

SCIENCES

Electronics Engineering, Field Director – Francis Balestra

Design Methodologies and Architecture,
Subject Head – Ahmed Jerraya

Multi-Processor System-on-Chip 1

Architectures

Coordinated by
Liliana Andrade
Frédéric Rousseau

ISTE

WILEY

First published 2020 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

© ISTE Ltd 2020

The rights of Liliana Andrade and Frédéric Rousseau to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Control Number: 2020940076

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78945-021-7

ERC code:

PE6 Computer Science and Informatics

PE6_1 Computer architecture, pervasive computing, ubiquitous computing

PE6_10 Web and information systems, database systems, information retrieval and digital libraries, data fusion

PE7 Systems and Communication Engineering

PE7_2 Electrical engineering: power components and/or systems

PART 3

Interconnect and Interfaces

9

Network-on-Chip (NoC): The Technology that Enabled Multi-processor Systems-on-Chip (MPSoCs)

K. Charles JANAC

Arteris IP, Campbell, USA

Over the last two decades, networks-on-chip (NoCs) have advanced from PhD theses to being the key technology enabling the creation of today's production of multi-billion transistor multi-core systems-on-chip (SoCs). This chapter first explains the history of on-chip interconnect technology, from buses to crossbars and finally NoCs, and then explains the key technical components of NoCs and the metrics that make them fit for a particular purpose. Methods of NoC configurability are explained, including component assembly and synthesis approaches. System-level services implemented by NoC technologies are explained, including quality-of-service (QoS)/bandwidth and latency guarantees, voltage domains, frequency domains, security, debug, performance profiling and functional safety. The chapter then explains the use of NoCs to carry more advanced communications semantics and explains how NoCs have led to a new era of heterogeneous cluster cache coherency and inter-die communications. Finally, the chapter speculates on the future of NoC technology including floorplan-guided automatic NoC topology synthesis, on-chip distance spanning technologies, chiplets and integrated diagnostics to optimize system-level runtime performance, safety and security.

For a color version of all figures in this book, see www.iste.co.uk/andrade/multi1.zip.

Multi-Processor System-on-Chip 1 – Architectures,
coordinated by Liliana ANDRADE and Frédéric ROUSSEAU. © ISTE Ltd 2020.

9.1. History: transition from buses and crossbars to NoCs

On a system-on-chip (SoC), intellectual property blocks (IPs) communicate over an interconnect system using transactions. A transaction is the operation of reading or writing bytes of data to and from addresses in a memory space. Transactions are composed of a *request*, which goes from an *initiator* to a *target* and contains at least the address and data in the case of a write operation, and a *response* flowing back from the *target* to the *initiator*, with status and data in case of read operation. The physical interfaces between IPs and the interconnect system are called *sockets*, usually synchronous, with a *master* side and a *slave* side. A number of different socket protocols have been defined by the SoC industry (Arm® AMBA® AXI, APB and AHB, OCP, etc.). Sometimes, transactions have to be converted between the initiator socket and the target socket of an interconnect, for instance, if both socket protocols are not identical. Such conversions must preserve the byte transfer semantics, i.e. the functionality of the transaction.

Interconnect systems have gradually evolved from simple buses to more scalable networks-on-chip (NoCs). It could be tempting to consider a NoC as a downscaled TCP/IP or ATM network; it is not. TCP/IP and ATM are connection-based networks, where connections are dynamically opened and closed between producer (often called *initiator* or master) and consumer (also called *target* or slave) ports. History shows that small systems are transaction-based, while large systems are connection-based. Attempts to design small connection-based systems (INMOS transputers), or large transaction-based systems (distributed shared memory PC clustering), have up to now failed. A time may come when SoCs are so huge that a connection-based network is required between transaction-based clusters, but this is not yet the case.

In the 1980s and 1990s, buses were the on-chip interconnect technology implemented in virtually all system-on-chip (SoC) semiconductor designs. Buses were typically handcrafted around either a specific set of features relevant to a narrow target market or support for a specific processor. On-chip buses were essentially implemented as scaled-down versions of board-level interconnects implemented in the 1980s (Benini and De Micheli 2002).

The next evolution of bus interconnects was the hybrid bus, which included a centralized crossbar switch with all the internal IP connections needed for the chip. This presented capacity and routing congestion problems as SoCs became more complex, so multi-layer hybrid buses were the next evolution of the interconnect technology. However, these multi-layer hybrid buses posed latency, frequency, power and area problems as SoC complexity continued to evolve.

Several trends forced evolutions of systems architectures that drove evolutions of required on-chip interconnect technologies. These trends are:

- Application convergence: the mixing of various traffic types in the same SoC design (video, communication, computing, etc.). These traffic types, although very

different in nature, for example, from the quality-of-service (QoS) point of view, had to share resources, typically a unified off-chip memory, that were assumed to be “private” and handcrafted to the particular traffic in previous designs.

- Moore’s law drove the integration of many IP blocks into a single chip. This is an enabler to application convergence, but it also allowed entirely new approaches (parallel processing on a chip using many small processors) or simply allowed SoCs to process more data streams (such as communication channels).

- IP reuse: the pervasive reuse of existing IP blocks is an absolute necessity. The shapes those to-be-reused IPs can take is quite variable: hard (layout-ready) or soft, in-house or third-party, simple (USB) or complex (complete H.264 decompression engine). The NoC interconnect has to convey transactions between sockets of different protocols (OCP, AXI, proprietary, etc.), each protocol being highly parameterized. This variety has major impacts from a performance perspective. The IPs may have to be clocked at different frequencies, and their sockets may present different data widths, leading to a wide range of peak throughputs, which must be adapted between initiators and targets. Initiators have different traffic patterns, in terms of transaction lengths, address alignments and regularity. Their reactions and resistance to latency or transient backpressure differ as well. Also, meeting the (QoS) required by each initiator despite the presence of the other traffic becomes more challenging as the number of IPs increases.

- Arrival of 40 nm silicon processes that allowed the building of SoCs with a higher level of functional integration. A consequence of silicon process evolution was that gates cost relatively less than wires, both from an area and performance perspective, than with previous process generations. Transistor logic gates scaled as silicon process critical dimensions decreased, but wires did not shrink at the same level because wire resistance parasitics became increasingly important.

- Time-to-market pressures drove most designs to make heavy use and reuse of synthesizable RTL rather than manual layout, in turn restricting the choice of available implementation solutions to fit a bus architecture into a design flow.

These trends drove the evolution of many new interconnect architectures. These include the introduction of split and retry techniques, removal of tri-state buffers and multi-phase clocks in favor of pure synchronous digital design, introduction of pipelining and various attempts to define standard communication sockets, such as VCI (Virtual Component Interface).

However, history has shown that there are conflicting trade-offs between compatibility requirements, driven by IP blocks reuse strategies, and the introduction of the necessary bus evolutions driven by technology changes: in many cases, introducing new features required many changes in the bus implementation, more importantly in the bus interfaces (e.g. the evolution from AMBA ASB to AHB2.0, then AMBA AHB-Lite, then AMBA AXI), with major impacts on IP reusability and new IP design.

Buses do not decouple the activities generally classified as transaction, transport and physical layer behaviors. This is the key reason they cannot adapt to changes in the system architecture or take advantage of the rapid advances in the silicon process technology. In addition, buses require centralized arbiter and response decoding logic, which becomes a performance bottleneck as the number of masters and slaves increases. An example of this is the Arm AHB 2.0 bus shown in Figure 9.1 (ARM 1999).

