arbiter and decoder.*

performance degradation and poor user experience for video and audio streaming applications. Therefore an efficient arbitration mechanism is required to support real-time and multimedia applications.

The address decoding is part of any bus arbitration scheme. Single bus arbitration can be implemented using a centralized or distributed approach Figure 2.5. In hierarchical bus architectures, an individual arbitration scheme is implemented for every single bus block, while bus bridges implement master or slave functionality. In crossbar designs, a dedicated address decoder is present for every master component. The crossbar arbiters are needed for every slave component to resolve contentions if more than one master tries to access the same slave.

A round-robin is one of the most widely used arbitration algorithms. In each cycle, one of the masters is granted the highest priority to access a shared resource. The highest priority is granted in round-robin order. If the master who was granted the highest priority does not access the bus, the master with the next highest priority can be granted a communication channel. The advantage of the round-robin scheme is that unused time slots are immediately reallocated to another master. Another positive aspect favourable for real-time applications is that the worst-case waiting time for bus access is reliably predictable for each master, which depends on the number of masters minus one.

A time division multiple access (TDMA) arbitration policy is based on the allocation of a fixed time slot for every bus master. The major advantage of the TDMA scheme is that every master is guaranteed a predictable bandwidth. As a drawback TDMA policy may cause significant latencies and inefficient bus utilization if master request patterns do not match the reserved time slots.

Static priority arbitration mechanism ensures that the highest priority master always gets access to the idle bus. If the highest priority master does not need a bus in the current clock cycle, access is granted to the next priority master. In this arbitration scheme, priority values are fixed during the entire chip operation. Note that priority arbitration cannot guarantee any bandwidth or bus access time for lower priorities masters. There is always a risk that lower priority masters will never get access to the bus if all time slots are utilized by higher priority initiators.

Dynamically adaptive arbitration algorithms assign master priorities and grant bandwidth slots by analysing bus access history. There are various adaptive approaches to distribute master priorities, for example in the [22] authors presented the arbiter that uses the look-up table (LUT) with master IDs that has dynamic and static part. The static part contains a simple list of all master IDs, while the dynamic part is implemented as a shift register. Whenever new master access the bus its ID is inserted into the dynamic LUT and the oldest entry is discarded. Every new clock cycle arbiter randomly chooses one slot from dynamic or static LUT. In such approach, the more some master uses the bus the more chances it has to get the next bandwidth slot, while the minimum bandwidth is guaranteed for every master by static LUT. Another example dynamic arbitration scheme was presented as part of Loterytbus arbitration scheme [23]. Lottery-based arbiter uses a list of “ticket” values given for every master. The arbitration decision is made by a pseudo-random generator that gives favour to masters with a larger ticket number. The ticket numbers can be distributed statically or dynamically by the lottery manager block.

There are several research attempts to analyse and compare different arbitration algorithms. Poletti et al. used the SystemC model of multiprocessor system-on-chip with AMBA bus to analyse and compare the performance of 3 different arbitration algorithms under different communication and workload patterns [24]. In this work, authors analyzed round-robin, TDMA and slot reservation arbitration policies. The slot reservation is a hybrid approach that uses TDMA to give every master some slot of contention-free access, while in inter slot time bus was arbitrated with a round-robin scheme. Firstly authors explored mutually dependent tasks workload so the different CPUs have to synchronize their execution at predefined points of time. For this type of workload, round-robin on average ensured lower execution time than others, while slot reservation was able to provide slightly better results than round-robin only with certain adjustments of reservation slot time length. For the second scenario when multiple CPUs were running independent workloads, round-robin approach again outperformed other algorithms. The last scenario explored pipelined workload when CPUs were almost free from memory accesses, so most of the bus transactions were related to communication between processors. The round-robin approach again outperformed TDMA, while the slot reservation scheme performed slightly better or equal to round-robin results.

Kulmala et al. presented another research to compare different arbitration mechanisms for FPGA based multiprocessor systems [22]. In this work, authors compared their novel dynamic arbitration approach with round-robin, priority-based and some hybrid arbitration algorithms. As a benchmark, complete MPEG-4 encoder was run on 4 and 11 processor systems. In this research authors measured HIBI [7] bus performance by directly counting bus traffic in words and by measuring the number of encoded video frames. Measuring tests were done with different clock frequencies and bus utilization rates. In their research round-robin algorithm outperformed all other arbitration schemes in most of the test cases. The dynamically adaptive arbitration scheme was able to outperform the round-robin algorithm only when the bus utilization rate was very low (the bus was not a limiting factor for system performance). Besides performance metrics, the authors provided valuable information on FPGA area utilization. Notably, the dynamically adaptive algorithm consumed 25 times more area than round-robin or priority-based algorithms.

3. NETWORKS-ON-CHIP

3.1 Network-on-chip basic concepts

Modern high-performance computing systems can contain dozens or even hundreds of CPU cores, multibank RAMs and various accelerators and IP blocks. In such complex systems with a large count of masters and slaves, on-chip communication will inevitably become a performance bottleneck if bus or crossbar based interconnect is used. As discussed in previous chapters such architectures suffer from scalability issues. Buses are limited in their throughput since all masters share the same communication link. Crossbars suffer from huge area overhead since the number of communication channels equals to $N_{master} \times N_{slave}$ and combinatorial delay in arbitration logic is proportional to the logarithmic function of number of masters $\log_2(N_{master})$. To deal with these limitations, industry and research community adopted a group of various interconnect designs named networks-on-chips.

Typical NoC uses a general-purpose interconnection network instead of a design-specific set of wires [25]. As shown in Figure 3.1, a NoC connects a set of network clients or processing elements(PE): CPUs, memories, peripheral controllers etc. Each processing element is placed in a specific location on a chip by following some regular pattern. The space between processing elements is dedicated to the network-on-chip wiring and there are no top-level connections between processing elements other than network wires.

NoC has many advantages over global wired buses such as structure, performance and resource utilization [25]. The regular structure of an on-chip network allows to highly optimize and control electrical properties of signal wires. Well-controlled electrical parameters such as low and predictable crosstalk can reduce power dissipation and propagation velocity in the communication wires [26]. Sharing communication links between processing elements makes wiring utilization more efficient. If some processing element is idle, others continue to use network resources.

Another unique networks-on-chip feature that distinguishes them from traditional bus-based architectures is the use of packets to route data over an interconnect. Every processing element is connected to the communication network via a standard interface designed to parse and send messages. Figure 3.2 shows a general NoC message struc-

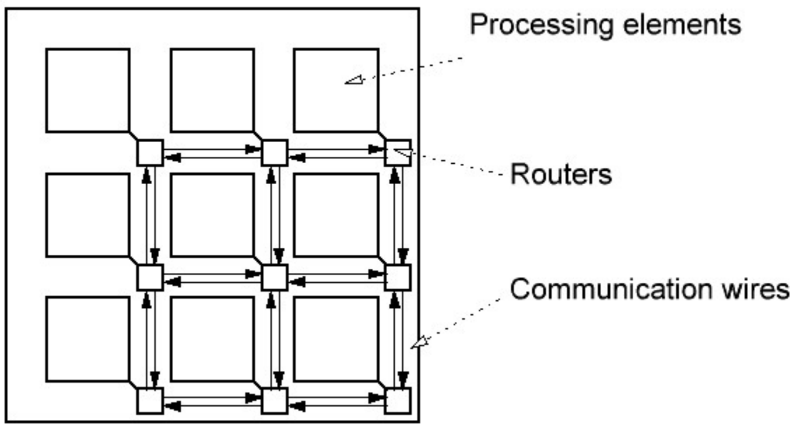


Figure 3.1. *Network-on-chip tiles and logic.*

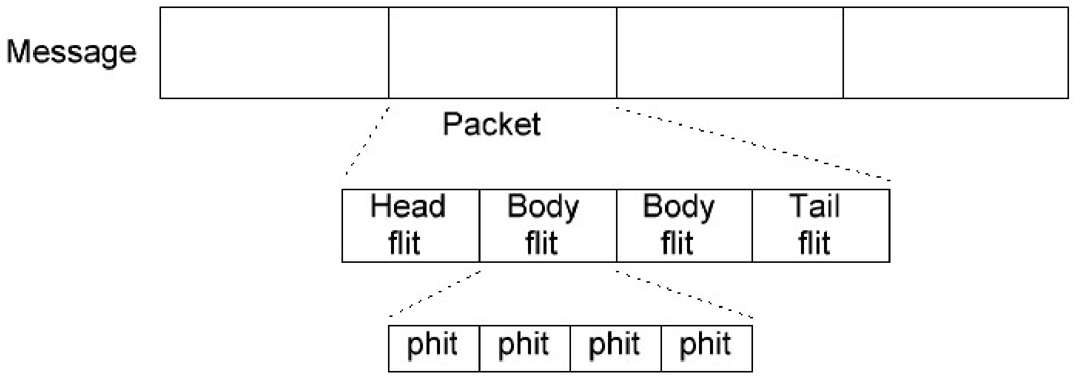


Figure 3.2. *Network-on-chip message structure.*

ture. Similarly to the large-scale networks, every message that is sent over NoC consists of one or several packets. Each packet is split into several flits (flow control units). Further, each flit is transferred as a set of phits (physical units). Typically phit has the same width as a communication link and is transferred in one clock cycle. As pointed in the [6] electrical noise due to crosstalk, electromagnetic interference, and radiation-induced charge injection will likely produce data errors when transmitting digital information over global SoC wires. Another unavoidable but rare source of data upsets is synchronization failures between different clock domains [27]. Packet-based data transmission allows employing of various error detection and error recovery techniques that have been developed for traditional large scale networks.

Network-on-chip stack can be mapped onto a standard ISO/OSI protocol layering model [3]. The physical layer is the lowest level of the protocol stack and represents the physical wiring of NoC. The data link layer consists of two sublayers: media access control layer (MAC) and data link control layer (DLC). MAC is responsible for physical medium access regulation and uses some sort of arbitration mechanism. The DLC sits on top of MAC layer and is responsible for error-tolerant communication by using error detection and error correction mechanisms. The network layer switches and routes packets through NoC. The switching mechanism determines the type of connection, while routing determines a message path between NoC nodes. The transport layer mainly decomposes network messages into packets at the source node and assembles messages from packets at the destination node. Session, presentation and application layers usually are out of NoC scope and implemented behind network interface by digital logic and software.

Network clients can layer higher-level protocols on top of NoC messages. Local logic of network interfaces can abstract various memory read and write protocols, data stream protocols or logical wires to the client. Similarly, local logic can provide a signal transformation from the source client to the router [25].

Despite the fact that NoCs in some characteristics and requirements are very similar to large scale networks, there are some unique characteristics of SoC networks [28] such as energy constraints and design-time specialization. Power spent on data storage and computation in modern chips was largely reduced due to transistors shrinking. However, the energy required for communication does not scale and only increases due to the rise of on-chip traffic. Power consumption is one of the major concerns in embedded systems. The need for more efficient traffic control and network resources utilization creates a set of new challenges that were not addressed by large scale network designs.

Design-time specialization is another unique aspect of NoC. Macroscopic networks emphasize general-purpose communication and modularity, but on-chip networks are free from such constraints since each network is designed on silicon from scratch. Standardization is needed to only define the network interface for the processing elements. However, network architecture can be tailored for a specific chip or set of applications.

Besides previously discussed challenges, on-chip networks can utilize some advantages that is not available for macro-network designs. As pointed in [25] micro-networks have enormous wiring resources. In this work, authors achieved 24000 pin connections for each router module. In contrast, pin count is one of the major limitations for inter-chip networks. On-chip network designers are allowed to easily trade network performance for pin count and explore new network topologies and architectures.

As discussed in [3, p.442-443], adoption of networks-on-chips is an evolutionary process, that was determined by several factors such as transistors shrinking and growing complexity of embedded systems in past decades Figure 3.3. Some advanced bus-based architectures such as cascaded crossbar solutions and some hybrid topologies are not far from network-on-chip designs that use crossbar in each router. While NoC solutions are much larger than bus-based architectures and employ many inventions from the communication domain, they still have to deal with the same design constraints, requiring trade-offs between performance, area, power usage, costs and reliability.

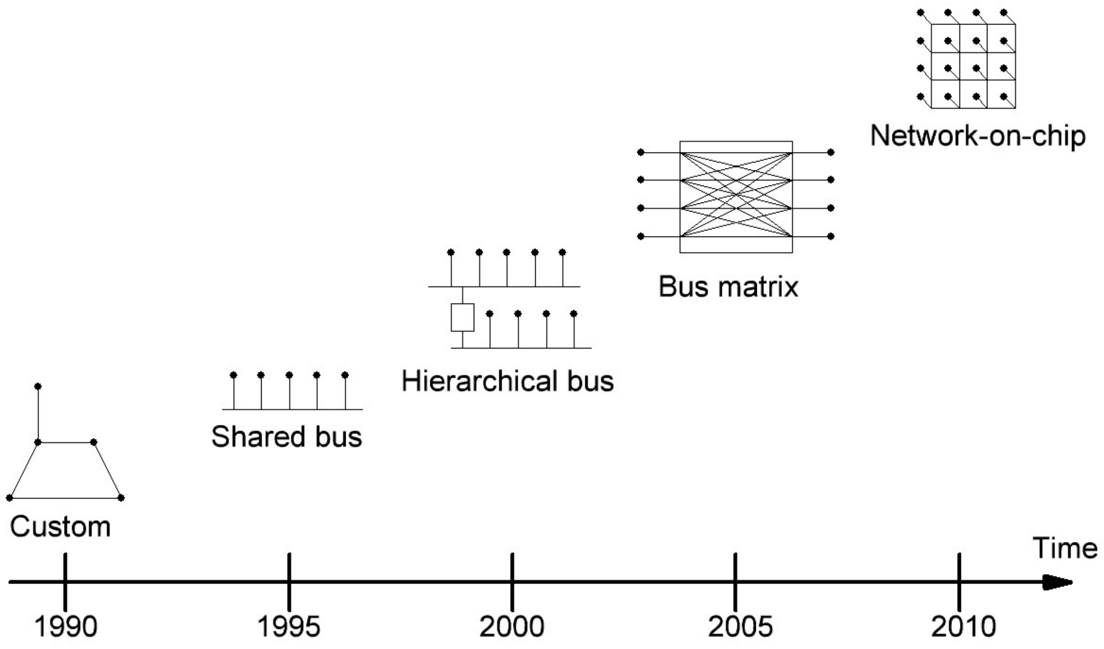


Figure 3.3. *Evolution of on-chip communication architectures [3, p.443].*

3.2 Network-on-chip router

Similarly to large scale networks, the basic building block of network-on-chip is a router module Figure 3.4. The role of a router is to accept packets from processing elements or other routers and direct them to the destination network clients. Typical routers employ virtual-channel flow control [25][29]. Every router input has a dedicated buffer and input controller with control logic for each virtual channel. When head flit arrives, the control logic uses route data from this flit to select an output port and then arbitrates incoming flit with other virtual channels. If flit wins arbitration, it is directed to the output port via crossbar switch. The output port can provide buffering as well. Packets typically traverse through many routers when moving from source to the destination client.

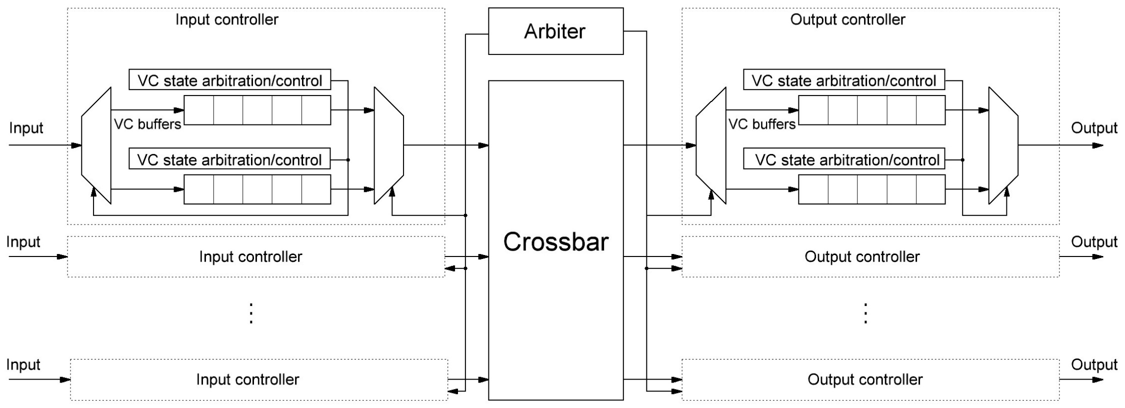


Figure 3.4. *Network-on-chip router.*

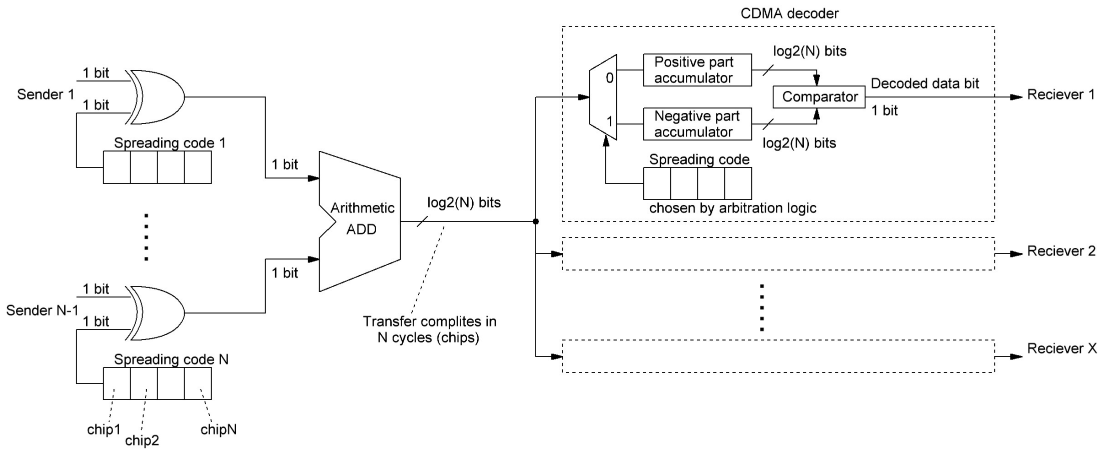
As stated in the [30, p.93-94], virtual channel support allows multiple packets to be sent via the same communication link simultaneously. Virtual channels allow the interleaving of fits that belong to the different packets. Note that virtual channels do not increase link bandwidth but enable more efficient link sharing between different network clients. Besides better resource sharing, virtual channels allow deadlock avoidance, packets isolation and quality of service guarantees.

The central part of a typical NoC router is a crossbar block that establishes connections between all network interfaces. There are several crossbar design techniques used in the NoC domain. One of the most widely used is space-division multiple access (SDMA) approach which is very identical to the bus-based crossbars, where a dedicated physical link is established between every pair of network interfaces.

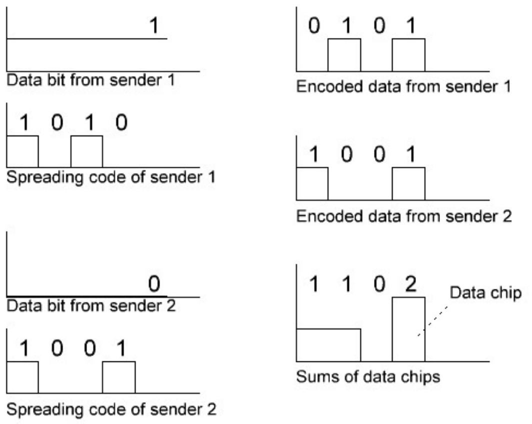
Code-division multiple access (CDMA) is an alternative for SDMA. CDMA technique is adopted from wireless communication domain to reduce arbitration overhead and physical resource usage [31][32]. The use of CDMA allows multiple transmitters to send information over a single communication line simultaneously and relies on the code orthogonality principle [33]. Each transmitter is assigned a unique spreading code. Before transmitting, each sender modulates each bit of a message by XORing it with N-bit spreading code, which leads to splitting one single cycle transaction into N transfers (chips). After XOR operation N bits that belong to the same chip and came from different transmitters are simply added together and sent over a CDMA line Figure 3.5. Since encoded bits from different transmitters are arithmetically added together the number of communication wires is $M \times log_2(N)$, where N is the number of network transmitters and M is a size of a message or phit. Because of the orthogonal property of spreading codes, data can be recovered at receiving end without data loss by multiplying incoming numbers with the same spreading codes used in the encoding stage [33].

Wang et al proposed a simplified decoding scheme for CDMA channel [33] Figure 3.5. The received chips are summated into two separate parts. If the corresponding spreading code bit is 0, the sum is added to the positive accumulator. If spreading code bit is 1, a chip goes to the negative accumulator. The original data bit is 1 when the value in the positive accumulator is larger than the value in the negative accumulator. Otherwise, the data bit is 0. If the original bit is 1 after the XOR operation, it can only contribute non-zero values to the sum if the spreading code bit is 0. Similarly, if the data bit is 0 it can contribute a non-zero value to the sum if the spreading code bit is 1. Such decoding scheme relies on the balanced property of spreading code sequence (number of 1 equals the number of 0). Several types of spreading codes were proposed for CDMA communication: Walsh code, Gold sequence, M-sequence, Kasami sequence. However, only Walsh codes have both orthogonal and balanced properties.

Time-division multiple access (TDMA) is another resource sharing approach [34]. When



(a) CDMA encoder and decoder schematic



(b) CDMA encoding example

Figure 3.5. *CDMA encoding.*

TDMA is in use, one single physical link is used to connect all router ports. Each transmitter reserves a dedicated time slot according to some scheduling algorithm. TDMA allows to spare logic and wiring resources because all transactions go through one fixed-sized physical link. On the other hand, TDMA limits bandwidth because all transmitters share the same link and lead to bandwidth waste if some transmitter is idle during its time slot.

The SDMA approach theoretically provides the highest possible bandwidth because each TX-RX pair has a dedicated physical link. But it is the most expensive in terms of wiring and logic. The wiring resource usage is proportional to$N \times M$, where N is the number of transmitters and M is the number of receivers. Moreover, a separate arbiter is needed for each receiver port. The CDMA approach provides a trade-off solution between TDMA and SDMA. Wiring resources used by CDMA are proportional to the $log_2(N)$, where N is the number of transmitters and only one arbiter module is needed to assign spreading codes for decoding. On the other hand, CDMA limits bandwidth because each bit is transmitted in S number of chips. The S number depends on the number of network transmitters and spreading code properties.

3.3 Network-on-chip topologies

The topology of network-on-chip refers to the physical organization of the network, number of nodes, routers and communication links and interconnection between them. The NoC topology has a large impact on communication and computation speed, power consumption and feasibility to meet physical design constraints. NoC topology choice requires evaluation of many topology alternatives to find a design that meets all communication requirements and imposes the minimum cost [35].

NoC topologies can be classified as regular and irregular or custom designs. Regular topologies feature homogeneous distribution of routers. Regular topologies are highly reusable and require minimal design time and cost but can lead to poor resource utilization. The majority of computing systems are not homogenous, different IP blocks and memories have variable areas and are not very easy to be placed on-chip regularly. The physical space dedicated for interconnect design very often are not regular so custom network topology has to be used. Routers may have a variable number of processing elements connections and links to other routers when the irregular topology is implemented.

Up to now, the research community was mainly focused on exploring basic regular topologies such as mesh, torus, star etc. However, industrial solutions put more effort to develop custom topologies using regular NoC topologies as building blocks to compile irregular networks Figure 3.6.

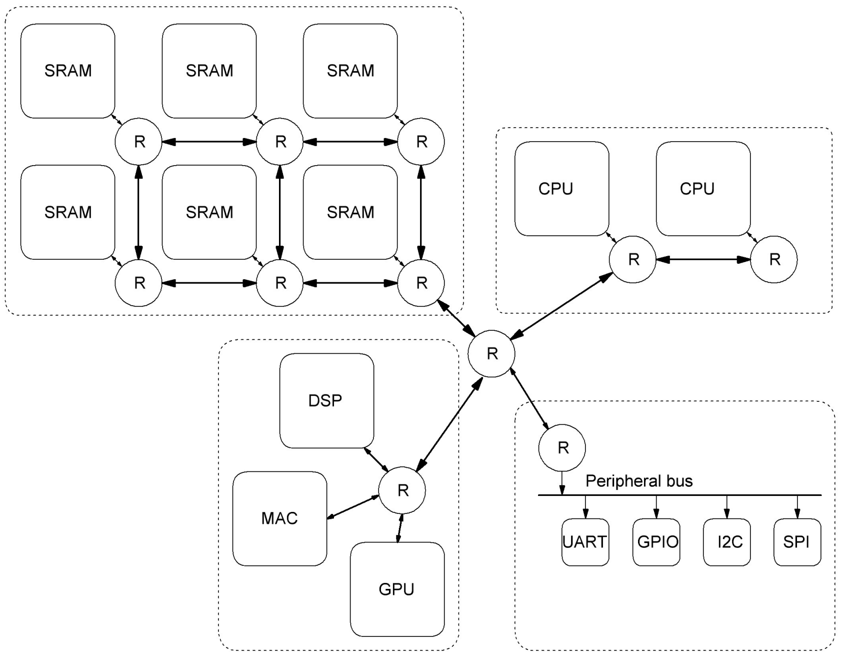


Figure 3.6. *Network-on-chip irregular topology.*

The 2D mesh topology is the simplest and most popular topology Figure 3.7. It consists of $N \times M$ routers. Every router, except those at the edges, connects to four neighbours routers and one processing element. All links are assumed to be equal in length and placed in a regular pattern. This simplifies the physical design process and allows to easily predict the required area for mesh topology because it is almost linearly proportional to the number of processing elements(nodes). The mesh topology has some drawbacks as well. The high number of routers increases area usage and usually leads to traffic congestions, especially in the central nodes [35]. An example of mesh topology is presented by Kumar et al in [36].

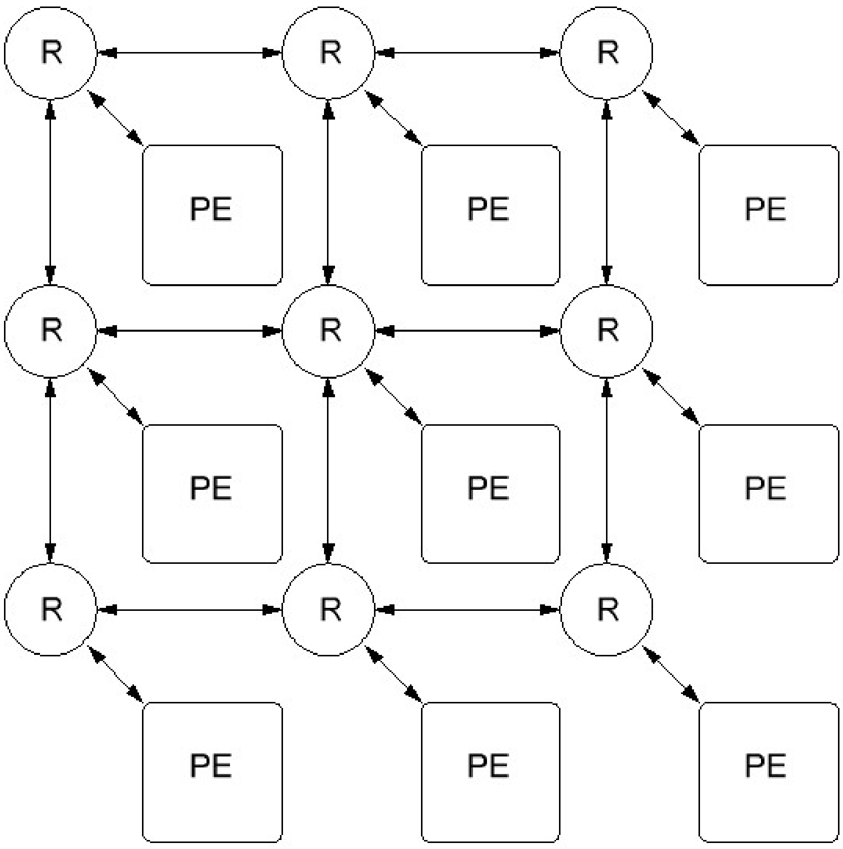
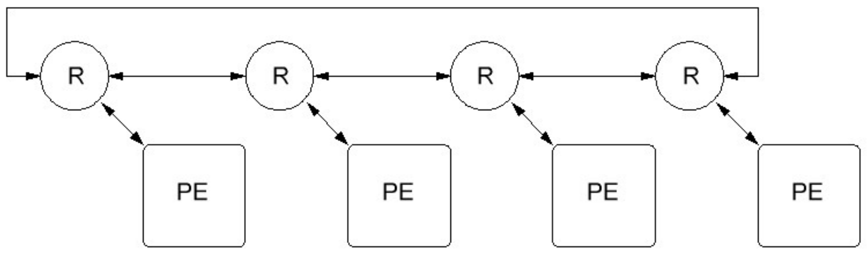
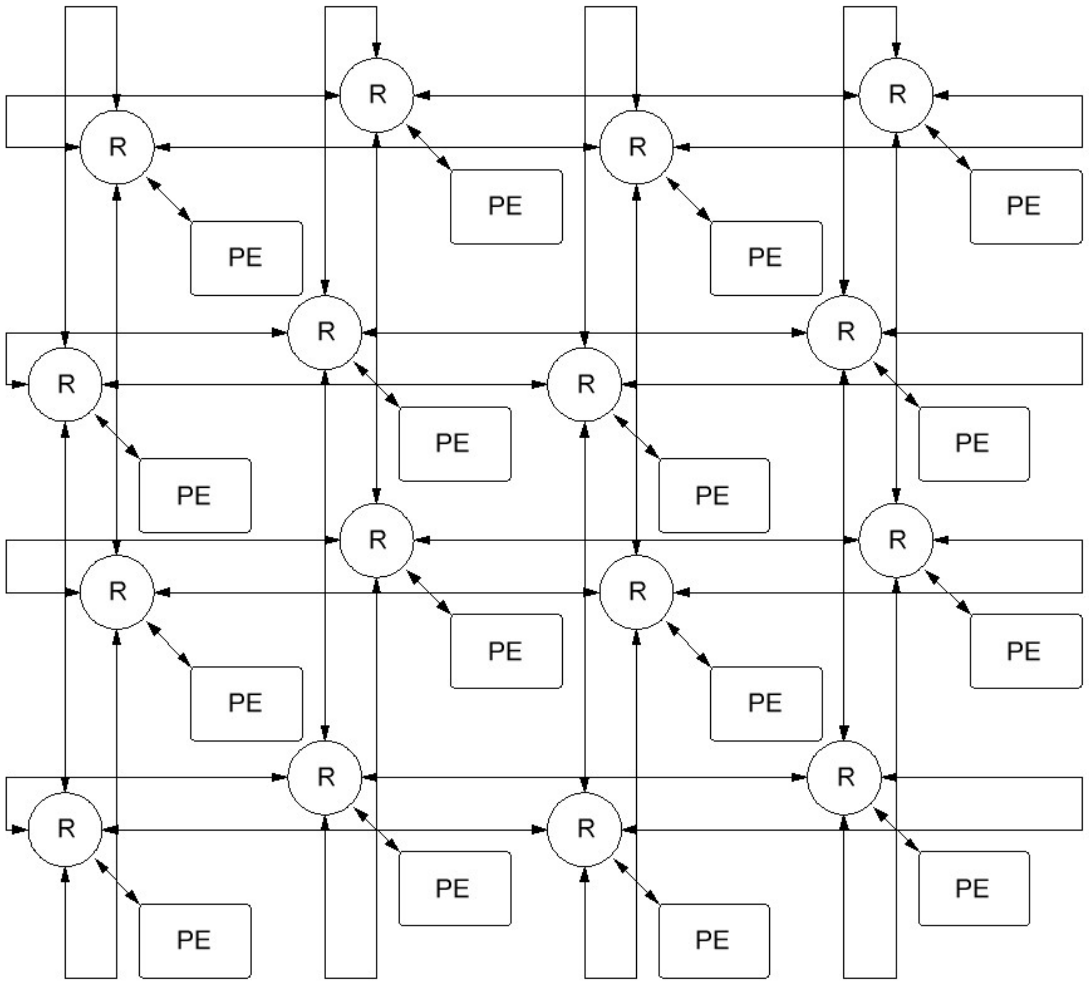


Figure 3.7. *Network-on-chip mesh topology.*

Torus is another well know NoC topology, which is formed by an n-dimensional grid with k-nodes in each dimension. One dimensional torus is known as ring topology and is shown in Figure 3.8(a). Ring topology is easy to implement but it suffers from scalability and performance issues as the number of nodes grows. The 2-dimensional torus is shown in Figure 3.8(b). The 2D-torus layout is similar to the mesh topology except that edge routers are connected to the opposite edge router via the wrap around routing wires in some cyclic order. These long edge to edge wires introduce long delays and produce most limitations related to the physical implementation of 2D-torus. The example of 2D-torus topology is described by Dally and Towels in [25].