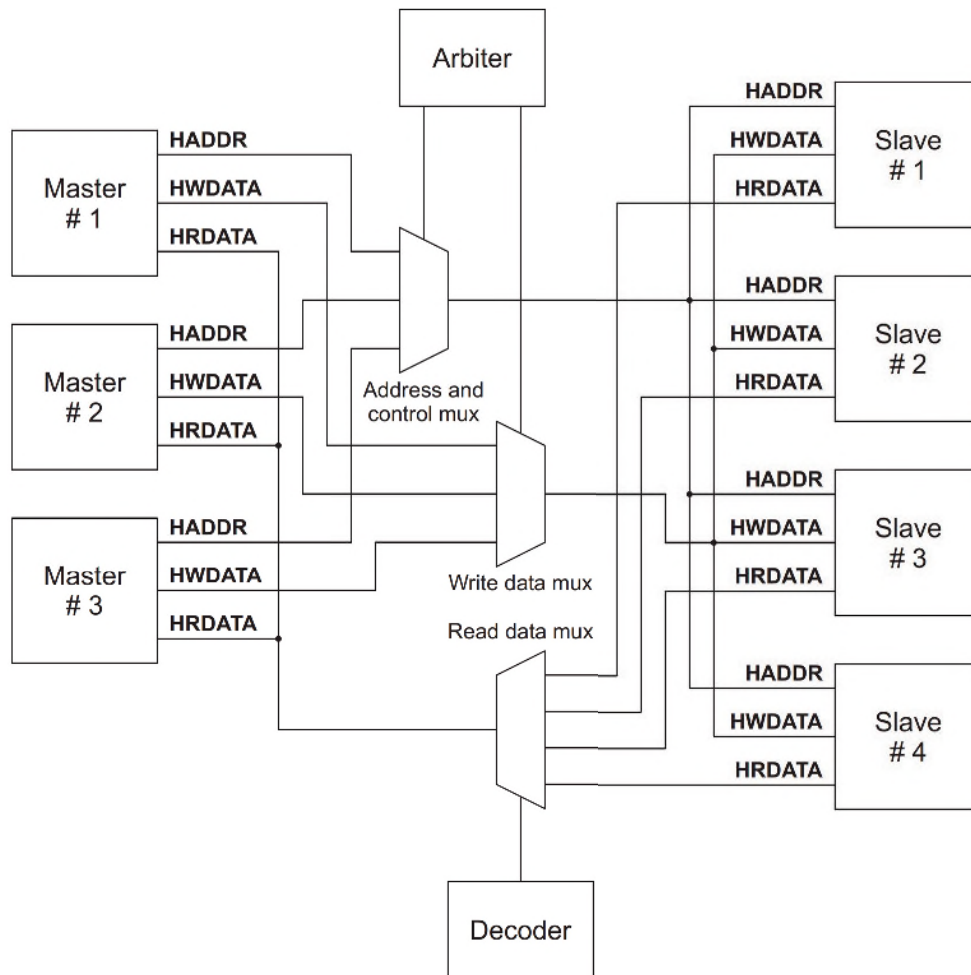


Figure 9.1. A traditional synchronous bus, in this case the implementation of an AMBA AHB bus, requires centralized arbiter and decoder logic (ARM 1999)

Consequently, changes to bus physical implementation can have serious ripple effects upon the implementations of higher-level bus behaviors. Replacing tri-state techniques with multiplexers had little effect upon the transaction levels but allowed much higher frequencies. Conversely, the introduction of flexible pipelining to ease

timing closure had area and latency effects on all bus architectures up through the transaction level.

Similarly, system architecture changes required new transaction types or transaction characteristics. New transaction types such as exclusive accesses were introduced nearly simultaneously within the OCP 2.0 and AMBA AXI standards. Out-of-order response capability is another example. Unfortunately, such evolutions typically affected the intended bus architectures down to the physical layer, if only by addition of new wires or opcodes. Thus, the bus implementations had to be redesigned.

As a result, bus architectures could not closely follow process evolution nor system architecture evolution. The bus architects always had to make compromises between the various driving forces and resist changes as much as possible.

In the data communications space, LANs and WANs successfully dealt with similar problems by using a layered architecture. By relying on the OSI model, upper- and lower-layer protocols independently evolved in response to advancing transmission technology and transaction level services. The decoupling of communication layers using the OSI model successfully drove commercial network architectures and made it possible for networks to follow very closely both physical-layer evolutions (from the Ethernet multi-master coaxial cable to twisted pairs, ADSL, fiber optics, wireless, etc.) and transaction-level evolutions (TCP, UDP, streaming voice/video data). This produced incredible flexibility at the application level (web browsing, peer-to-peer, secure web commerce, instant messaging, etc.), while maintaining upward compatibility (old-style 10 Mb/s or even 1Mb/s Ethernet devices are still commonly connected to LANs).

Following the same trends, networks started to replace buses in much smaller systems: PCI-Express is a network-on-a-board, replacing the PCI board-level bus. Replacement of SoC buses by NoCs followed the same path, driven by economics that proved the use of NoC:

- reduced SoC manufacturing cost;
- increased SoC performance;
- lower power consumption;
- reduced SoC time to market and/or non-recurring engineering expense (NRE);
- reduced SoC time to volume;
- reduced SoC design risk.

In each case, if all the other criteria are equal or better, NoCs are the better interconnect option compared to buses.

The next section describes network-on-chip architecture and explains how it improves SoC-level performance, power consumption and area.

9.1.1. NoC architecture

The advanced network-on-chip uses system-level network techniques to solve on-chip dataflow transport and management challenges (De Micheli and Benini 2017). As discussed in the previous section and shown in Figure 9.1, synchronous bus limitations lead to system segmentation and tiered or layered bus architectures.

Contrast this with the NoC approach, first commercialized by Arteris in 2006, illustrated in Figure 9.2. The NoC is a homogeneous, scalable switch fabric network.

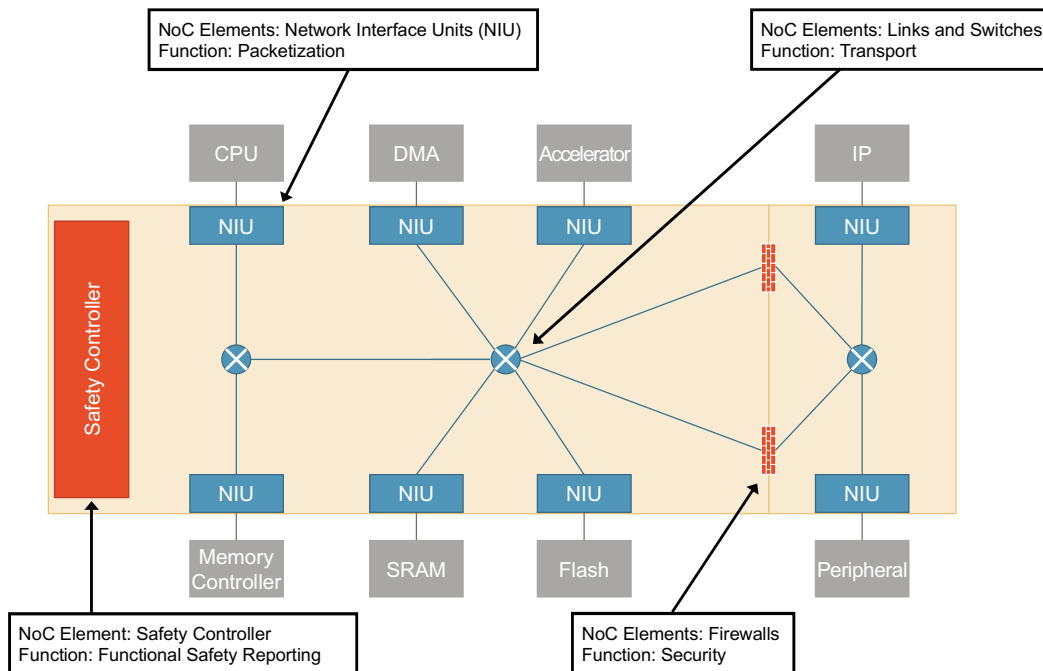


Figure 9.2. Arteris switch fabric network (Arteris IP 2020)

This switch fabric forms the core of the NoC technology and transports multi-purpose data packets within complex, IP-laden SoCs. Key characteristics of this architecture are:

- layered and scalable architecture;
- flexible and user-defined network topology;
- point-to-point connections and a globally asynchronous locally synchronous (GALS) implementation decouple the IP blocks.

9.1.1.1. NoC layers

IP blocks communicate over the NoC using a three-layered communication scheme (see Figure 9.3), referred to as the transaction, transport and physical layers.

The *transaction layer* defines the communication primitives available to interconnected IP blocks. Special NoC interface units (NIUs), located at the NoC periphery, provide transaction-layer services to IP blocks with which they are paired. This is analogous, in data communications networks, to network interface cards that source/sink information to the LAN/WAN media. The transaction layer defines how information is exchanged between NIUs to implement a particular transaction. For example, a NoC transaction is typically made of a request from a master NIU to a slave NIU, and a response from the slave to the master (Lecler and Baillieu 2011, p. 134). However, the transaction layer leaves the implementation details of the exchange to the transport and physical layer. NIUs that bridge the NoC to an external protocol (such as AMBA AXI) translate transactions between the two protocols, tracking transaction state on both sides. For compatibility with existing bus protocols, Arteris NoC implements traditional address-based load/store transactions, with their usual variants including incrementing, streaming, wrapping bursts and so forth. It also implements special transactions that allow sideband communication between IP blocks.