(a) Network-on-chip one dimensional torus.



(b) Network-on-chip two dimensional torus.

Figure 3.8. *Network-on-chip torus topology.*

The polygon is another well-known approach to design NoC topology. In its simplest form, a polygon topology consists of n number of routers connected in a ring with several diagonal communication links between non-adjacent routers. The example of 8 edges polygon (octagon) topology is shown in Figure 3.9. Such topology allows using of simple

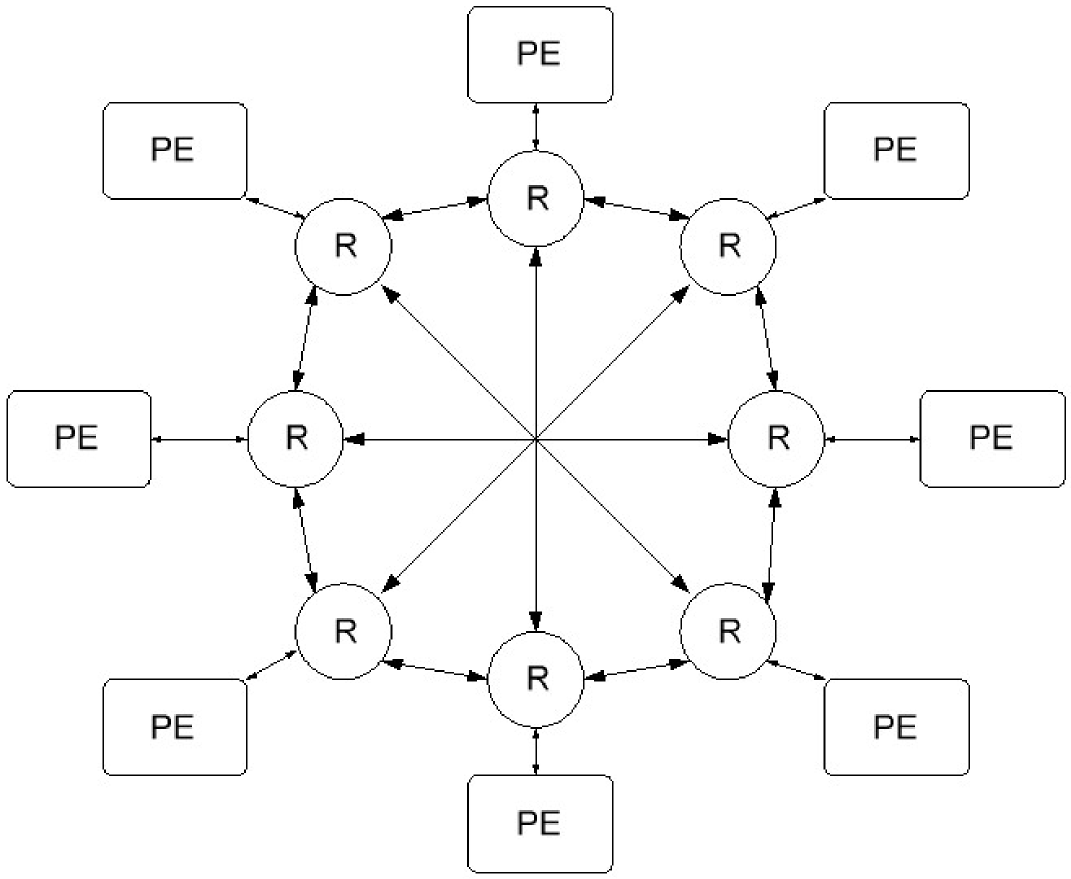
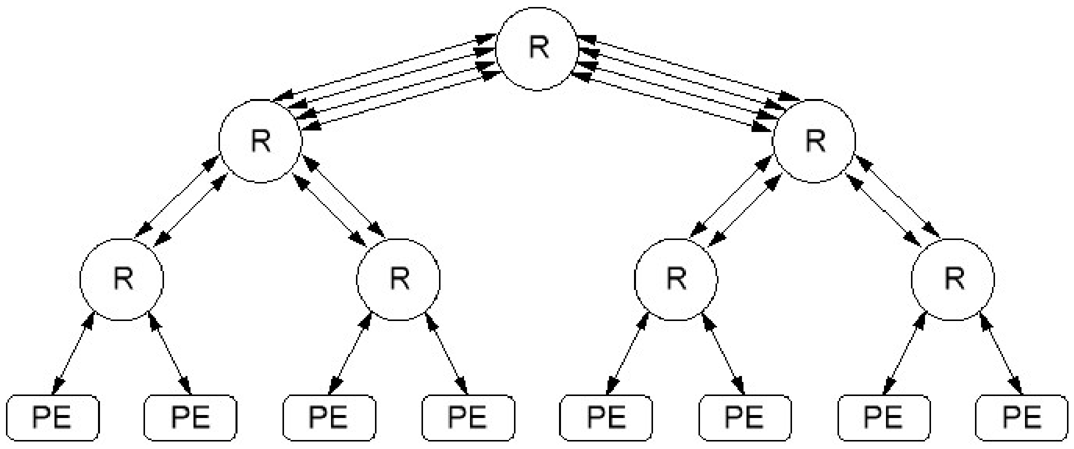


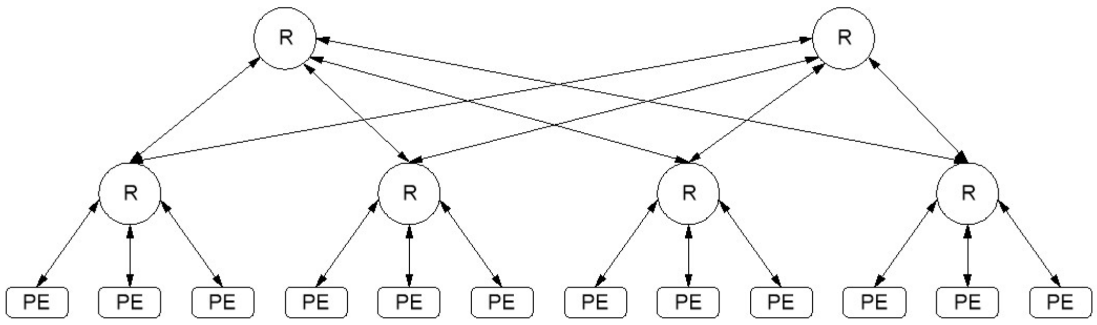
Figure 3.9. *Network-on-chip octagon topology.*

routing algorithms to find the shortest communication paths but increases the use of wiring resources. The example of octagon topology is explored by Reehal et al in [37].

Fat-tree topology can have many variations but in all of them, network nodes or processing elements are assigned to the leaves of the tree while routers take place of tree nodes. Figure 3.10(a) illustrates a fat-tree topology that follows a binary tree structure with a single root node. Another example in Figure 3.10(b) shows butterfly fat-tree topology with several root nodes. The unique feature of fat-trees is that links that are closer to the tree root have more throughput capacity because they have to handle more traffic from child routers and leaf nodes. The higher throughput near the root router is ensured by adding more physical links. Fat-tree family of NoC topologies was extensively studied by Karniemi in [38].



(a) Network-on-chip binary tree topology.



(b) Network-on-chip butterfly topology.

Figure 3.10. *Network-on-chip fat-tree topology.*

4. BALLAST CHIP ARCHITECTURE

This chapter will explore architecture of the Ballast chip, which was designed by Tampere University with industrial partners as part of the SoC Hub activity [2]. Ballast is a heterogeneous multiprocessor system-on-chip (MPSoC) for edge computing, machine vision and research application. The simplified high-level block diagram of the Ballast chip is shown in Figure 4.1. The presented chip has three RISC-V[39] CPU subsystems, DSP coprocessor, deep learning accelerator, ethernet communication core, chip-to-chip communication interface and rich sensor interfaces. The majority of utilized IP blocks are open-sourced solutions from the research and industrial community.

The sysctrl subsystem is implemented by adapting and customizing Pulpissimo microcontroller [17][18] developed by ETH Zurich and University of Bologna. The main role of sysctrl CPU is to run boot code software and provide control of all reset and clock sources in the system. The sysctrl subsystem is not intended to run computationally intense workloads. To reduce area footprint and power consumption, this subsystem contains RISC-V IBEX CPU core [40] instead of CV32E40P CPU [41], which is the default option for the original pulpissimo microcontroller. To further reduce area footprint, sysctrl has a smaller SRAM (2x32 kB) and ROM (1 kB) size and a limited set of peripheral interfaces.

Ballast chip is a globally asynchronous locally synchronous (GALS) electronic circuit, meaning that different subsystems are clocked with independent phase-locked loop (PLL) clock sources. The ability to control clocks of various subsystems is essential for research applications because it allows to accurately control power consumption and disable idling subsystems. For this purpose, sysctrl subsystem was augmented with a set of clock control registers, which is connected to its internal APB bus and visible to other CPUs via top-level interconnect. The set of clock control registers allows application software to control reset signals of other subsystems and toggle switching between external reference clock or internal PLL block. Every PLL block has a dedicated subset of sysctrl registers to control and monitor its operation and adjust output frequency.

The Ballast boot procedure is executed by sysctrl subsystem. The Boot ROM contains initial boot routines, which are executed by default when IBEX core starts its operation. The role of the boot ROM code is to perform I/O pin initialization and start loading subsystems SRAM with code from external data storage devices. Depending on execution flow and

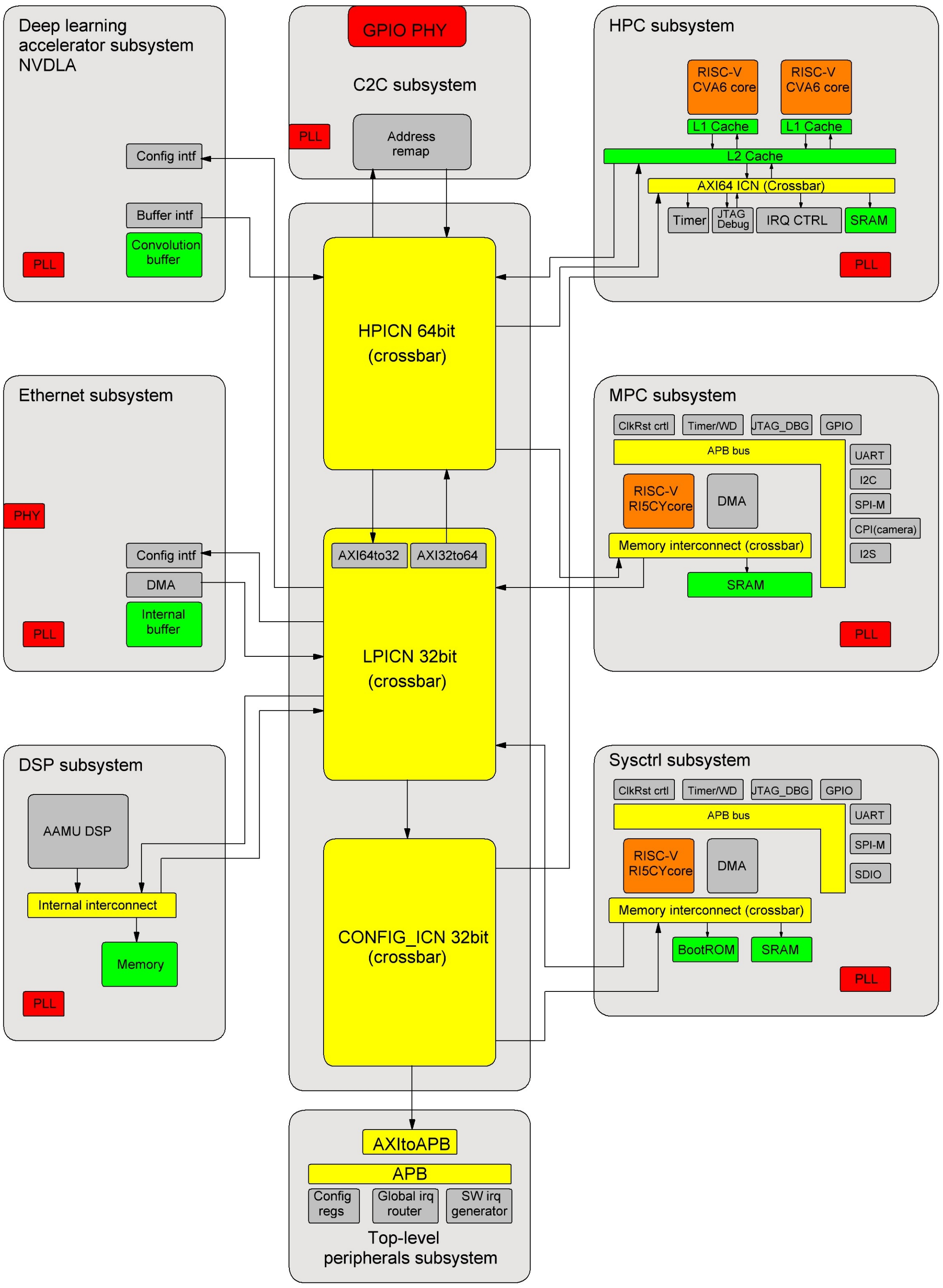


Figure 4.1. *Ballast block diagram.*

user-defined pin signals, boot ROM code can load from an Secure Digital (SD) card using direct access hardware, software-controlled Secure Digital Input Output (SDIO) interface or serial peripheral interface (SPI) mode. The external storage device may contain next stage boot code or SW applications. After code is loaded, systrl transfers execution to the SRAM. Alternatively, application or boot code can be loaded to the SRAM by using JTAG connected debug modules.

Medium performance computing (MPC) subsystem is intended for edge computing, robotics and various data acquisition applications. MPC subsystem implementation is based on the original Pulpissimo microcontroller. This subsystem features RISC-V CV32E40P 4 stage computing core with floating-point extension and 520 kB of SRAM divided into 6 memory banks (2x32kB and 4x114kB). To support various sensors and I/O devices, MPC subsystem has rich set of peripheral interfaces such as I2C, I2S, UART, SPI and CPI(camera parallel interface). To accelerate computer vision and machine learning applications MPC subsystem includes hardware processing engine (HWPE), which is closely connected to its internal interconnect and is capable of single instruction multiple data (SIMD) type of operations through RISC-V instruction extensions and supports up to 4 parallel MAC computations on 32bit data.

High-performance computing (HPC) subsystem is designed to run computationally demanding applications and is capable of running Linux and Unix-like operation systems. This subsystem includes 2 Ariane CPU cores [42][43], L2 cache logic, internal interrupt controller, timers and debug module. Ariane CPU is a 64-bit RISC-V processor. It supports all three privilege levels of RISC-V specification (user mode, supervisor mode and machine mode). To support virtual address translation, which is a must for the Linux support, Ariane core includes memory management unit (MMU), which consists of page table walker (PTW) and translation look-aside buffer (TLB). Each Ariane core has 16kB of L1 instruction cache and 32 kB of L1 data cache and is connected to the shared 256 kB L2 cache.

Sysctrl , MPC and HPC subsystems share one set of JTAG pins to connect with external debuggers. All CPU debuggers are connected in a daisy-chain configuration that is shown in Figure 4.2. Sysctrl and MPC subsystems have two independent debug modules: the RISC-V debug module[44] that is fully compliant with RISC-V external debug V 0.13 specification[39] and the legacy PULP debug module that is mainly used to speed-up simulations and to back up the main debug module. The HPC subsystem has one RISC-V debug module that is attached to both CPU cores and configured to operate with 64-bit architecture. The design for test (DFT) module is inserted by a synthesis tool to perform boundary-scan tests at the manufacturing stage and can be accessed via the same JTAG interface by configuring the select pin.

Chip-to-chip subsystem is designed to provide connections with external chips and FP-

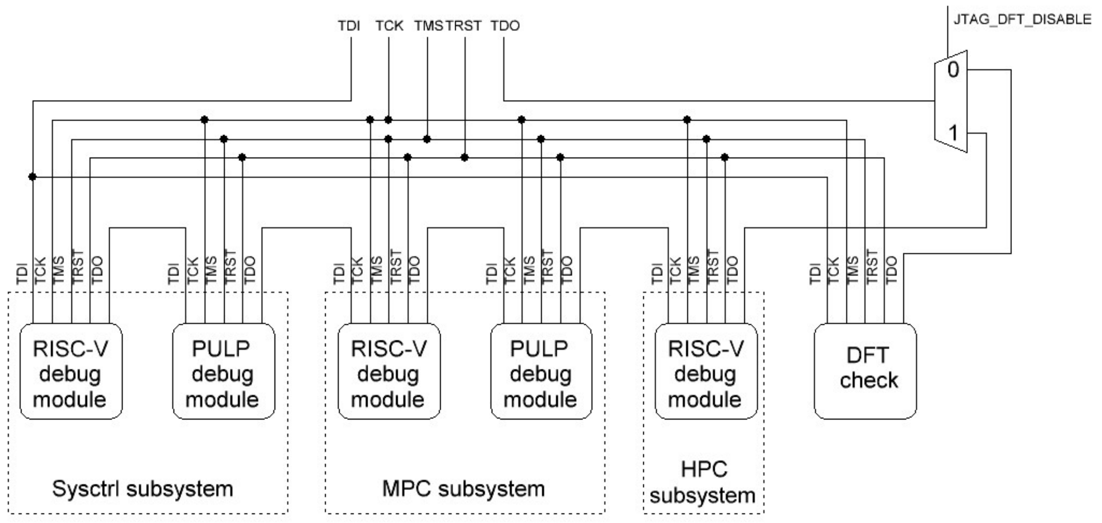


Figure 4.2. *Ballast debug modules.*

GAs. It allows to connect mass storage devices and DDR interfaces and makes it possible to build clusters with multiple Ballast chips. The physical link implementation is based on differential unidirectional signalling and Acknowledge/Valid flow control scheme. From the internal interconnect side C2C subsystem exposes 64-bit AXI-4 master and slave ports. According to the latest synthesis results, chip-to-chip can support up to 1833 Mbits/s bandwidth for write transactions and up to 592 Mbits/s for read transactions if maximum burst length is used.

AI accelerator subsystem design is based on NVIDIA Deep Learning Accelerator (NVDLA)[45]. NVIDLA is a free and open architecture that accelerates deep learning inference. The internal data path of the AI subsystem is designed to provide hardware acceleration for a single layer of convolutional neural network. The convolution pipeline consists of a convolutional buffer, convolution core, activation engine and polling engine. The intermediate results are stored in the internal memory block.

The ethernet subsystem is based on open-sourced ethernet MAC controller distributed by OpenCores community [46]. The MAC supports bandwidth up to 1000 Mbps and conforms to IEEE 802.3ab specification. Ethernet subsystem includes DMA engine to enable data streams between global memory and the MAC.

DSP subsystem is designed to act as a co-processor accelerator to support various signal processing tasks such as demosaicing, denoising, colour mapping, white balancing, gamut mapping, tone mapping and audio processing. The core was designed using the TTA-Based Co-design Environment (TCE) tools which provide an instruction set simulator, assembly and C compiler and VHDL generator [47]. Transport Triggered Architecture (TTA) is a design approach where the accelerator internal datapaths are exposed in the instruction set. TTA processor datapath consists of execution units, register files and buses that connect them. A DSP accelerator is programmed by controlling directly the buses, and operations are executed as a side-effect of these moves. The synthesized version of DSP subsystem runs at 600 Mhz clock speed and can sustain maximum of 2.5 operations per cycle.

Top-level peripheral subsystem consists of global interrupt router, software interrupt generator and set of control registers. The interrupt router receives interrupt signals from AI subsystem, ethernet MAC and software interrupt generator. Incoming interrupts are then routed to the intended receiver according to the configuration register values Figure 4.3. Software interrupt generator is designed as a set of registers in which every interrupt line has read, set and clear address offsets. Besides global interrupt control, top-level peripheral subsystem provides register interfaces to remap chip-to-chip address offsets, do physical memory and pad configurations.

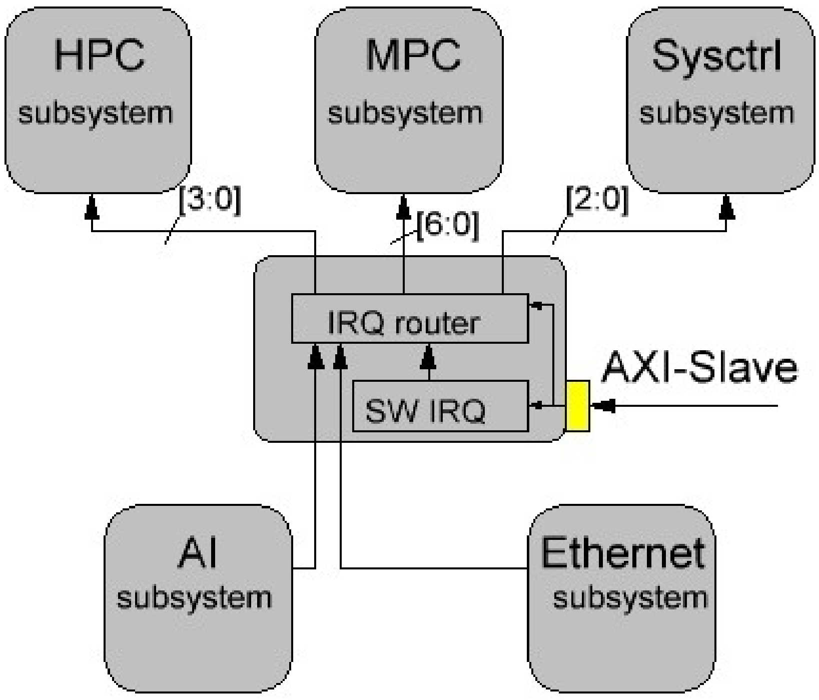


Figure 4.3. *Ballast interrupt router.*

5. BALLAST CHIP INTERCONNECT

5.1 Ballast interconnect specification and requirements.

A design specification for on-chip interconnect may include various requirements, constraints and design parameters such as interfaces or bus protocol, list of master and slave connections, desired bandwidth, maximum latency, physical layout, expected power consumption, topology, communication priorities etc. Depending on the design stage and adopted methodology, many of these factors may be or may not be included in the interconnect specification. A specification for absolutely novel chip architecture will include much fewer details than a specification for a next-generation chip, that is based on previous products. A chip that is intended for a very narrow scope of applications can be specified with more details than a chip designed for a wide set of applications.

Nowadays a large scale industrial chip design will likely start with functional and loosely-timed modelling. Modern interconnection IP technologies such as FlexNoC from Arteris[48] or Corelink Interconnect from ARM[49] are delivered with a design suite that will not only generate RTL but provide SystemC or another interconnect model that will allow system architects to adjust and test future interconnect parameters. However, due to the open-source nature of the future Ballast chip, all proprietary interconnect design technologies were discarded. Considering a small size of the Ballast chip, functional and architectural modelling of its interconnect was omitted as well, so all initial design explorations have been started with RTL models.

One of the major research goals of the SoC Hub project is to explore new chip design methodologies and speed-up design processes by applying an agile development approach. To enable continuous testing and integration, the interconnect design started at the same point of time with other subsystems straight after the preliminary architecture definition of the Ballast chip. Due to the novelty of the Ballast chip, little information was available to specify its on-chip communication architecture. The expected use cases of the future chip and analysis of data traffic were used as a primary specification for the interconnect design Figure 5.1.

The first set of applications considers machine learning workloads that are accelerated by AI subsystem. Generated interconnect traffic is illustrated with green colour:

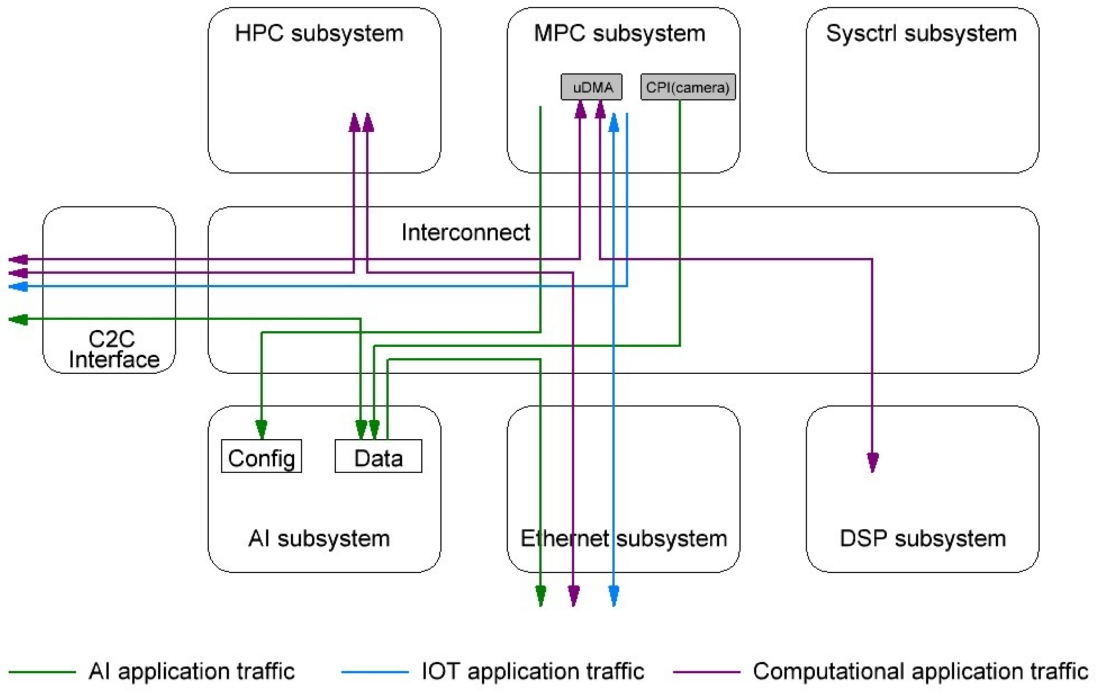


Figure 5.1. *Ballast use cases.*

- AI subsystem classifies images that come from MPC's camera interface. MPC subsystem controls the AI subsystem pipeline. Neural network weights and parameters are loaded from an external source via the chip-to-chip interface. Inference results are returned via ethernet or chip-to-chip interfaces.
- AI subsystem classifies data that comes from the chip-to-chip interface. MPC subsystem controls the NVIDLA execution and dataflow. Weights are loaded from the chip-to-chip interface.

Another set of use cases depicts IoT and data acquisition applications. Interconnect traffic is illustrated with blue colour:

- MPC subsystem reads sensor data from peripheral interfaces, performs data processing and output via the chip-to-chip interface.
- MPC subsystem reads sensor data from peripheral interfaces and receives network messages from the ethernet port. MPC performs data processing and control external electrical or mechanical systems.

The third set of applications deals with computationally intensive workloads. Interconnect traffic is illustrated with purple colour:

- MPC subsystem controls DMA engine to exchange data between chip-to-chip interface and DSP subsystem. Use DMA to stream output data via the chip-to-chip port.

- HPC subsystem fetches data from chip-to-chip interface and performs computations. Send results to the chip-to-chip interface or ethernet subsystem.

Listed use cases do not represent all possible applications of the Ballast chip, but help to identify points where more bandwidth was needed to be provided by future interconnect. Most of the data traffic is expected to go through chip-to-chip port, HPC subsystem, AI subsystem and ethernet. An additional requirement for Ballast interconnect was to provide connectivity between all bus masters and slaves, so all on-chip resources are visible to all CPUs and external requestors.

5.2 Ballast interconnect design and implementation.

The block diagram of the final version of the Ballast interconnect is shown in Figure 5.2. The Ballast interconnect consists of three hierarchical blocks: high performance interconnect, low performance interconnect and configurational interconnect. Each hierarchical block is implemented as an AXI4 protocol crossbar. The top-level topology can be classified as a chain of cascaded bus matrices. To illustrate bandwidths evaluations, the figure includes maximum clock frequencies for master, slaves and interconnect blocks.

AXI4 specification[50] was chosen as the main protocol for the Ballast interconnect implementation. AXI is supported by many open-sourced and proprietary IPs, the original Pulpissimo version has an external AXI slave and master ports. AXI protocol features separate read and write channels and allows the addition of slice registers to any communication channel to achieve timing closure. It supports up to 256 beat bursts, byte strobes and multiple other features for high-bandwidth and low-latency designs. If a particular subsystem uses another standard for its internal communication, the AXI protocol converter has to be added to the subsystem wrapper.