The *transport layer* defines rules that apply as packets are routed through the switch fabric. Very little of the information contained within the packet (typically, within the first cell of the packet, a.k.a., header cell) is needed to actually transport the packet. The packet format is very flexible and easily accommodates changes at the transaction level without impacting the transport level. For example, packets can include byte enables, parity information or user information depending on the actual application requirements, without altering packet transport or physical transport.

A single NoC typically uses a fixed packet format that matches the complete set of application requirements. However, multiple NoCs using different packet formats can be bridged together using translation units.

The transport layer may be optimized to application needs. For example, wormhole packet handling decreases latency and storage but might lead to lower system performance when crossing local throughput boundaries, while store-and-forward handling has the opposite characteristics. The Arteris architecture allows optimizations to be made locally. Wormhole routing is typically used within synchronous domains in order to minimize latency, but some amount of store-and-forward is used when crossing clock domains.

The *physical layer* defines how packets are physically transmitted over an interface, much in the same way that Ethernet defines 10 Mb/s, 1 Gb/s, etc., physical interfaces. As explained above, protocol layering allows multiple physical interface types to coexist without compromising the upper layers. Thus, NoC links between switches can be optimized with respect to bandwidth, cost, data integrity and even off-chip capabilities, without impacting the transport and transaction layers. In addition, Arteris has defined a special physical interface that allows independent

hardening of physical cores, and then connection of those cores together, regardless of each core clock speed and physical distance within the cores (within reasonable limits guaranteeing signal integrity). This enables true hierarchical physical design practices.

A summary of the mapping of the protocol layers into NoC design units is illustrated in Figure 9.3.

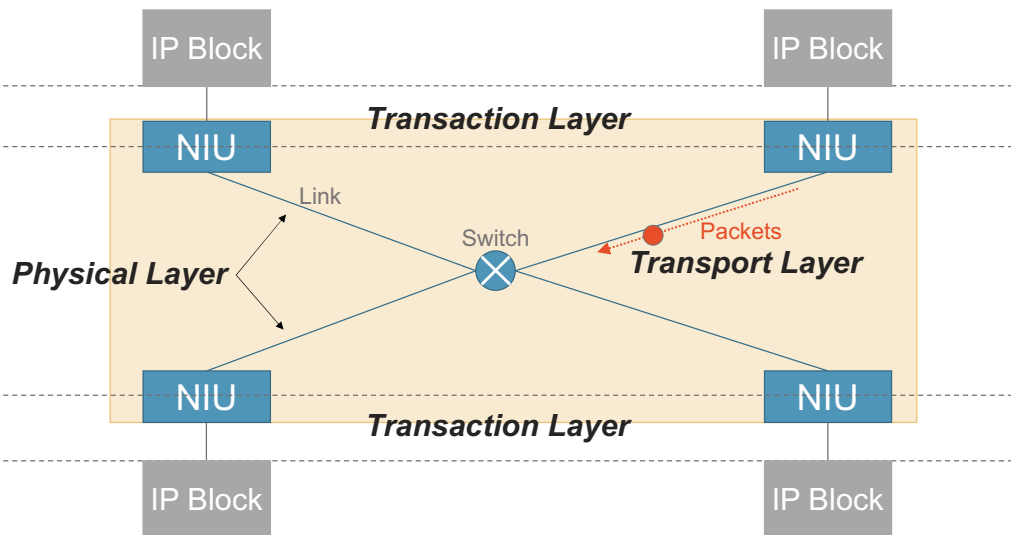


Figure 9.3. NoC layer mapping summary (Arteris IP 2020)

9.1.1.1.1. NoC layered approach benefits

The benefits of this layered approach are:

- *Separate optimizations of transaction and physical layers:* the transaction layer is mostly influenced by application requirements, while the physical layer is mostly influenced by silicon process characteristics. Thus, the layered architecture enables independent optimization on both sides. A typical physical optimization used within the NoC is the transport of various types of cells (header and data) over shared wires, thereby minimizing the number of wires and gates.

- *Scalability:* since the switch fabric deals only with packet transport, it can handle an unlimited number of simultaneous outstanding transactions (e.g. requests awaiting responses). Conversely, NIUs deal with transactions and their outstanding transaction capacity must fit the performance requirements of the IP block or subsystems that they service. However, this is a local performance adjustment in each NIU that has no influence on the setup and performance of the switch fabric. In addition, since the complex logic is typically located inside the NIUs, the elements of the transport network (switch, pipelines, etc.) are typically small and fast. This gives much flexibility to scale up or down the switch fabric to match the needed performance and layout constraints at the lowest cost.

- *Aggregate throughput*: throughput can be increased on a particular path by choosing the appropriate physical transport, up to even allocating several physical links for a logical path. Because the switch fabric does not store transaction state, aggregate throughput simply scales with the operating frequency, number and width of switches and links between them, or more generally with the switch fabric topology.
- *Quality-of-service (QoS)*: transport rules allow traffic with specific real-time requirements to be isolated from best-effort traffic. It also allows large data packets to be interrupted by higher-priority packets transparently to the transaction layer.
- *Timing convergence*: the transaction and transport layers' clocking schemes are dependent on the attached IP clocking schemes, while the physical layer has freedom to choose another clocking scheme, typically one that is best suited for distance spanning, low power and optimum routing congestion reduction. Arteris IP's NoC architecture uses a globally asynchronous, locally synchronous (GALS) approach: NoC units are implemented in traditional synchronous design style (a unit being, for example, a switch or an NIU), sets of units can either share a common clock or have independent clocks. In the latter case, special links between clock domains provide clock resynchronization at the physical layer, without impacting transport or transaction layers. This approach enables the NoC to span a SoC containing many IP blocks or groups of blocks with completely independent clock domains, reducing the timing convergence constraints during back-end physical design steps.
- *Easier verification*: layering fits naturally into a divide-and-conquer design and verification strategy. For example, major portions of the verification effort need only concern themselves with transport-level rules since most switch fabric behavior may be verified independent of transaction states. Complex, state-rich verification problems are simplified to the verification of single NIUs; the layered protocol ensures interoperability between the NIUs and transport units.
- *Customizability*: user-specific information can be easily added to packets and transported between NIUs. Custom-designed NoC units may make use of such information. For example, the design of "firewalls" that make use of pre-defined and/or user-added information to shield specific targets from unauthorized transactions. In this case, and many others, such application-specific design would only interact with the transport level and not even require the custom module designer to understand the transaction level.

9.1.1.2. NoC modularity and hierarchical architecture

A NoC packetizes transactions into transport packets and sends data from source to destination. The route that the packets take is based on the network topology. A flexible NoC technology enables designers to design the topology based on performance and floorplan requirements (Lecler and Baillieu 2011, p. 145). This allows the user to make trade-offs and optimize individual data paths to best fit design constraints.

To design a NoC that is flexible, all of its different elements or units need to be implemented in a modular fashion. This allows any of the units to be easily inserted or removed, at any arbitrary location. Some examples of NoC units include:

- *Network interface units (NIUs)*: these are the units that perform packetization and de-packetization. They are the entry and exit points of the NoC, converting between the IP's interface protocol (e.g. AXI) and the internal NoC transport packets.
- *Switches*: these are the arbiters and routers. A combination of switches makes up the NoC topology.
- *Links*: placeholders for adding potential pipeline stages.
- *Pipeline stages*: the actual pipeline elements used to break long timing paths.

As the SoC interconnect, all these NoC units are placed at the top level of the chip, connecting different IPs and routes together. As we will see in section 9.3, NoC modularity and hierarchy makes the efficient implementation of a variety of SoC-level “services” possible, such as QoS, safety, security, debug, performance analysis and perhaps some that are yet to be discovered or defined.

9.1.1.3. NoC trade-offs

In spite of the obvious advantages, a layered strategy to on-chip communication must not model itself too closely on board- or system-level data communication networks.