According to use cases analysis, the larger portion of interconnect traffic is concentrated around chip-to-chip port, HPC and AI subsystems. To support machine learning and computationally intensive applications, the high-performance interconnect (HPICN) provides a matrix interconnection between HPC L2 cache master and slave ports, chip-to-chip master and slave ports and AI subsystem master. To speed-up data exchange and minimize latency between ML accelerator and camera interface, high performance interconnect is connected with MPC slave port as well. Besides that, HPICN has master and slave ports toward low performance interconnect to provide communication between other subsystems. All connected masters, slaves and crossbar itself are configured to operate with 64 bit data width and 32 bit address. The low performance interconnect (LPICN) is designed to provide communication between subsystems that are not expected to produce a lot of interconnect traffic. MPC and Sysctrl masters, DSP master and slave, Ethernet master and slave and AI configuration slave are all interconnected by LPICN logic. All mentioned subsystems and crossbar are configured to operate with 32 bit data width and 32 bit

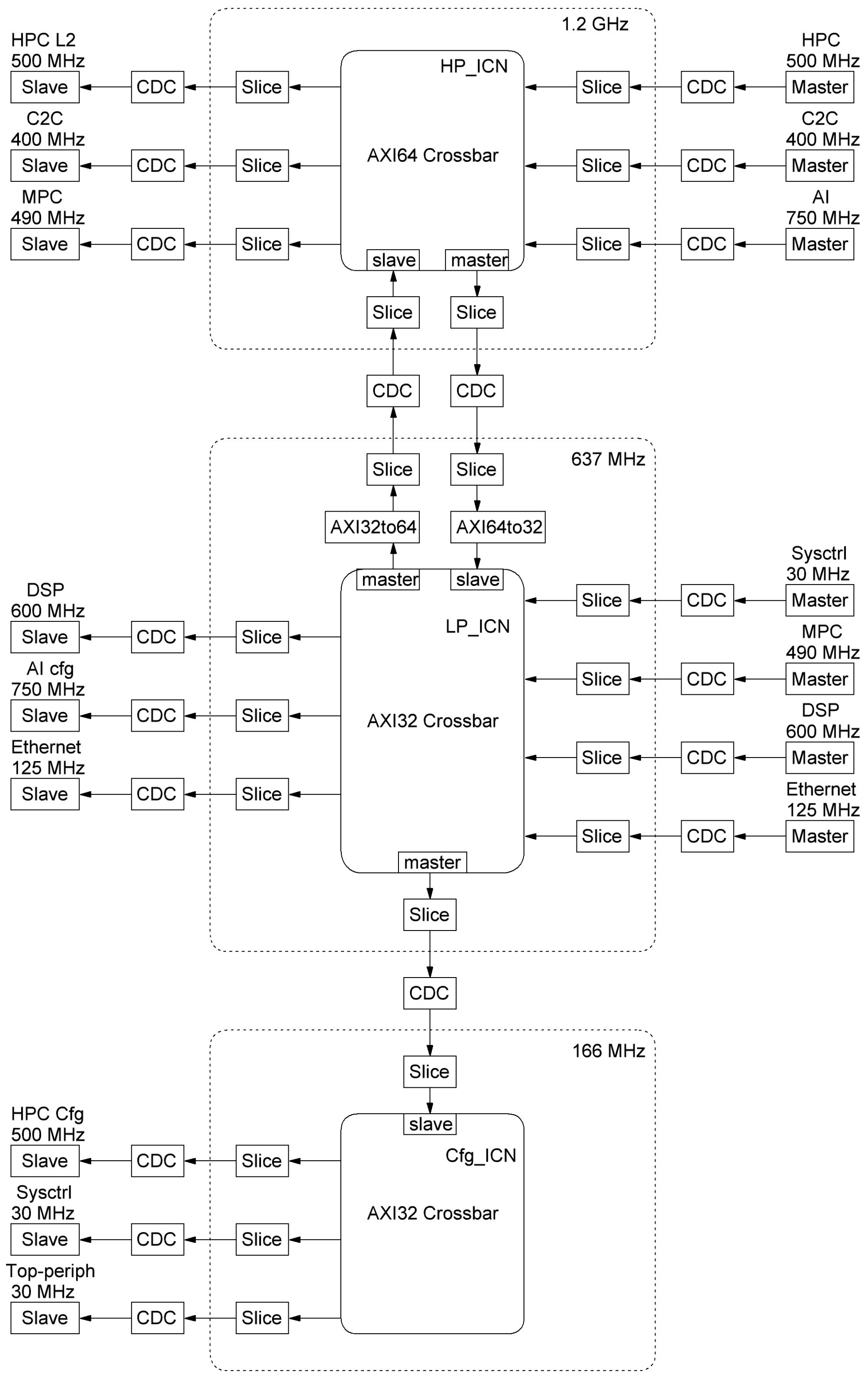


Figure 5.2. *Ballast interconnect.*

addresses. From HPICN side, low performance interconnect block includes protocol converters[51][52], which translate 64 bit data AXI transactions to 32 bit data transactions. Also, LPICN acts as the only master for the configurational interconnect block.

The configurational interconnect (ConfigICN) is another AXI crossbar that is designed to provide connections for those slaves which are supposed to be very rarely accessed by applications or during boot code execution. It connects with LPICN master port, Sysctrl slave, HPC config slave port and Top-level peripheral subsystem.

The crossbar component is a major building block of the Ballast chip interconnect shown in Figure 5.3. Its implementation is provided as open-source solution shared by Pulp Project[53][54]. The crossbar design is based on AXI protocol multiplexer[55], demultiplexer[56] and address decoder components for AR and AW channels. The crossbar component supports round-robin arbitration scheme. According to our previous analysis in the Chapter 2.3, a round-robin scheme is a preferable option for general purpose applications and we use it in the Ballast interconnect.

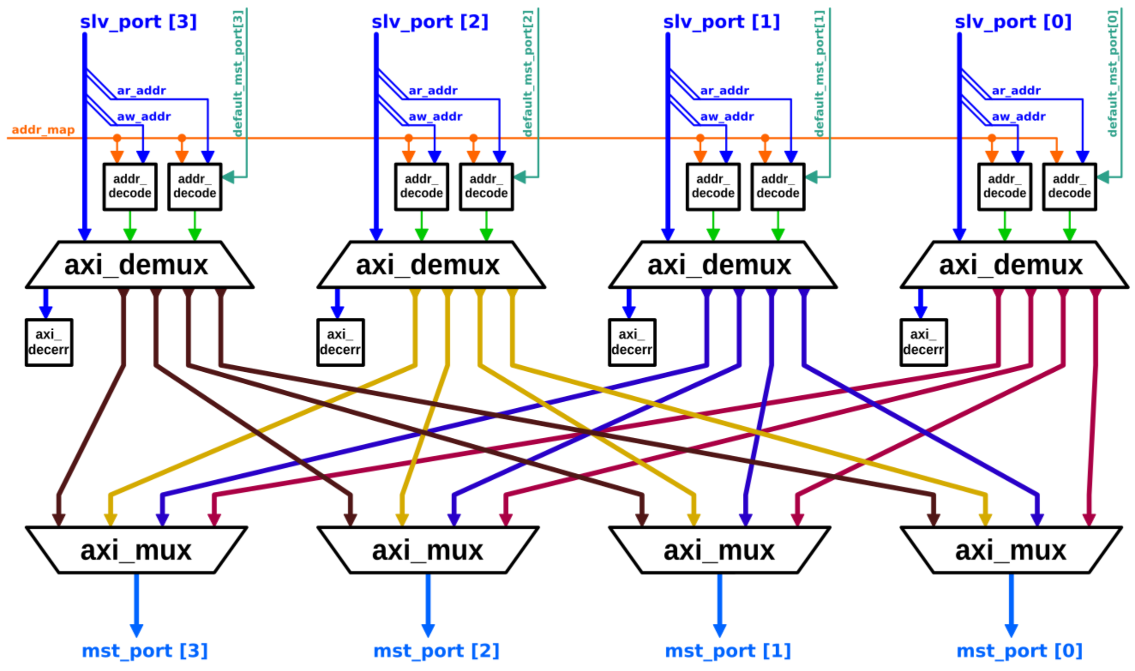


Figure 5.3. *AXI crossbar [53].*

The AXI demultiplexer component is shown in Figure 5.4. This module connects one AXI slave port to the configurable number of AXI master ports. AW and AR channels have a select inputs (slv_aw_select_i and slv_ar_select_i), in the crossbar design these signals are driven by address decoders to determine to which master port an ongoing AXI transaction is sent. Incoming AXI data and control signals are directly driven to all master ports, while valid signals (mst_aw_valids_o, mst_w_valids_o and mst_ar_valids_o) are controlled by the select component. The response beats coming via B and R channels from master ports to the slave port are multiplexed by round-robin arbiters (rr_arb_tree).

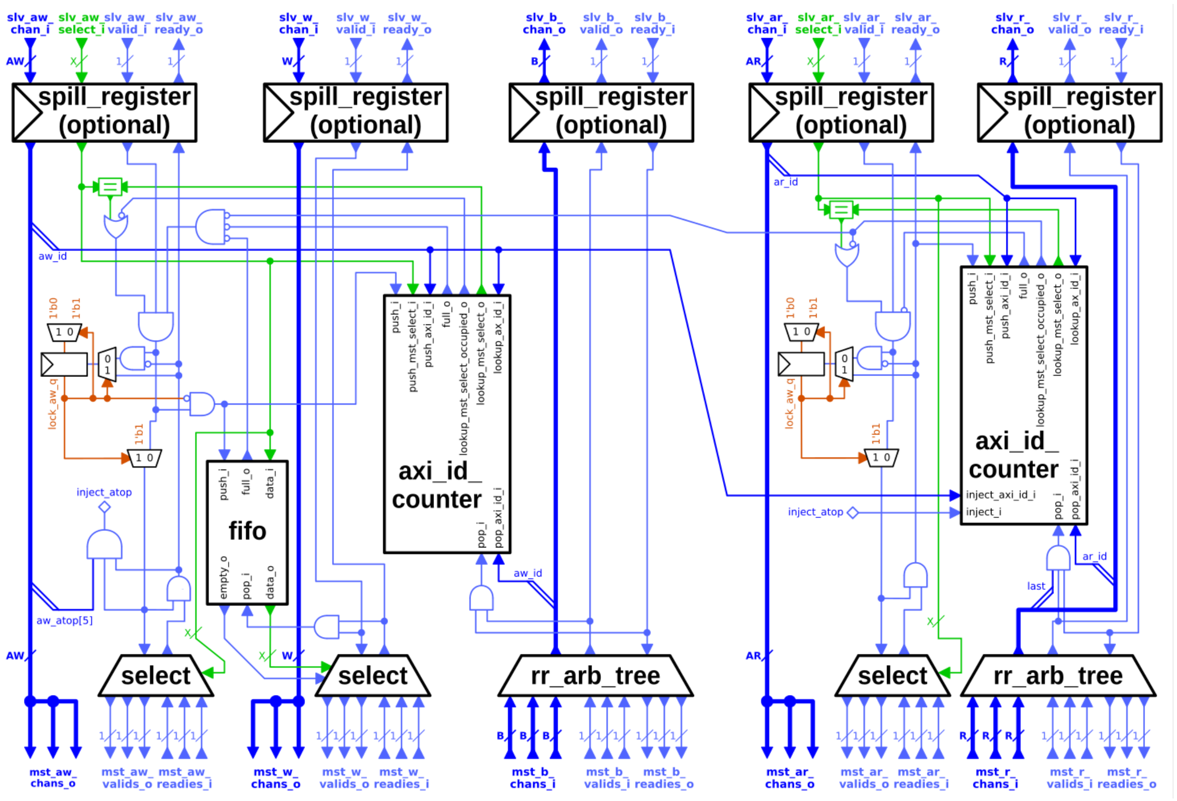


Figure 5.4. *AXI demultiplexer [56].*

According to the AXI protocol requirements when a demultiplexer gets multiple transactions with the same ID that go to different master ports, it has to deliver corresponding responses in the same order as requests were sent. Since downstream components connected to the demultiplexer are independent and cannot guarantee response order, the demultiplexer includes axi_id_counter component. This component contains a counter and index register for each possible ID value. When multiple transactions arrive with the same ID but target different master ports demultiplexer will stall those transactions that target the second master port until all responses will not reach the first master port. The index register will contain an index of the first master port. Each incoming request will increase the ID counter by one while every accepted response will decrease the counter by one. When the counter value is zero new requests can be sent that may go to another master port [54].

The AXI multiplexer is presented in Figure 5.5. This module merges multiple AXI slave connections to the single master port. The rr_arb_tree module implements a round-robin arbitration scheme. According to the AXI protocol W beats that belong to the different AW requests cannot be interleaved, so the last W beat that belongs to the first AW request has to be sent before the first W beat that belongs to the second AW request can be sent. To fulfil this requirement some bits of AW_ID get pushed into a FIFO module. When the FIFO is not empty the data determines which W slave port gets connected to the master port. When the last beat of a write transaction is sent, the data gets popped from FIFO.

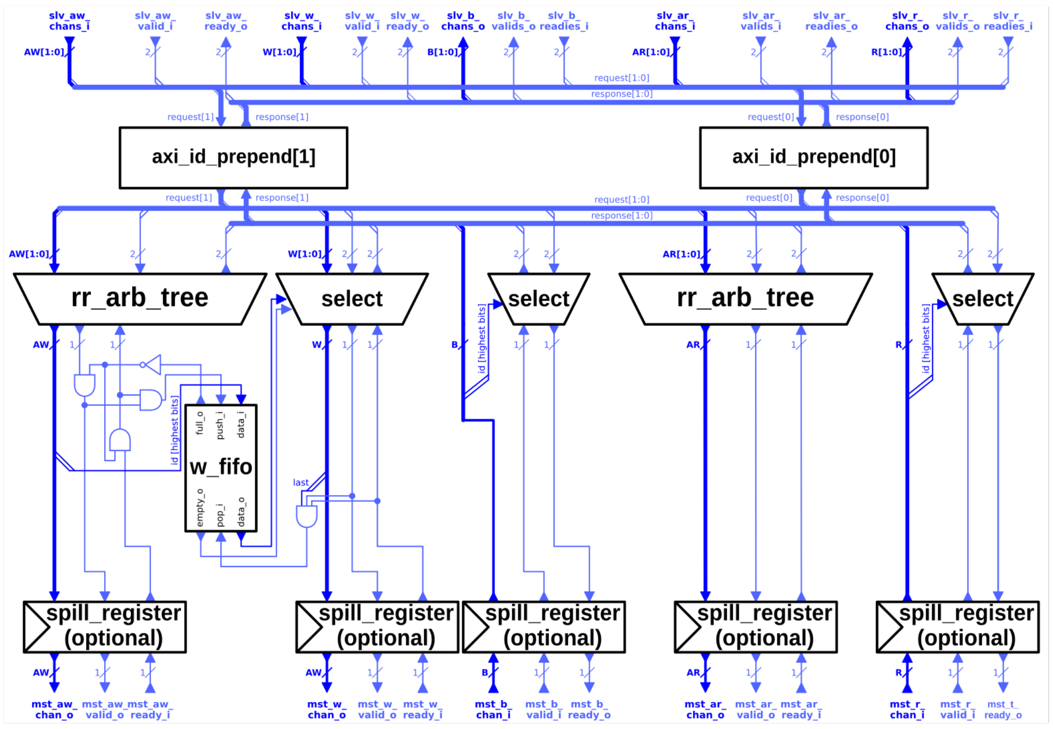

axi_id_prepend[1]
axi_id_prepend[0]
rr_arb_tree
select
select
rr_arb_tree
select
w_fifo
spill_register (optional)
spill_register (optional)
spill_register (optional)
spill_register (optional)
spill_register (optional)

Figure 5.5. *AXI multiplexer [55].*

When an AXI master initiates a transaction with a particular slave component, AWADDR and ARADDR values are used by the crossbar to route protocol requests according to the memory map. When some slave sends a response to the master via write response or read data channels, BID or RID identification values are used by a crossbar component to find the master which initiated the transaction. The original version of the crossbar component from the Pulp project repository prepends some unique bit values to the most significant part of AWID, WID and ARID to find what master initiated a transaction when the response arrives from slave component [54]. BID and RID values that came with response data have to be equal to the ID values that came with request. The crossbar component uses early prepended ID bits to find the request initiator and cuts these bits when sending a response to the upstream component. Such an approach works when interconnect consists of a single crossbar or when only a top-level crossbar has master port connection towards another crossbar.

In the Ballast chip, high performance and low performance interconnect have to be able to communicate in both directions, so both crossbars have a master port toward each other. In such a case, appending ID values to the most significant bits will produce a glitch condition. Let’s suppose that two crossbars that follow ID prepend strategy need to have master connections between them. The first crossbar has n number of ID bits to receive requests from upstream masters, then slave connections will have n+x1 ID bits (x1 is the number of the most significant bits which will be used to route responses). This means that the second crossbar will need n+x1 of ID beats from the master's side and n+x1+x2 ID bit towards its slave connections. In case a design includes a master connection from the second crossbar to the first one most significant bits will be lost and disable response routing mechanism. To handle this issue, Ballast development team altered the original crossbar implementation so that all masters and slaves have fixed sized ID widths and each crossbar use dedicated sensitivity bit indexes to route responses Figure 5.6.