In data communication networks, the transport medium (e.g. optical fiber) is much more costly than the transmitter and receiver hardware and often uses “wave pipelining” (i.e. multiple symbols on the same wire in the case of fiber optics or controlled impedance wires). Inside the SoC, the relative cost and performance of wires and gates is different and wave pipelining is too difficult to control. As a result, NoCs will not, at least for the foreseeable future, be able to efficiently serialize data over single on-chip wires but must find an optimal trade-off between clock rate (100 MHz to multi-GHz) and number of data wires (16, 32, 64, etc.) for a given throughput.

Further illustrating the contrast, data communication networks tend to be focused on meeting bandwidth-related quality of service requirements, while SoC applications also focus on latency constraints.

Moreover, a direct on-chip implementation of traditional network architectures would lead to significant area and latency overheads. For example, the packet dropping and retry mechanisms that are part of TCP/IP flow control require significant data storage and complex software control. The resulting latency and area would be prohibitive for most SoCs, and largely useless as the source of errors in the transmission of packets are totally different in a SoC compared to, say, inside a data center.

Designing a NoC architecture that excels in all domains compared to buses requires a constant focus on appropriate trade-offs.

9.1.2. *Extending the bus comparison to crossbars*

In the previous section, the NoC was compared and contrasted with traditional bus structures. We pointed out that system-level throughput and latency may be improved with bus-based architectures by using pipelined crossbars or multilayer buses.

A crossbar has the advantage of connecting all initiators to all targets simultaneously without requiring centralized arbitration logic. However, because traditional crossbars still mix transaction, transport and physical layers in a way similar to traditional busses, they present only partial solutions (Hennessy and Patterson 2019, pp. F31–F32). They continue to suffer in the following areas:

- *Scalability*: to route responses, a traditional crossbar must either store some information about each outstanding transaction or add such information (typically, a return port number) to each request before it reaches the target and relies on the target to send it back. This can severely limit the number of outstanding transactions and inhibit one's ability to cascade crossbars. Conversely, Arteris' switches do not store transaction state, as packet routing information is assigned and managed by the NIUs and is invisible to the IP blocks. This results in a scalable switch fabric able to support an unlimited number of outstanding transactions.

- *IP block reusability*: traditional crossbars handle a single given protocol and do not allow mixing IP blocks with different protocol flavors, data widths or clock rates. Conversely, the Arteris transaction layer supports mixing IP blocks designed to major socket and bus standards (such as AHB, OCP and AXI), while packet-based transport allows mixing data widths and clock rate.

- *Maximum frequency, wire congestion and area*: crossbars do not isolate transaction handling from transport. Crossbar control logic is complex, data paths are heavily loaded and very wide (address, data read, data write, response, etc.) and SoC-level timing convergence is difficult to achieve. These factors limit the maximum operating frequency. Conversely, within the NoC, the packetization step leads to fewer data path wires and simpler transport logic. Together with a globally asynchronous locally synchronous implementation, the result is a smaller and less congested switch fabric running at higher frequency.

Common crossbars also lack additional services that Arteris NoC offers, such as functional safety data protection, error logging and runtime reprogrammable features.

9.1.3. *Bus, crossbar and NoC comparison summary and conclusion*

The major innovation of NoCs is that they separate the physical, transport and transaction layers, which allows the on-chip interconnect to be less tied to the attached

IP blocks and their chosen transaction protocols, and vice versa. This allows for flexible methodologies that support incremental changes to the SoC design.

As discussed in section 9.1.2, hierarchies of crossbars or multilayered buses have characteristics somewhere in between traditional buses and NoCs; however, they fall far short of the NoC with respect to performance and the ability to enable complex SoC designs.

Detailed comparison results necessarily depend on the SoC application, but with increasing SoC complexity and performance, the NoC is clearly the best IP block integration solution for high-end SoC designs today and in the foreseeable future.

9.2. NoC configurability

For a SoC design team, the NoC development or configuration overlaps the whole SoC design project lifetime: from early SoC architecture phase, to final place and route, passing through performance qualification, RTL design, IP integration, verification, debug and bring-up and software integration. We must pay special attention so that a decision made during an early phase, such as design architecture, does not jeopardize the feasibility of a later phase, such as physical layout. Moreover, it is not unusual that marketing requirements for a given SoC change while the chip is being designed, adding or removing some IPs, and thus impacting the NoC specification. As the complexity and cost of brand new architecture increases, organizing projects around reusable platforms becomes more efficient. Small variations of the platform or derivatives may then be brought to the market in shorter cycles. The interconnect system design methodology must make sure that those late specification changes are smoothly integrated in the design and verified.

9.2.1. Human-guided design flow

The NoC user's primary challenge is to navigate between the flexibility required to only pay for useful resources required without incurring the cost of overengineering the interconnect. A deliberate human-guided design flow is required to do this.

- In a first phase of NoC design flow, we define the *functional specification* of the NoC: what are the different sockets? The transaction protocols? Subsets of the protocols used? What is the memory map? The objective is to have clear statements about the possible transactions that initiators can request and targets can accept. Remember, some initiators may emit unaligned or wrapping transactions towards targets which do not support them. This implies conversions, and it is important to know whether that underlying logic must be implemented or not. The specification phase leaves completely aside how the functionality is implemented.

– The *performance specification* is the definition of the quality-of-service (QoS) expected from the NoC. We should note that performance of an interconnect is not in any way determined by its internal resources (number of pipeline stages, number of possible pending transactions, etc.), but by the fact that a system using this NoC either behaves correctly or not: will a displayed video image freeze, the loud speaker click or the embedded web-browser be too slow? At this stage, we only define the objectives. Again, the actual implementation is left for a further step. This phase is also a good time to collect data on the actual behavior of the IP blocks around the NoC: what are the transaction patterns? How do they react to transient back pressure? To latency? This behavior of the system around the interconnect is defined in *scenarios* and examined through performance simulation that is built into the NoC configuration tooling.

– In the *architecture* phase, we now allocate hardware resources in the NoC, defining how transaction flows are merged and scattered, tuning wire bundle serialization, buffer size, arbitration schemes, pipeline stages location, clock and power crossings location, etc. The architecture may be simulated against the previously defined performance-oriented scenarios in order to determine, on the one hand, if the scenario performance criteria are met, and on the other hand, what the saturated or under-used resources in the NoC are, and make the corresponding adjustments to the NoC configuration.

– In the *NoC structural* phase, the NoC is translated into hardware blocks. The translation is automated, yet some minor but crucial details may be fine-tuned, such as the presence and/or reset values of configuration registers. At the end of this phase, the register transfer level (RTL) description of the NoC can be exported together with synthesis scripts, ready to be integrated, simulated and synthesized.

– In the *verification* phase, the RTL description is checked against its functional and performance specifications. For the functional verification, a test bench and stimuli are generated automatically out of the specification. The test bench uses third-party protocol checkers and ensures that transactions are correctly translated. Protocol coverage is measured to ensure the stimuli have stressed the interconnect in a number of predefined corner cases. For performance verification, the scenarios are played against the RTL description of the NoC to verify that their performance criteria are met.

9.2.2. *Physical placement awareness and NoC architecture design*

Designing the NoC architecture is one of the most critical steps in achieving a feasible design. At 16 nm and smaller process generations, physical design effects intruded on the SoC architecture phase. It became quite possible to design a valid SoC architecture that was not implementable during the place and route phase of the design. Therefore, modern NoCs have to take into account physical constraints during the architecture and RTL development phases at 16 nm process generations and below. A NoC topology needs to take into account floorplan constraints in order to verify that

timing can be closed given the proposed architecture. The timing closure then needs to be verified using physical synthesis prior to turning over the design to the place and route team. Figure 9.4 contrasts two different NoC topologies – the diagram on the left has a NoC that is designed without taking the floorplan into consideration, while the diagram on the right shows a NoC that has a floorplan-friendly topology. The gray rectangles with the letters “SW” represent a switch in the NoC.