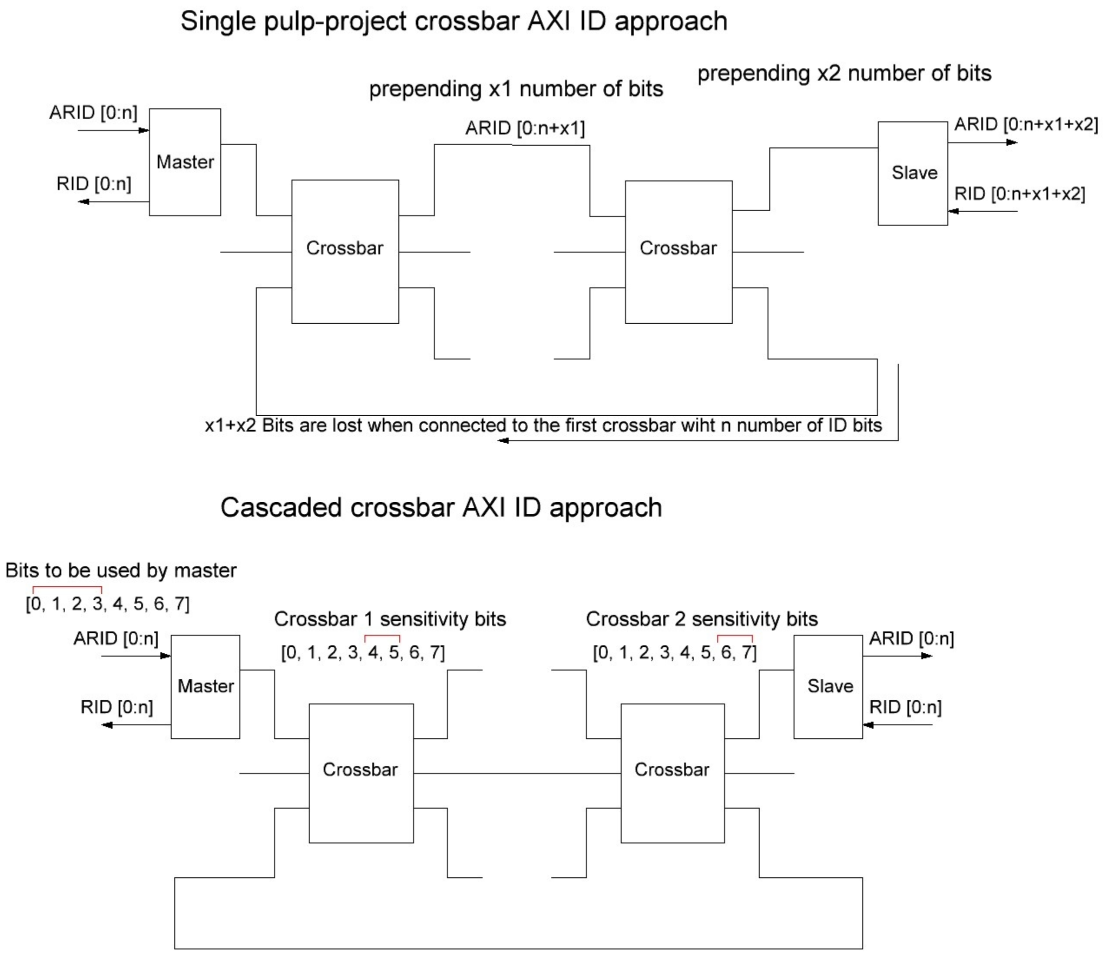

Single pulp-project crossbar AXI ID approach
prepending x1 number of bits
prepending x2 number of bits
ARID [0:n]
RID [0:n]
Master
ARID [0:n+x1]
Crossbar
Crossbar
Slave
ARID [0:n+x1+x2]
RID [0:n+x1+x2]
x1+x2 Bits are lost when connected to the first crossbar wiht n number of ID bits
Cascaded crossbar AXI ID approach
Bits to be used by master
[0, 1, 2, 3, 4, 5, 6, 7]
Crossbar 1 sensitivity bits
[0, 1, 2, 3, 4, 5, 6, 7]
Crossbar 2 sensitivity bits
[0, 1, 2, 3, 4, 5, 6, 7]
ARID [0:n]
RID [0:n]
Master
Crossbar
Crossbar
Slave
ARID [0:n]
RID [0:n]

Figure 5.6. *AXI ID for cascaded crossbar topology.*

To evaluate the design complexity of the Ballast interconnect the number of lines of code (LOC) was calculated for the interconnect submodules. Appendix A demonstrates the hierarchy of interconnect design files and HDL modules, gives LOC number for every module and mentions whether a module was designed by Ballast development team or given by Pulp project AXI library. In terms of unique source files, the total line count is LOC_{total} = 14589 where $LOC_{ballast}$ = 7038 contributed by Ballast project and LOC_{pulp} = 7551 come from Pulp AXI library. Note that the majority of Ballast design files contain module instantiations and build design hierarchy and topology, while PULP files mostly contribute to the actual AXI functionality. For better estimations of design complexity, we use another LOC calculation approach in which every unique design module contributes to the total LOC number even if the same design file was used to instantiate multiple different HDL modules. With this method LOC_{total} = 106508, $LOC_{ballast}$ = 30952 and LOC_{pulp} = 75608. According to these numbers, Pulp AXI library contributes more than 70% of the RTL source code and in our opinion gives more realistic estimations of design complexity contributions.

5.3 Ballast interconnect verification.

During the development of the Ballast chip, various techniques were used to test, debug and verify on-chip interconnect. The first approach that was used at the initial phases of the chip development relied on the testbench designed with System Verilog and Universal Verification Methodology (UVM) libraries[57]. Each connectivity port of the Ballast interconnect RTL model was connected with UVM classes that act as AXI masters or slave components Figure 5.7. The flexibility of the UVM environment allowed to quickly test and evaluate different interconnect architectures and find and resolve the majority of logical errors and bugs in the interconnect RTL implementation.

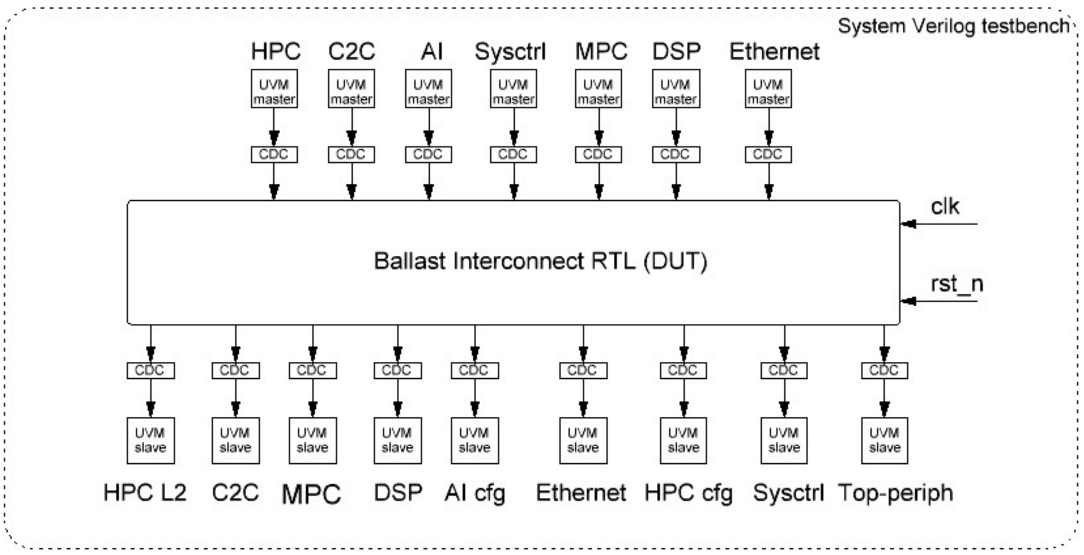


Figure 5.7. *Ballast UVM testbench.*

The second approach we used in Ballast simulation and verification is RTL virtual platform testbench. Once functionally correct RTL models of interconnect, Systcrtl CPU and other subsystems were available, Ballast team connected them to simulate the whole system behaviour. Firstly, RISC-V GCC compiler was used to compile software tests. Then testbench scripts converted compiled code into RAM vectors with address values and data bytes to be written into CPU RAMs. Besides RAM vectors virtual platform testbench includes test case scenarios to simulate debug modules commands such as CPU stall, breakpoints, system register writes etc. Then the special set of system Verilog functions is used to drive JTAG pins to simulate real debugger access, control CPU models execution, load RAM vectors with software tests and debug module commands Figure 5.8. The majority of developed software tests relied on communication between different subsystems and covered on-chip interconnect operation as well.

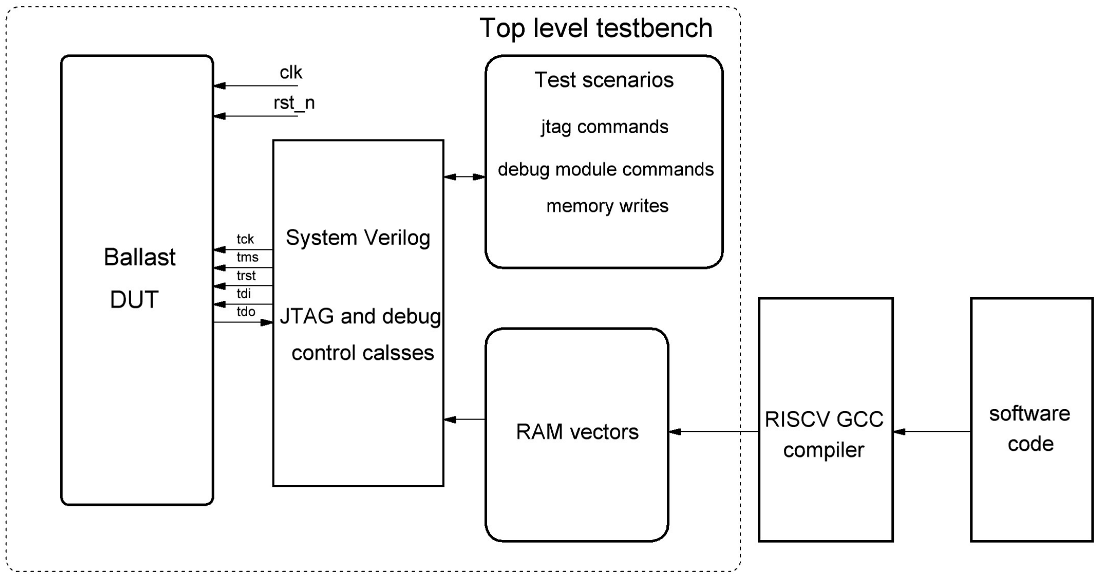


Figure 5.8. *Ballast RTL virtual platform.*

The RTL virtual platform concept allowed to use the same set of software tests with early RTL simulations, FPGA prototyping and Gate Level Simulations. The virtual platform approach in hardware design allowed us to adopt agile development techniques such as continuous integration and the automatic testing pipeline. In the middle stages of Ballast development, every RTL change, fix or new feature was tested by the testing pipeline before being merged with the master branch.

5.4 Ballast interconnect performance evaluations.

To evaluate achieved interconnect performance metrics we calculate maximum throughput numbers and minimal latency between every master and slave pair. The maximum bandwidths figures are mostly relevant for those masters who are capable to produce bus requests in long bursts or able to generate read and write requests every clock cycle without waiting for the response channel. Such types of bus accesses are usually produced by DMA devices to transfer large blocks of data. A CPU cache is another example of a device that relies on maximum bandwidth figures because it usually transfers data between memory in blocks, called cache lines.

According to the final synthesis results, the maximum clock frequency that can be achieved by HPICN logic is 1,2 GHz. The maximum theoretical throughput for each master is 1.2

***Table 5.1.** Ballast interconnect latencies.*

	HPC L2 slv	C2C slv	MPC slv	DSP slv	AI_cfg slv	Ethernet slv	HPC_cfg slv	Sysctrl slv	Top-periph slv
HPC mstr		6H 5 ns	7H 5,83 ns	6H+8L 17,56 ns	6H+8L 17,56 ns	6H+10L 20,70 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns
C2C mstr	6H 5 ns		7H 5,83 ns	6H+8L 17,56 ns	6H+8L 17,56 ns	6H+10L 20,70 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns
AI mstr	6H 5 ns	6H 5 ns	7H 5,83 ns	6H+8L 17,56 ns	6H+8L 17,56 ns	6H+10L 20,70 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns	6H+8L+5C 47,68 ns
Sysctrl mstr	6H+8L 17,56 ns	6H+8L 17,56 ns	7H+8L 18,39 ns	4L 6,28 ns	4L 6,28 ns	6L 9,42 ns	4L+5C 36,40 ns		4L+5C 36,40 ns
MPC mstr	6H+9L 19,12 ns	6H+9L 19,12 ns		5L 7,85 ns	5L 7,85 ns	7L 10,99 ns	5L+5C 37,97 ns	5L+5C 37,97 ns	5L+5C 37,97 ns
DSP mstr	6H+8L 17,56 ns	6H+8L 17,56 ns	7H+8L 18,39 ns		4L 6,28 ns	6L 9,42 ns	4L+5C 36,40 ns	4L+5C 36,40 ns	4L+5C 36,40 ns
Ethernet mstr	6H+10L 20,70 ns	6H+10L 20,70 ns	7H+10L 21,53 ns	6L 9,42 ns	6L 9,42 ns		6L+5C 39,54 ns	6L+5C 39,54 ns	6L+5C 39,54 ns

GHz × 8 bytes = 9.6 GB/s for the write channel and 9.6 GB/s for the read channel assuming that there is no competition between masters to access the same slave. Considering that HPICN clock frequency is higher than clock in chip-to-chip, HPC or AI subsystems, the high-performance interconnect is not expected to become a limiting factor for all possible use cases. The maximum clock frequency for the low performance interconnect is 637 MHz and the maximum theoretical throughput for each LPICN master is 637 MHz × 4 bytes = 2.548 GB/s for the write channel and 2.548 GB/s for the read channel. The MPC subsystem is only capable of single burst transactions and not expected to exceed LPICN bandwidth. The DSP subsystem runs at 600 MHz clock frequency and is supposed to utilize the full capabilities of LPICN and intensively communicate with HPICN. ConfigICN runs at a maximum of 166 MHz with maximum bandwidth equal to 166MHz× 4 bytes = 664 MB/s.

The latency is the most relevant parameter to evaluate bus performance when a bus master mostly produces single-word accesses or short bursts. For example, MPC or Sysctrl subsystem doesn’t have any CPU cache and completely stalls CPU pipeline when performing load or store operation until a protocol request fully propagates to its destination and the response returns back from slave to master. Every crossbar has a fixed latency of two clock cycles. Some latency is also added by clock-domain crossing (CDC) bridges and slice registers. The CDC bridges are necessary to connect logic circuits that operate with different clock sources. Slice registers are added to split long interconnect wires and increase the maximum clock frequency. The Ballast interconnect latencies for each master are shown in Table 5.1.

Table 5.1 demonstrates request latencies for each master and slave. The first row in each cell represents the latency calculation method, the second row shows approximate latency for the maximum frequency values. In calculation formulas letter H depicts high performance interconnect clock period, which equals 1 ÷ 1.2 GHz = 0.833 ns, letter L is low performance interconnect clock period which equals 1 ÷ 637 MHz = 1.57 ns, C refers to configurational interconnect clock period of 1 ÷ 166 MHz = 6 ns. The crossbar components have a fixed latency of 2 clock cycles by design. Protocol converters are implemented as state machines with registered output and have a latency of 2 clock cycles as well. The clock bridges are implemented as double clocked FIFOs. When calculating CDC latency between interconnects we add one clock period of the slowest clock (for example, in our method CDC bridge between LPICN and ConfigICN adds 6ns of latency). The other latencies are added by slice registers and vary for each master or slave port. Note that AXI protocol [50] consists of multiple communication phases and this table shows only one-way request latency illustrated in Figure 5.9. The table do not includes response latency (backward path shown in Figure 5.9). The response latency is equal to the request latency. This table does not include internal data path latency needed for data processing inside of slave modules and ignores latencies of clock bridges between subsystems and interconnect.