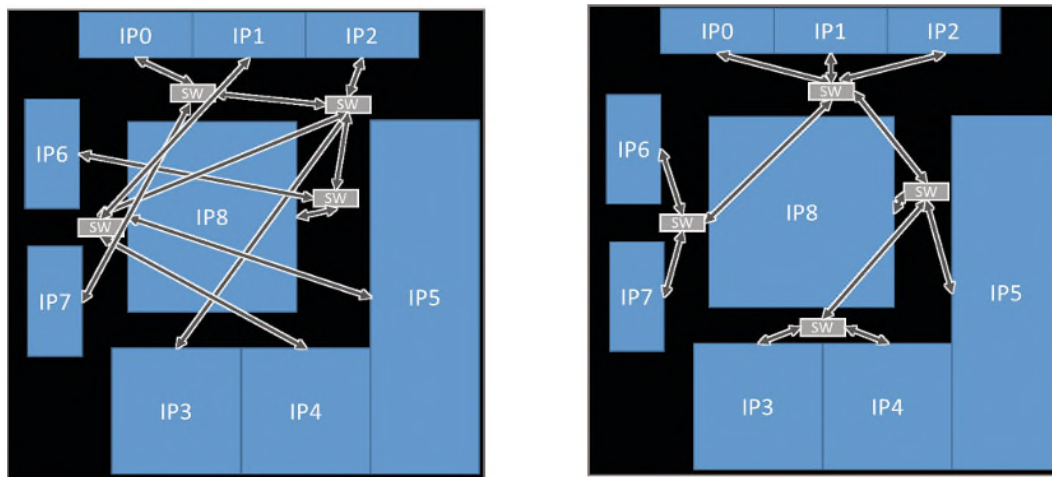


Figure 9.4. *The NoC on the left has a floorplan-unfriendly topology, whereas the NoC on the right has a topology that is floorplan-friendly (Arteris IP 2020)*

The second part of the NoC architecture design is to place pipeline stages at proper locations to allow the NoC paths to span distances. Given a clock period and transport delay of the process node, it is possible to calculate the number of pipeline stages required to span a particular distance.

In the example presented in Figure 9.5:

- if a path is running at a 600 MHz clock frequency, the cycle time is 1.67 ns, of which 1.42 ns is usable;
- in the TSMC 28HPM process, transport delay is approximately 0.644 ns/mm;
- therefore, the pipeline-to-pipeline maximum distance is $1.42/0.644 = 2.2$ mm;
- for a path that is 7 mm in length, at least three pipeline stages are required to span the distance, as illustrated in Figure 9.5.

This calculation needs to be done for every path in the NoC and have pipeline stages enabled accordingly in order for it to close timing. It is very common at this point to:

- fail to enable certain critical pipeline stages resulting in the need for more timing debug and synthesis, place and route (SP&R) iterations;

– over-provision of the number of pipeline stages in anticipation of reducing the number of SP&R runs.

The problem of misconfiguring pipeline stages results in more back-end iterations, and increased latency, and area – therefore power consumption – of the NoC. As a result, automating the process of adding pipeline stages is becoming essential to help address this problem in a systematic manner. This is a prime area of research for future NoC technology development.



Figure 9.5. Pipeline stages are required in a path to span a particular distance given the clock period and transport delay (Arteris IP 2020)

9.3. System-level services

As previously discussed, the modular and hierarchical nature of a NoC allows the creation of system-level “services” that provide value to the SoC designer, the chip end-user or both.

9.3.1. Quality-of-service (QoS) and arbitration

Sophisticated QoS and arbitration schemes are system-level services that are made possible thanks to the modular nature of NoC elements. Any type of communications within the NoC can be prioritized at different levels, and these priorities can be changed dynamically at runtime either automatically by the NoC itself (based on information obtained from other NoC services described in section 9.3.2) or at runtime through software interaction with programmable NoC elements using control and status registers (CSRs). Switches and other elements within the NoC can then order the packet communications passing through them using the priorities associated with the packets (Hennessy and Patterson 2019, pp. F21–F23).

Multiple types of NoC hardware mechanisms can be used to implement the optimal ordering within the NoC, with “optimal” being defined as the proper trade-off between bandwidth, latency, area and power consumption required to meet the desired use cases. One example of an ordering mechanism is hardware logic that ensures that higher-priority traffic is prioritized through the NoC before lesser-priority data to ensure system-level latency requirements are met. An example technology to do this is credit-based flow control mechanisms that can ensure prioritized use of links between NoC elements. However, there are other QoS technologies that can perform this task with less area and buffering. The key is to use the proper technology that best helps meet the SoC-level performance, area and power consumption goals.

9.3.2. *Hardware debug and performance analysis*

Because a NoC “sees” most of the data communicated within a SoC, it is useful to be able to inspect the data traveling through it not only to debug SoC-level functionality but also to examine the on-chip dataflow to ensure the SoC is meeting performance requirements. To do this, a NoC must offer the designer different types of probes that can be configured to intercept data-in-transit and provide useful information back to the designer (during development) or system (during runtime). The types of information that can be determined based on this NoC data include errors, either internal to the NoC or created by attached IP blocks, and performance metrics, such as bandwidth and latency. The hardware probes can act like CPU performance counters, filtering on packet or transaction parameters and generating performance histograms and event statistics, or traces, which can be output to industry standard debug infrastructures, like AMBA CoreSight, or to internal memories for use at runtime.

The usefulness of these hardware mechanisms is enhanced when similar capabilities are implemented in the SystemC performance models generated by the NoC configuration software, not just the NoC RTL that becomes part of the SoC. In this way, the SoC architecture and design teams have better correlation between simulated and actual functionality and performance, which results in a more predictable chip behavior and design schedule.

9.3.3. *Functional safety and security*

9.3.3.1. *NoC functionality, failure modes and safety mechanisms*

Safety at the SoC level has become important in life-critical industries such as the automotive industry, where the ISO 26262 specification provides guidance related to functional safety implementation and processes for automotive electronics. Within a SoC, NoC functional safety mechanisms are used to detect, mitigate and correct faults and failures caused by random errors when a system is in operation.

Single-event effects (SEE) – electrical disturbances caused by cosmic rays and the ionization energy emitted when they interact with semiconductors – are the cause of random errors. These random errors can have transient (temporary) or permanent effects. Examples of random, transient effects – also known as “soft errors” – include single-bit upsets (SBU), such as “bit flips” in memory cells or logic flops, and single-event transients (SET), which are voltage glitches that may or may not cause an error. There are also instances where these can occur simultaneously, resulting in multiple bit upsets (MBU). “Hard errors” caused by SEEs resulting in permanent damage include single-event latch-up (SEL), single-event burnout (SEB) and single-event gate rupture (SEGR) (JESD86A 2006, pp. 3–4).

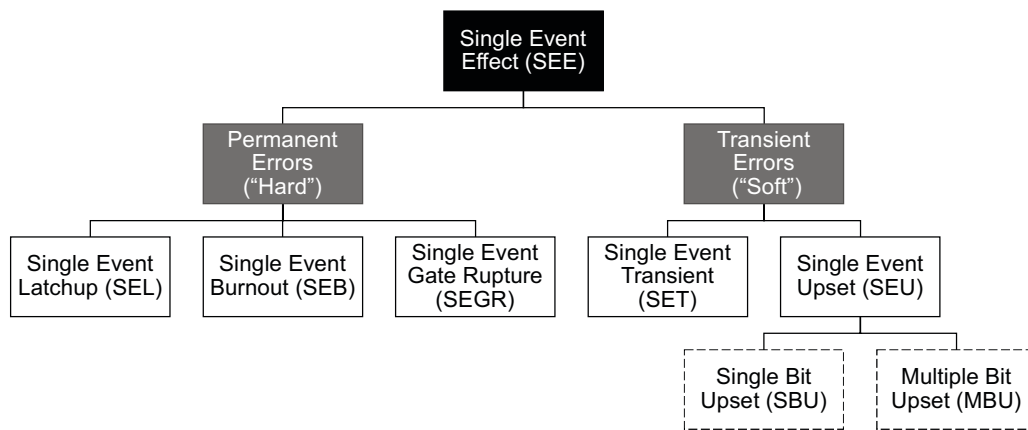


Figure 9.6. Single-event effect (SEE) error hierarchy diagram
(ISO26262-11 2018b; JESD86A 2006)

Because the cause of these errors is natural physical phenomena and manufacturing defects, they will occur randomly, so it is important to detect and mitigate their effects to achieve and maintain system safety. This is especially important for the NoC interconnect, which sees most of the dataflow on the chip and can therefore detect and perhaps mitigate errors that occur within the NoC or even within the IP blocks attached to it. Today’s state-of-the-art NoCs have safety-specific technical features within their products to do this. Below are examples of these features, also known as functional safety mechanisms:

- adding and checking parity or ECC bits added to on-chip communications traffic;
- duplicating logic and comparing results;
- triple-mode redundancy (TMR) or majority voting;
- communication time-outs;
- hardware checkers that verify correctness of operation;

- safety controller to gather error messages from throughout the system, make sense of them and communicate higher up the system;
- built-in self-test (BIST) for all functional safety mechanisms to detect latent faults.

The state-of-the-art NoC interconnect implements these features to provide data protection for automotive functional safety as well as reliability, enabling SoCs built with them to qualify for up to ISO 26262 ASIL D certification.

9.3.3.1.1. NoC functional safety analysis and FMEDA automation

Completing failure mode effects analysis (FMEA) and failure mode effects and diagnostic analysis (FMEDA) is particularly difficult for interconnect IP. First, because they are safety elements out of context (SEooC, as are all IP and generally chips), they therefore have to be validated against agreed assumptions of use (AoU). Also, interconnect IP is highly configurable, which makes reaching agreement on the assumptions of use, defining potential failure modes and effects and validating safety mechanisms even more challenging than it is for other IP, as safety assurance must be determined on the configuration built by the integrator/commercial interconnect IP user.

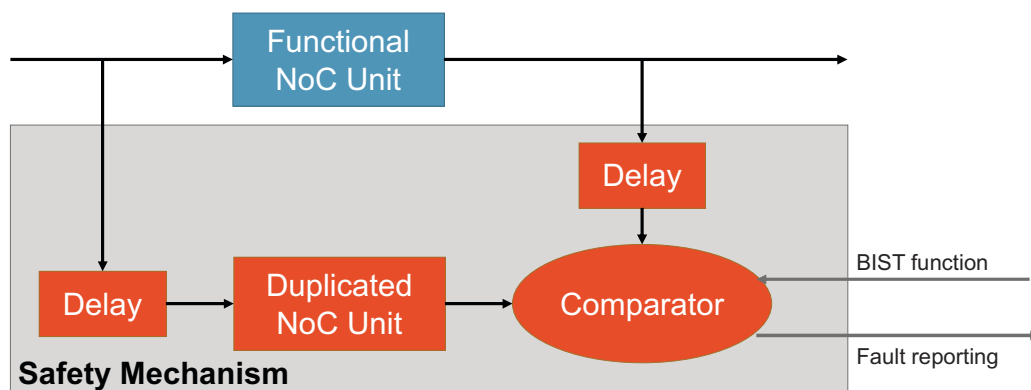


Figure 9.7. Failure mode effects and diagnostic analysis (FMEDA) includes analysis of safety mechanisms, such as hardware unit duplication and a built-in self-test (BIST) (Arteris IP 2020)

A state-of-the-art way to handle this complexity is to start with a qualitative, bottoms-up FMEA analysis, which can then guide the creation of a configuration-specific FMEDA (ISO26262-10 2018a, pp. 75–76). This requires modular NoC components that can be configured in specific hierarchies with safety mechanisms tied to those components, so that as you configure the IP, those safety mechanisms are automatically implemented in a manner that ensures safety for the overall function. For example, in the Arteris IP NoCs, the network interface unit (NIU) responsible

for packetization can be duplicated (Figure 9.7). Then in operation, results from these units are continually compared; variances which would signal a failure are searched for. NoCs can also implement ECC or parity checks in the transport. Following the FMEA/FMEDA philosophy, these safety mechanisms are designed to counter the various potential failure modes identified by the IP vendor. And because of the modularity, functional safety diagnostic coverage of the entire NoC and the NoC-level failure mode distribution (FMD) can be calculated based on the concatenation of individual module coverage metrics.

The NoC IP should also provide BIST functions to check the logic for safety mechanisms at reset and during runtime. All of this safety status rolls up into a safety controller provided as part of the NoC, which can be monitored at the system level.

9.3.3.2. *NoC contributions to system-level security*

Security is often portrayed as a software- or system-level issue with most engineers' perception of "hardware security" focused on dedicated security IP blocks such as trusted boot controllers and security protocol or cryptographic accelerators. However, as in the enterprise computing world where a major component of "IT security" is network security, ensuring SoC-level security requires hardware security features in the network-on-chip interconnect. The most important NoC security capability is the firewall, similar to an IT equipment firewall, which is programmed by chip designers to allow certain data to pass when it is expected in a use case, and blocks or poisons data when it should not be in transit. Firewalls protect parts of the SoC from security intrusions that have found their way into the system through an attack surface, such as a memory (buffer overflow error), intentional power glitch or other man-made phenomena.

NoC firewalls filter traffic at runtime, only allowing traffic to transit through the firewall if it passes a chip designer-defined set of tests. These tests within firewalls can be programmed based on a set of use cases that are expected at runtime and are implemented as part of a system-level hardware and software security scheme. In addition to detecting the progression of data that is determined to be unsecure, there is flexibility in how the firewall reports the transgression, tags the data and other actions. One possible mechanism is that unallowed packets are "poisoned", indicating to logic blocks that receive the data that the data may be corrupted or otherwise suspect. This allows the SoC security architect to determine and limit what responses a hacker can glean from the system, making the hacker's job harder.

9.4. Hardware cache coherence

As the number and types of processing elements within modern SoCs have increased, it has become increasingly important for hardware to help manage the

dataflow created by these IP blocks. Managing data access for multiple processors within a SoC is complicated by the sheer number of cores as well as the varying traffic patterns of dissimilar, heterogeneous cores. Implementing cache coherence at the SoC-level in hardware is the best way to create scalable coherency while enabling simpler software. However, because each coherent SoC architecture is different, the means to implement SoC-level cache coherence must be highly configurable. For these reasons, NoC technology has become the state of the art for multi-processor cache coherent systems.

NoC technology is ideally suited to implementing hardware cache coherence for many reasons. First, NoCs are highly configurable and can adapt to different transaction protocols, power domains and functional safety requirements. As explained in section 9.1.1, NoC architectures separate the transaction from the transport and physical mechanisms of the interconnect. This separation allows novel cache coherence features that would be extremely difficult to do in a technology other than NoC, for example, integrating IP using multiple types of protocols, such as AMBA ACE and CHI, in the same cache coherent system. Second, the hierarchical and modular nature of the NoC technology allows for the strategic addition of caches within a system to allow new capabilities. An example of this is a proxy cache, which allows multiple non-coherent IP blocks, such as artificial intelligence (AI) accelerators using the AXI protocol, to be connected into clusters that participate as first-class citizens within the cache coherent domain. Another benefit of NoCs in implementing cache coherence is that their modularity allows each NoC element to be placed within a chip floorplan to optimize for the desired mix of performance, power consumption and die area. This is in stark contrast to older bus- and crossbar-based “cache controller” IP, which had to be implemented as a monolithic IP block, often causing SoC routing congestion, timing closure and performance issues.

9.4.1. NoC protocols, semantics and messaging

NoC interconnects usually implement cache coherency with a directory protocol instead of a broadcast protocol. This allows for more point-to-point communications through the network, which lowers the bandwidth required as the number of coherent agents scales up. NoC interconnects can have multiple directories which track the location and state of shared data (cache lines) within the coherent domain (Figure 9.8).

Although a cache coherent NoC manages dataflow similar to any other NoC, the message types and sizes being communicated through the NoC differ in their nature from “traditional” NoCs. In addition to read-and-write messages and data common to protocols such as AXI and OCP, a cache coherent NoC carries messages implementing a protocol that describes the states of data that can be located within a set of memory locations within the system, including IP block caches, NoC caches

and off-SoC memory. To do this, a cache coherent NoC has a native internal protocol that allows it to carry the semantics of the traffic of attached IP, which are using their own protocols. To accommodate multiple industry-standard protocols such as AMBA ACE and CHI, a NoC must have an underlying native protocol that is a “superset” protocol, implementing all of the required semantics for both protocols. The flexibility of the NoC technology allows for the communications of nearly any type of semantics, whether load/store, cache coherence, message passing or others not yet invented.