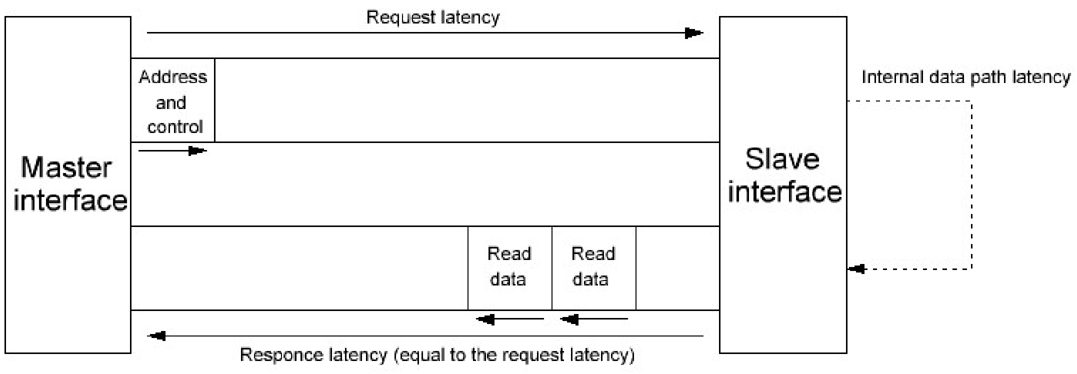


Figure 5.9. *AXI protocol latency illustration.*

6. CONCLUSION

The design of an on-chip interconnect is a complicated engineering task that requires careful analysis of design requirements and many compromise decisions between maximal throughput, minimal latency, power consumption, chip area, reliability and scalability features. In this thesis we explored various on-chip interconnect design technologies and provide a report about interconnect design of the new Ballast chip.

Firstly, we analyzed the basics and current technology state of bus-based interconnect designs such as shared buses and hierarchical buses. A shared bus or hierarchical bus interconnect is a preferable option for small scale or low latency designs with few bus masters and slaves. The bus-based interconnects occupy a very small chip area, consume a limited amount of power, communicate with small latency but demonstrate a lack of scalability and can not sustain high throughput for large scale designs, since all masters communicate via one or few communication channels and compete for shared resources.

To cope with scalability and throughput limitations of shared buses, crossbar based designs were adopted by the research and industrial community. The crossbar or bus matrix provides a dedicated communication link for each master and slave pair and can sustain high data throughput which does not depend on the number of master or slave components. As a drawback, the bus matrix consumes a larger area and power. As a trade-off solution, chip architectures may use a hybrid approach connecting a single crossbar with shared bus blocks or connecting multiple smaller crossbars in a cascaded manner. Such compromise designs can efficiently interconnect dozens of masters and slaves [11].

The network-on-chip is another modern interconnect design technology that heavily relies on network and communication theory. In modern large scale chips long communication wires, interference and parasitic effects became a real practical issue for robust communication. The network on chip relies on packet transfers and error recovery algorithms to transmit data across long distances. The packet transfers require the use of complicated routing logic, buffers and crossbar blocks. Network-on-chips consume a large chip area and power, transfer signals with relatively high latency but provide high throughput and remain the only interconnect option for large scale chips with hundreds of IP blocks.

The Ballast is the novel chip designed by Tampere University and SoC Hub consortium. The Ballast system-on-chip was designed for edge computing, data acquisition, robotics and general computing applications for research and industrial purposes. The chip features 6 masters and 9 slave components to be connected with the global interconnect. After careful analysis of the expected data traffic and other interconnect requirements Ballast team decided to use a cascaded crossbar approach. The Ballast chip interconnect consists of three AXI crossbar blocks with adjustable clock sources: the high performance interconnect, low performance interconnect and configurational interconnect. The modular approach allows to provide enough throughput for every master component with tolerable latency and saves chip area for on-chip IP blocks. The implemented interconnect demonstrates how to use AXI crossbars in hierarchical manner to design interconnects for the future SoC Hub chips with reduced efforts.

To improve interconnect design approach for the future SoC hub chips, we can propose to design an automatic software or script to generate RTL code for specified interconnect topologies from basic building blocks such as AXI crossbars, protocol converters, CDC bridges and slice registers. Such a tool in combination with UVM test benches and RTL virtual platform will allow a quick evaluation and testing of many different interconnect architectures with different number of master and slave connections and will help to find an optimal solution for future SoC Hub chips.

REFERENCES

[1] *ITRS Reports*. URL: http://www.itrs2.net/itrs-reports.html (visited on 27/03/2022).

[2] *SoC Hub: A new ecosystem initiative for a world-class co-developed Finnish System-on-Chip.* URL: https://sochub.fi/ (visited on 27/03/2022).

[3] Pasricha, S. and Dutt, N. *On-chip communication architectures system on chip interconnect*. Morgan Kaufmann, 2008.

[4] Kulmala, A. Scalable Multiprocessor System-on-chip Architecture Design on FPGA. PhD thesis. Tampere University of Technology, 2009.

[5] Nurmi, J. et al. Interconnect-Centric design for Advanced Soc and Noc. (2004).

[6] Benini, L. and Micheli, G. D. Networks on chips: a new SoC paradigm. *Computer* 35 (2002), pp. 70–78.

[7] Salminen, E., Kangas, T., Hämäläinen, T., Riihimäki, J., Lahtinen, V. and Kuusilinna, K. HIBI communication network for system-on-chip. 43 (2006), pp. 185–205.

[8] Ryu, K. K. and Mooney, V. Automated bus generation for multiprocessor SoC design. eng. *IEEE transactions on computer-aided design of integrated circuits and systems* 23.11 (2004), pp. 1531–1549.

[9] Pasricha, S., Dutt, N. and Ben-Romdhane, M. Constraint-driven bus matrix synthesis for MPSoC. *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*. IEEE Press, 2006, pp. 30–35.

[10] Jun, M., Yoo, S. and Chung, E.-Y. Mixed integer linear programming-based optimal topology synthesis of cascaded crossbar switches. *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. ASP-DAC '08. IEEE Computer Society Press, 2008, pp. 583–588.

[11] Yoo, J., Lee, D., Yoo, S. and Choi, K. Communication Architecture Synthesis of Cascaded Bus Matrix. *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 171–177.

[12] *Arm CoreLink CCI-550 Cache Coherent Interconnect Technical Reference Manual*. URL: https//developer.arm.com/documentation/100282/0100/ (visited on 27/03/2022).

[13] *LogiCORE AXI Interconnect v2.1 Product Guide*. URL: https://docs.xilinx.com/v/u/en-US/pg059-axi-interconnect/ (visited on 27/03/2022).

[14] *AXI SystemVerilog Modules for High-Performance On-Chip Communication*. URL: https://github.com/pulp-platform/axi/ (visited on 27/03/2022).

[15] Lahtinen, V., Salminen, E., Kuusilinna, K. and Hämäläinen, T. Comparison of synthesized bus and crossbar interconnection architecures. *IEEE International Symposium on Circuits and Systems*. Vol. 5. 2003, pp. 433–436.

[16] Angiolini, F., Meloni, P., Carta, S., Raffo, L. and Benini, L. A Layout-Aware Analysis of Networks-on-Chip and Traditional Interconnects for MPSoCs. *IEEE transactions on computer-aided design of integrated circuits and systems* 26 (2007), pp. 421–434.

[17] *PULPissimo microcontroller open-source implementation*. URL: https://github.com/pulp-platform/pulpissimo/ (visited on 27/03/2022).

[18] Schiavone, P. D., Rossi, D., Pullini, A., Di Mauro, A., Conti, F. and Benini, L. Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX. *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. 2018, pp. 1–3.

[19] Pasricha, S., Dutt, N., Bozorgzadeh, E. and Ben-Romdhane, M. Floorplan-aware automated synthesis of bus-based communication architectures. *Proceedings - Design Automation Conference*. 2005, pp. 565–570.

[20] Murali, S. and Micheli, G. An Application-Specific Design Methodology for STbus Crossbar Generation. *Proceedings of the conference on design, automation and test in europe*. Vol. 2. 5. IEEE Computer Society, 2005, pp. 1176–1181.

[21] Jun, M., Yoo, S. and Chung, E.-Y. Topology Synthesis of Cascaded Crossbar Switches. *IEEE transactions on computer-aided design of integrated circuits and systems* 28 (2009), pp. 926–930.

[22] Kulmala, A., Salminen, E. and Hämäläinen, T. Distributed bus arbitration algorithm comparison on FPGA-based MPEG-4 multiprocessor system on chip. *IET computers digital techniques* 2 (2008), pp. 314–325.

[23] Lahiri, K., Raghunathan, A. and Lakshminarayana, G. The LOTTERYBUS on-chip communication architecture. *IEEE transactions on very large scale integration (VLSI) systems* 14 (2006), pp. 596–608.

[24] Poletti, F., Bertozzi, D., Benini, L. and Bogliolo, A. Performance Analysis of Arbitration Policies for SoC Communication Architectures. *Design automation for embedded systems* 8 (2003), pp. 189–210.

[25] Dally, W. and Towles, B. Route packets, not wires: on-chip interconnection networks. *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. IEEE, 2001, pp. 684–689.

[26] Dally, W. J. and Poulton, J. W. *DigitalSystems Engineering*. Cambridge University Press, 1998.

[27] Ackland, B., Anesko, A., Brinthaupt, D., Daubert, S., Kalavade, A., Knobloch, J., Micca, E., Moturi, M., Nicol, C., O'Neill, J., Othmer, J., Sackinger, E., Singh, K., Sweet, J., Terman, C. and Williams, J. A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP. *IEEE journal of solid-state circuits* 35 (2000), pp. 412–424.

[28] Benini, L. and De Micheli, G. Powering networks on chips. *International Symposium on System Synthesis (IEEE Cat. No.01EX526)*. IEEE, 2001, pp. 33–38.

[29] Dally, W. Virtual-channel flow control. *IEEE transactions on parallel and distributed systems* 3 (1992), pp. 194–205.

[30] Dimitrakopoulos, G., Psarras, A. and Seitanidis, I. *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*. Springer New York, 2015, pp. 1–175.

[31] Nikolic, T., Djordjevic, G. and Stojcev, M. Simultaneous data transfers over peripheral bus using CDMA technique. *2008 26th International Conference on Microelectronics*. IEEE, 2008, pp. 437–440.

[32] Ahmed, K. E., Rizk, M. R. and Farag, M. M. Overloaded CDMA Crossbar for Network-On-Chip. *IEEE transactions on very large scale integration (VLSI) systems* 25 (2017). ISSN: 1063-8210.

[33] Wang, X., Ahonen, T. and Nurmi, J. Applying CDMA Technique to Network-on-Chip. *IEEE transactions on very large scale integration (VLSI) systems* 15 (2007).

[34] Xue, Y., Qian, Z., Wei, G., Bogdan, P., Tsui, C.-Y. and Marculescu, R. An efficient Network-on-Chip (NoC) based multicore platform for hierarchical parallel genetic algorithms. eng. *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. IEEE, 2014, pp. 17–24.

[35] Tatas, K. *Designing 2D and 3D Network-on-Chip Architectures*. 1st ed. 2014. Springer New York, 2014.

[36] Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., Tiensyrja, K. and Hemani, A. A network on chip architecture and design methodology. *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. IEEE, 2002, pp. 117–124.

[37] Reehal, G., Ghany, M. A. A. E. and Ismail, M. Octagon architecture for low power and high performance NoC design. *2012 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE, 2012, pp. 63–67.

[38] Kariniemi, H. *On-line reconfigurable extended generalized fat tree network-on-chip for multiprocessor system-on-chip circuits*. 2006.

[39] *RISC-V instruction set architecture (ISA) and related specifications.* URL: https://riscv.org/technical/specifications/ (visited on 27/03/2022).

[40] *Ibex: An embedded 32 bit RISC-V CPU core.* URL: https://ibex-core.readthedocs.io/en/latest/ (visited on 27/03/2022).

[41] *OpenHW Group CV32E40P User Manual.* URL: https://docs.openhwgroup.org/projects/cv32e40p-user-manual/ (visited on 27/03/2022).

[42] *Ariane RISC-V CPU.* URL: https://github.com/lowRISC/ariane/ (visited on 27/03/2022).

[43] Zaruba, F. and Benini, L. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FD-

SOI Technology. *IEEE transactions on very large scale integration (VLSI) systems* 27 (2019), pp. 2629–2640.

[44] *RISC-V Debug Support for PULP Cores.* URL: https : / / github . com / pulp - platform/riscv-dbg/ (visited on 27/03/2022).

[45] *NVIDIA Deep Learning Accelerator (NVDLA) free and open architecture.* URL: http://nvdla.org/ (visited on 27/03/2022).

[46] *10_100_1000 Mbps tri-mode ethernet MAC.* URL: https : / / opencores . org / projects/ethernet_tri_mode/ (visited on 27/03/2022).

[47] *TTA-based Co-Design Environment.* URL: http : / / openasip . org/ (visited on 27/03/2022).

[48] *FlexNoC Interconnect IP*. URL: https://www.arteris.com/flexnoc/ (visited on 27/03/2022).

[49] *Highly Configurable Topology for SoC Connectivity*. URL: https : / / www . arm . com/products/silicon-ip-system/corelink-interconnect/nic/ (visited on 27/03/2022).

[50] *AMBA AXI and ACE Protocol Specification Version.* URL: https://developer. arm . com / documentation / ihi0022 / e / AMBA - AXI3 - and - AXI4 - Protoco Specification/ (visited on 27/03/2022).

[51] *AXI data width upsizer.* URL: https : / / github . com / pulp - platform / axi / blob/master/src/axi_dw_upsizer.sv/ (visited on 27/03/2022).

[52] *AXI data width downsizer.* URL: https://github.com/pulp-platform/axi/ blob/master/src/axi_dw_downsizer.sv/ (visited on 27/03/2022).

[53] *AXI Fully-Connected Crossbar.* URL: https://github.com/pulp- platform/ axi/blob/master/doc/axi_xbar.md/ (visited on 27/03/2022).

[54] Kurth, A., Ronninger, W., Benz, T., Cavalcante, M., Schuiki, F., Zaruba, F. and Benini, L. An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication. *IEEE transactions on computers* (2021), pp. 1–1.

[55] *AXI protocol multiplexer.* URL: https://github.com/pulp- platform/axi/ blob/master/doc/axi_mux.md/ (visited on 27/03/2022).

[56] *AXI protocol demultiplexer.* URL: https://github.com/pulp-platform/axi/ blob/master/doc/axi_demux.md/ (visited on 27/03/2022).

[57] *Standard Universal Verification Methodology*. URL: https://www.accellera. org/downloads/standards/uvm/ (visited on 27/03/2022).