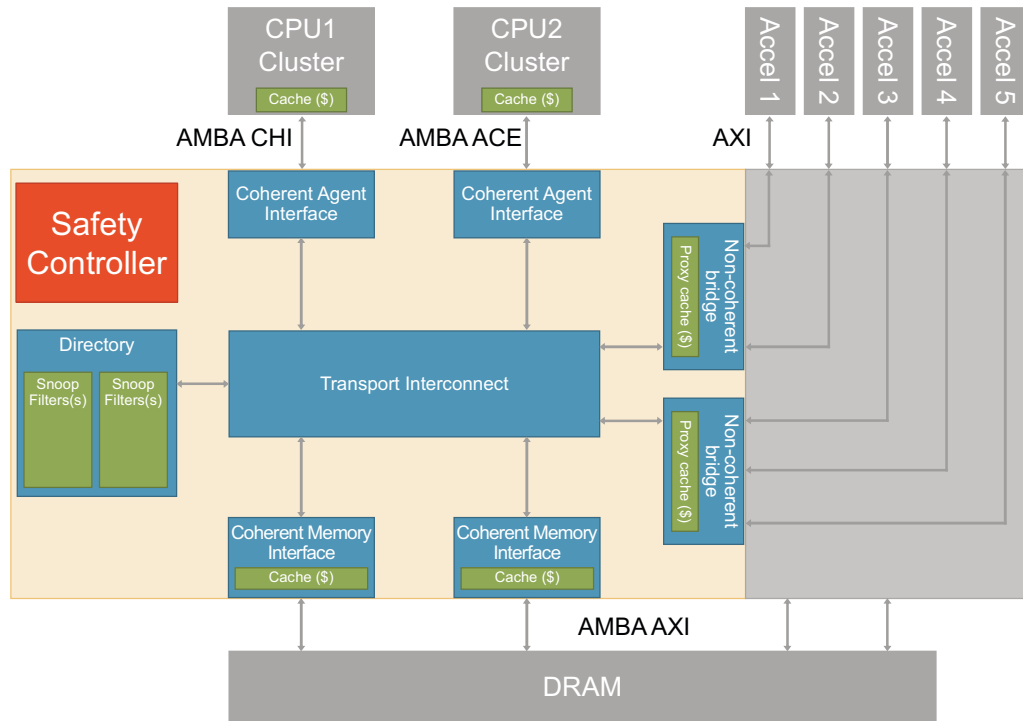


Figure 9.8. A cache coherent NoC interconnect allows the integration of IP using multiple protocols (Arteris IP 2020)

9.5. Future NoC technology developments

Network-on-chip has established itself as the foundation of nearly all SoCs having any semblance of complexity in size, semantics or power consumption requirements and will be the underlying in-silicon communications backbone for chips for years to come. However, time does not stand still and there are multiple avenues for the NoC technology innovation on the horizon. Here are a few.

9.5.1. Topology synthesis and floorplan awareness

As SoCs become larger and more complex with hundreds of initiators and targets, highly heterogeneous traffic patterns and chip floorplan sizes surpassing 600 square

multimeters in today's sub-16 nm processes, it is becoming more difficult for chip architects to understand and analyze all the options required to configure the on-chip interconnect. In most use cases, restricting the alternatives to regular topologies such as meshes and rings to simplify the analysis results in suboptimal performance, power and area results. One solution is to provide more automation at earlier stages in the NoC configuration design flow by using more system-level information earlier in the flow to make decisions. In the future, NoC configuration tooling will import user-defined and production (LEF/DEF format) floorplans for the entire SoC; automatically generate a NoC topology that adapts to the floorplan while meeting performance, power and area metrics; and automatically configure pipelines in NoC logic and wire links to meet timing closure.

This level of automation requires innovations in the domains of graph theory, linear optimization and other mathematical constructs. Intelligent algorithmic implementation will be required to ensure configuration tooling runtimes are acceptable. The result of this innovation will be the capability to create even more complex chips than today's, in less development time while achieving an equal or higher quality of results.

9.5.2. Advanced resilience and functional safety for autonomous vehicles

As automobiles and other vehicles become more autonomous, the brains of these systems are being required to function in the presence of errors, faults and failures. In the current edition of ISO 26262, a system failure that affects safety can lead to a fallback fail-safe state where a driver assumes control of the vehicle. As autonomous driving technologies progress, this fallback process is being supplanted by the requirement for fail-operational, where the system remains available and functions despite faults. Implementing SoCs capable of fail-operational has at least two opportunities for technical innovation.

First, SoC architectures will add even more redundancy and checking to reduce common cause failures and increase fault tolerance. Many of the techniques used in aviation, space, military and medical applications will be adapted for on-chip implementation in these SoCs. Examples of techniques include triple-modular redundancy (TMR, also called 2-out-of-3 voting), increased diversity of processing and logic elements, and the addition of even more diagnostics and runtime testing. Second, the nature of diagnostics will change from *detecting* when something has failed to *predicting* when something might fail, before the failure occurs. An example of this is in-silicon sensors that detect the rate of transistor aging.

In all the cases above, the NoC interconnect will play a hugely important role because it sees all the data on the SoC and is physically implemented throughout the floorplan. This makes the NoC an ideal target for implementing these new functional

safety mechanisms and diagnostics as well as integrating the information from them for on-chip analysis and action.

9.5.3. *Alternatives to von Neumann architectures for SoCs*

Most SoC architectures have evolved as instantiations of the von Neumann architecture where CPU functionality connects to a single memory hierarchy and a set of I/O devices. As Moore's law has slowed down and SoC frequencies have leveled off, SoC architectures have changed to adapt to this new reality and obtain better performance than was previously possible. Two innovation paths where the NoC technology plays a critical role are in parallel processing architectures and hierarchical cache coherence. Both allow more processing to occur simultaneously while keeping more of the data being acted upon within the SoC in local memories, rather than needing to be sent off-die and succumbing to the latency, power consumption and bandwidth penalties of off-die memory accesses.

9.5.3.1. *Massively parallel AI/machine learning architectures*

Artificial intelligence and machine learning applications have spearheaded a long-anticipated revolution in massively parallel computing architectures. Rather than being implemented solely in exotic supercomputers, these architectures are being created for mass-market chips that reside in data center servers, vehicles and consumer electronics. The current state of the art is related to hardware implementation of various flavors of neural networks, but this could change in the future. What will not change is the need to have multiple processing units (which today are usually the same but are becoming heterogeneous for greater efficiency) connected within a SoC with multiple memories.

Because of the ease of analysis and implementation, many of today's AI chips are organized using a NoC mesh or ring topology connecting tens or hundreds of a single type of processing element. Each processing element is a subsystem containing logic and memories, often connected together by its own internal NoC. As the number of processing elements increases, chip designers often choose to create hard macro tiles of synthesized parts of the mesh structure (Figure 9.9). These tiles are then connected at the top level to create an even larger NoC topology. As chips scale, being able to easily create and close timing of replicable tiles and hard macros using a NoC will become extremely important. Technical innovation opportunities will arise in how to implement internal NoC addressing methods to efficiently enable this.

Another area of technical effort for these types of chips is in NoC transaction semantics, to be extended beyond the traditional reads and writes. For example, an efficient update of neural network weights to multiple processing elements requires the NoC to broadcast write data while meeting latency goals and not consuming egregious amounts of NoC bandwidth. Conversely, a NoC able to simultaneously broadcast read

operations to multiple targets and aggregate the results using mathematical functions such as addition, min and max will offer great performance benefits in multiple applications (Jerger *et al.* 2017, pp. 148–149).

In addition to NoC tiling and semantics, AI chip architectures are evolving to use different types of processing elements (heterogeneous) which need to be connected in more complex and irregular topologies than simple meshes and rings. The topology synthesis and floorplan awareness technologies described earlier in section 9.5.1 will help accomplish this.

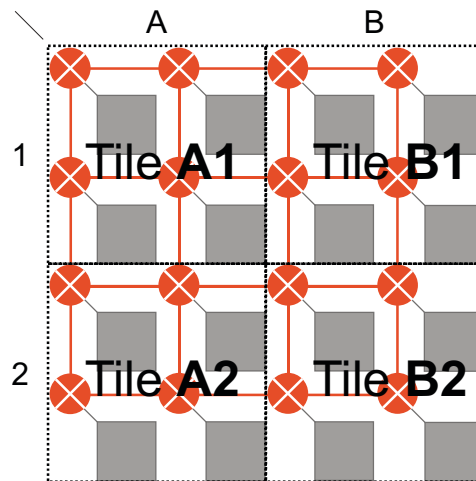


Figure 9.9. NoC interconnects enable easier creation of hard macro tiles that can be connected at the top level for scalability and flexibility (Arteris IP 2020)

9.5.3.2. Hierarchical cache coherence

Another way to work around the off-die memory bottlenecks of the typical von Neumann architecture is again to provide more in-SoC memory than traditional designs, as well as to do so in a way that is more software-friendly than the massively parallel architectures with explicit data exchange between units, described above. Hierarchical cache coherence will become mainstream over the coming decade to solve both these problems. In this scheme, cache coherent subsystems are created containing multiple processing elements and a subsystem-specific directory and cache, and many of these subsystems are then connected together in another cache coherent subsystem with its own directory and caches (Sorin *et al.* 2011, pp. 181–184). Although any of the caches within the overall system can be snooped by any coherent agent, architects will want to design the set of coherent subsystems so that data locality within subsystems is optimized.