APPENDIX A: BALLAST INTERCONNECT HIERARCHY

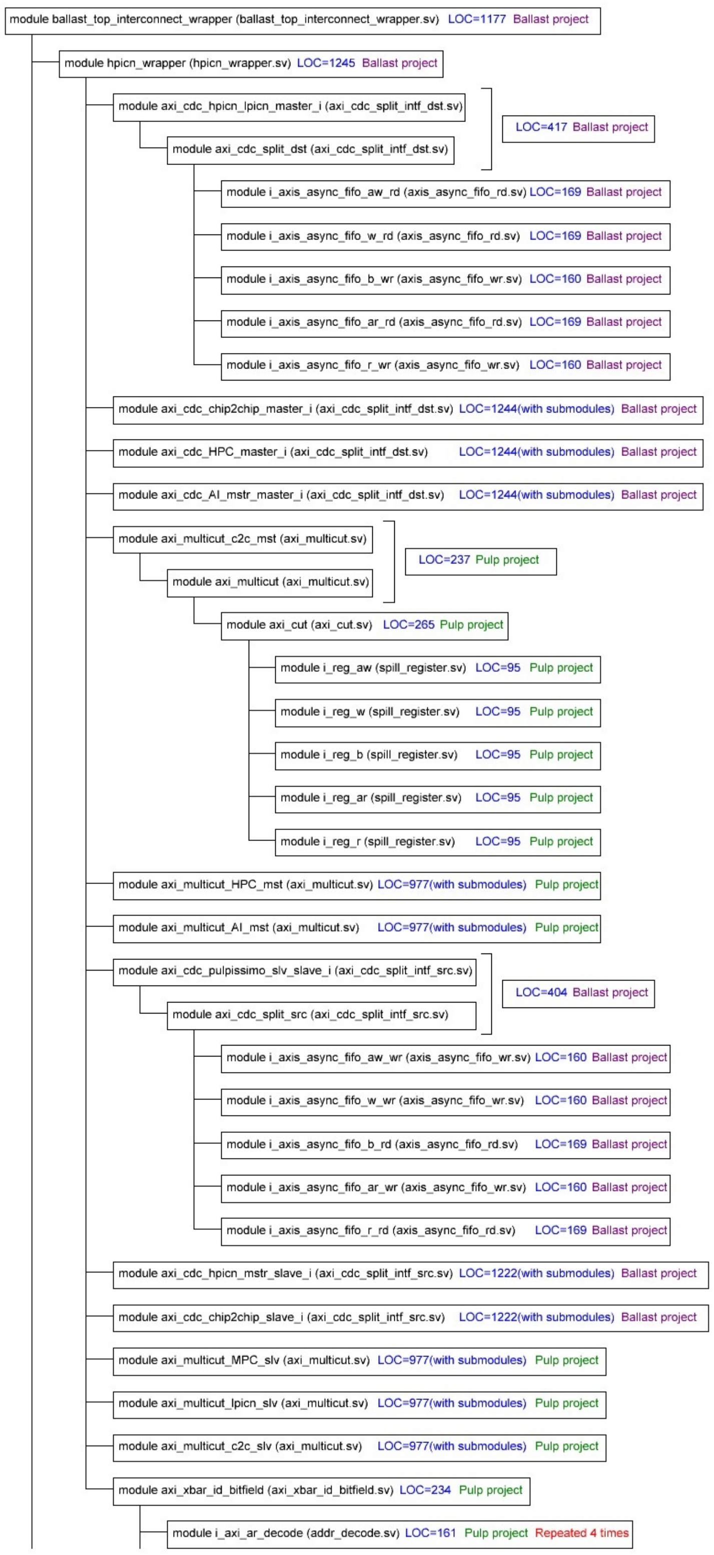

module ballast_top_interconnect_wrapper (ballast_top_interconnect_wrapper.sv) LOC=1177 Ballast project
module hpicn_wrapper (hpicn_wrapper.sv) LOC=1245 Ballast project
module axi_cdc_hpicn_lpicn_master_i (axi_cdc_split_intf_dst.sv)
module axi_cdc_split_dst (axi_cdc_split_intf_dst.sv)
LOC=417 Ballast project
module i_axis_async_fifo_aw_rd (axis_async_fifo_rd.sv) LOC=169 Ballast project
module i_axis_async_fifo_w_rd (axis_async_fifo_rd.sv) LOC=169 Ballast project
module i_axis_async_fifo_b_wr (axis_async_fifo_wr.sv) LOC=160 Ballast project
module i_axis_async_fifo_ar_rd (axis_async_fifo_rd.sv) LOC=169 Ballast project
module i_axis_async_fifo_r_wr (axis_async_fifo_wr.sv) LOC=160 Ballast project
module axi_cdc_chip2chip_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_HPC_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_AI_mstr_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_multicut_c2c_mst (axi_multicut.sv)
module axi_multicut (axi_multicut.sv)
LOC=237 Pulp project
module axi_cut (axi_cut.sv) LOC=265 Pulp project
module i_reg_aw (spill_register.sv) LOC=95 Pulp project
module i_reg_w (spill_register.sv) LOC=95 Pulp project
module i_reg_b (spill_register.sv) LOC=95 Pulp project
module i_reg_ar (spill_register.sv) LOC=95 Pulp project
module i_reg_r (spill_register.sv) LOC=95 Pulp project
module axi_multicut_HPC_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_AI_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_cdc_pulpissimo_slv_slave_i (axi_cdc_split_intf_src.sv)
module axi_cdc_split_src (axi_cdc_split_intf_src.sv)
LOC=404 Ballast project
module i_axis_async_fifo_aw_wr (axis_async_fifo_wr.sv) LOC=160 Ballast project
module i_axis_async_fifo_w_wr (axis_async_fifo_wr.sv) LOC=160 Ballast project
module i_axis_async_fifo_b_rd (axis_async_fifo_rd.sv) LOC=169 Ballast project
module i_axis_async_fifo_ar_wr (axis_async_fifo_wr.sv) LOC=160 Ballast project
module i_axis_async_fifo_r_rd (axis_async_fifo_rd.sv) LOC=169 Ballast project
module axi_cdc_hpicn_mstr_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_chip2chip_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_multicut_MPC_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_lpicn_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_c2c_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_xbar_id_bitfield (axi_xbar_id_bitfield.sv) LOC=234 Pulp project
module i_axi_ar_decode (addr_decode.sv) LOC=161 Pulp project Repeated 4 times

Figure A.1. *Ballast interconnect HDL hierarchy.*

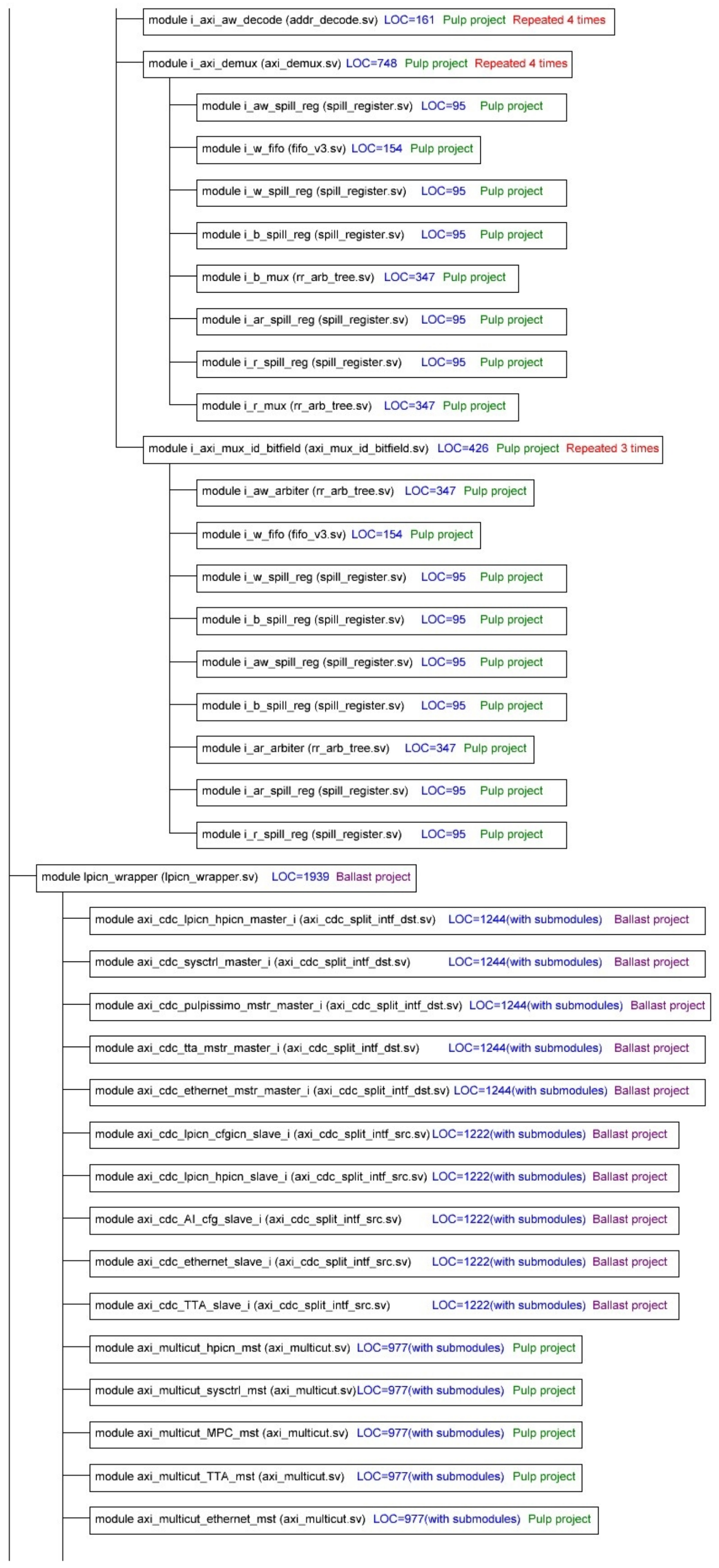

module i_axi_aw_decode (addr_decode.sv) LOC=161 Pulp project Repeated 4 times
module i_axi_demux (axi_demux.sv) LOC=748 Pulp project Repeated 4 times
module i_aw_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_w_fifo (fifo_v3.sv) LOC=154 Pulp project
module i_w_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_b_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_b_mux (rr_arb_tree.sv) LOC=347 Pulp project
module i_ar_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_r_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_r_mux (rr_arb_tree.sv) LOC=347 Pulp project
module i_axi_mux_id_bitfield (axi_mux_id_bitfield.sv) LOC=426 Pulp project Repeated 3 times
module i_aw_arbiter (rr_arb_tree.sv) LOC=347 Pulp project
module i_w_fifo (fifo_v3.sv) LOC=154 Pulp project
module i_w_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_b_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_aw_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_b_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_ar_arbiter (rr_arb_tree.sv) LOC=347 Pulp project
module i_ar_spill_reg (spill_register.sv) LOC=95 Pulp project
module i_r_spill_reg (spill_register.sv) LOC=95 Pulp project
module lpicn_wrapper (lpicn_wrapper.sv) LOC=1939 Ballast project
module axi_cdc_lpicn_hpicn_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_sysctrl_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_pulpissimo_mstr_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_tta_mstr_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_ethernet_mstr_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_lpicn_cfgicn_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_lpicn_hpicn_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_AI_cfg_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_ethernet_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_TTA_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_multicut_hpicn_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_sysctrl_mst (axi_multicut.sv)LOC=977(with submodules) Pulp project
module axi_multicut_MPC_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_TTA_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_ethernet_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project

Figure A.1. *Ballast interconnect HDL hierarchy(cont).*

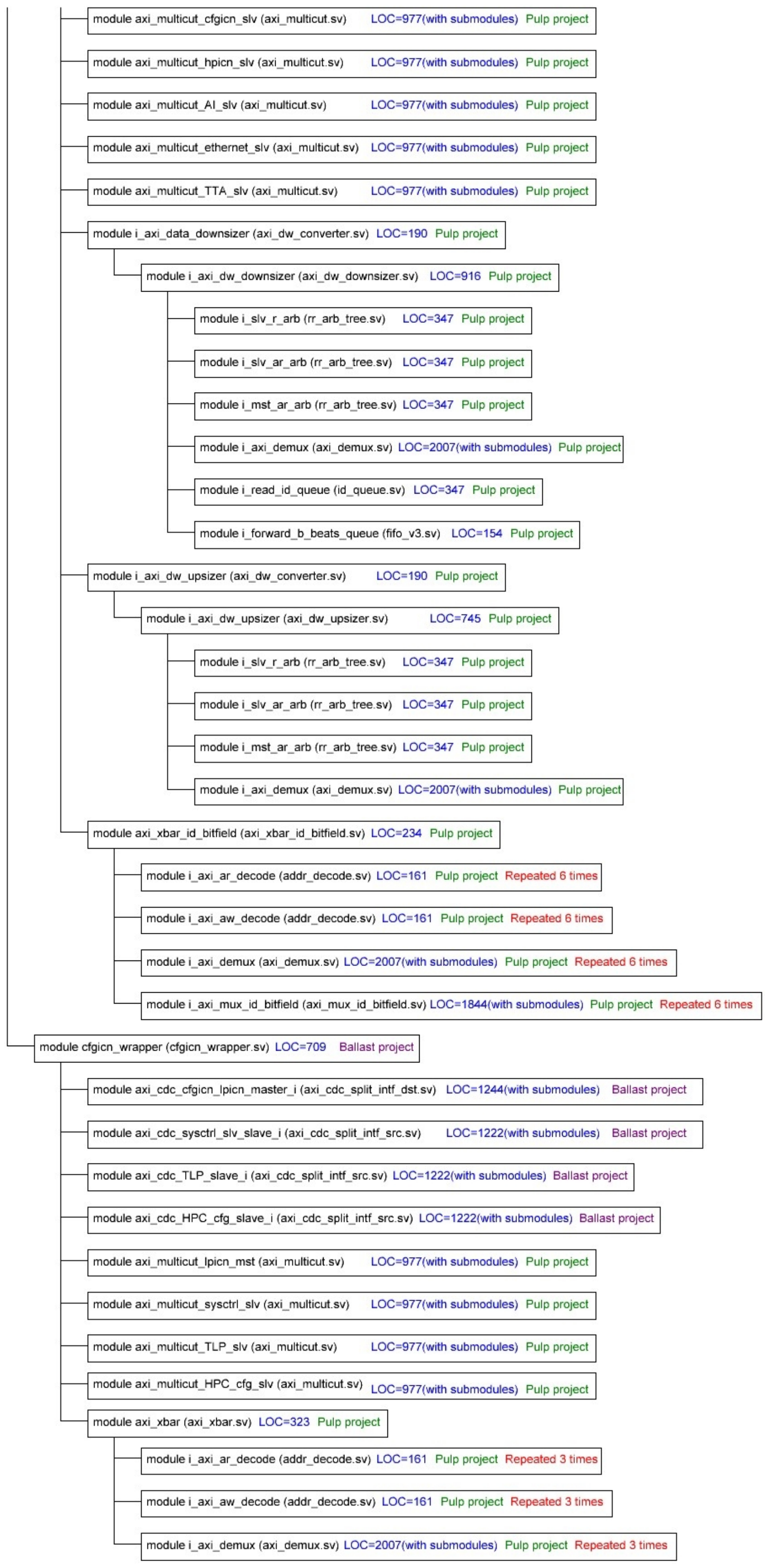

module axi_multicut_cfgicn_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_hpicn_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_AI_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_ethernet_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_TTA_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module i_axi_data_downsizer (axi_dw_converter.sv) LOC=190 Pulp project
module i_axi_dw_downsizer (axi_dw_downsizer.sv) LOC=916 Pulp project
module i_slv_r_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_slv_ar_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_mst_ar_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_axi_demux (axi_demux.sv) LOC=2007(with submodules) Pulp project
module i_read_id_queue (id_queue.sv) LOC=347 Pulp project
module i_forward_b_beats_queue (fifo_v3.sv) LOC=154 Pulp project
module i_axi_dw_upsizer (axi_dw_converter.sv) LOC=190 Pulp project
module i_axi_dw_upsizer (axi_dw_upsizer.sv) LOC=745 Pulp project
module i_slv_r_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_slv_ar_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_mst_ar_arb (rr_arb_tree.sv) LOC=347 Pulp project
module i_axi_demux (axi_demux.sv) LOC=2007(with submodules) Pulp project
module axi_xbar_id_bitfield (axi_xbar_id_bitfield.sv) LOC=234 Pulp project
module i_axi_ar_decode (addr_decode.sv) LOC=161 Pulp project Repeated 6 times
module i_axi_aw_decode (addr_decode.sv) LOC=161 Pulp project Repeated 6 times
module i_axi_demux (axi_demux.sv) LOC=2007(with submodules) Pulp project Repeated 6 times
module i_axi_mux_id_bitfield (axi_mux_id_bitfield.sv) LOC=1844(with submodules) Pulp project Repeated 6 times
module cfgicn_wrapper (cfgicn_wrapper.sv) LOC=709 Ballast project
module axi_cdc_cfgicn_lpicn_master_i (axi_cdc_split_intf_dst.sv) LOC=1244(with submodules) Ballast project
module axi_cdc_sysctrl_slv_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_TLP_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_cdc_HPC_cfg_slave_i (axi_cdc_split_intf_src.sv) LOC=1222(with submodules) Ballast project
module axi_multicut_lpicn_mst (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_sysctrl_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_TLP_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_multicut_HPC_cfg_slv (axi_multicut.sv) LOC=977(with submodules) Pulp project
module axi_xbar (axi_xbar.sv) LOC=323 Pulp project
module i_axi_ar_decode (addr_decode.sv) LOC=161 Pulp project Repeated 3 times
module i_axi_aw_decode (addr_decode.sv) LOC=161 Pulp project Repeated 3 times
module i_axi_demux (axi_demux.sv) LOC=2007(with submodules) Pulp project Repeated 3 times

Figure A.1. *Ballast interconnect HDL hierarchy(cont).*