Hierarchical coherency will benefit from the flexibility and configurability of the NoC technology, allowing support for heterogeneous processing elements and

coherency clusters. Also, the hierarchical coherency subsystems can be synthesized into hard macros and tiles, as described in section 9.5.3.1, thereby enabling massive scalability of on-die cache coherent systems.

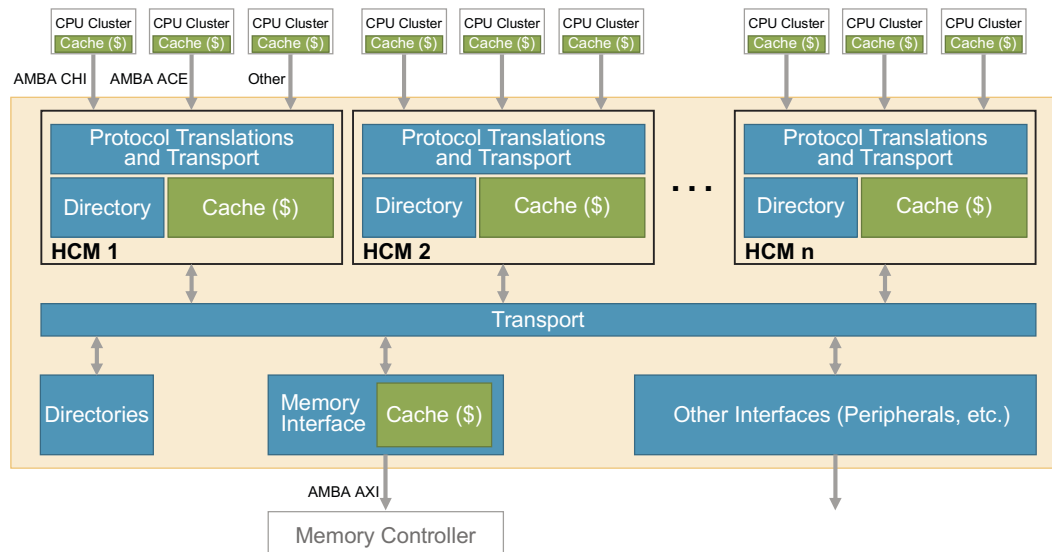


Figure 9.10. Hierarchical coherency macros enable massive scalability of cache coherent system (Arteris IP 2020)

Figure 9.10 shows an example of how this will be implemented within the cache coherent NoC. The hierarchical coherency subsystems are called hierarchical coherency macros (HCM) in the diagram. These can be the same or have heterogeneous contents, and each HCM contains its own directory with snoop filter(s) and cache. The multiple HCMs are connected within a separate cache coherent subsystem that ties them all together into one cache coherent NoC.

9.5.4. Chiplets and multi-die NoC connectivity

As massively parallel architectures using tiling and hierarchical cache coherence are developed, another need becomes obvious: how do we expand these systems beyond the confines of a single die? The NoC technology will play a critical role in making this possible.

Today's state-of-the-art multi-die and chiplet systems are being developed using somewhat *ad hoc* or bespoke connectivity solutions. This will soon change as inter-die connectivity standards evolve and stabilize. For the development of companion accelerator SoCs for server chips such as the Intel Xeon family, compute express link (CXL) is emerging as a first-generation chip-to-chip standard. The CXL

standard allows one-way coherency with the x86 or other CPU chip acting as a master with CXL-attached accelerator chips being slaves. Another standard, called cache coherent interconnect for accelerators (CCIX), allows all the dies or chips connected using the interface to share a common view of memory and be able to snoop each other, therefore creating a multi-die cache coherent system. CCIX is expected to become a standard for the creation of chiplet SoCs, which contain multiple dies within the same chip package acting together as one larger chip.

The configurability and flexibility of the NoC interconnect technology makes it ideal for use in chips using CXL, CCIX or other die-to-die communication standards. One issue with these solutions, though, is the requirement to use a PHY as the underlying physical and transport mechanism. Both CXL and CCIX use the PCIe PHY as the physical layer connection mechanism with changes to parts of the upper PCIe layers (link, transaction and protocol) to implement the new standards. All-digital connectivity technology will emerge for chiplet/die-to-die connectivity within a chip package where long distances will not need to be traversed. This will be underpinned by a foundation of high-bandwidth NoCs within each chiplet, physically connected by through-silicon via (TSV), silicon-interposer and other 3D chip integration technologies. Using chiplet architectures while minimizing power consumption and maintaining on-die-level bandwidth and latency will require all-digital interconnects using the NoC technology.

9.5.5. Runtime software automation

One of the challenges of large-scale SoC development is keeping the associated runtime software in sync with the on-chip peripheral and processor hardware capabilities and versions. A NoC interconnect implementation has much of the information that is required to create the foundation of operating system device drivers and hardware abstraction layers (HAL), including items such as memory maps, register maps, peripheral access and command capabilities and mask and interrupt information. Information from the NoC configuration tooling, for instance, exported using the IP-XACT format, could be used to populate header file information for software device drivers and hardware abstraction layer components for industry-standard operating systems such as Linux (with udev) and AUTOSAR (with its Microcontroller Abstraction Layer, or MCAL, shown in Figure 9.11).

The benefits of using information from the NoC configuration to populate these parts of the software infrastructure are that it ensures that the software stays in sync with the implemented hardware capabilities (version control), reduces the chance of errors due to retyping copious amounts of information (systematic errors) and automates traceability for compliance to functional safety standards such as ISO 26262 for automotive and IEC 61508 for general electronics systems.

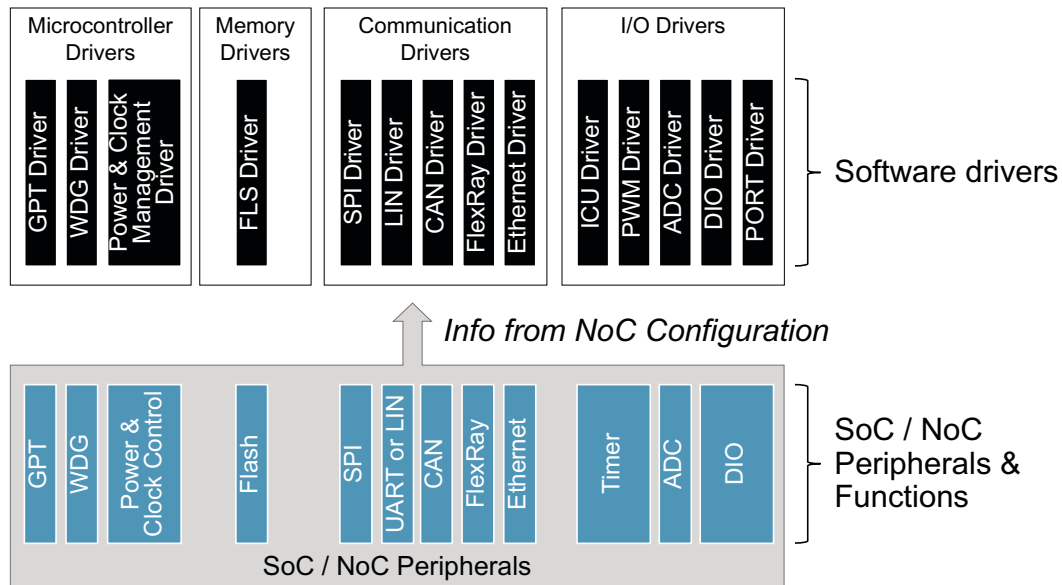


Figure 9.11. An AUTOSAR MCAL showing how NoC configuration information will be used as a “golden source” for software driver header files. Adapted from Renesas (2020)

9.5.6. Instrumentation, diagnostics and analytics for performance, safety and security

As we have seen in the previous sections, the NoC interconnect has a “God’s eye” view of data flowing through a chip, and we have already implemented the technical means to see this data in real time using hardware probes and performance counters (see section 9.3.2), data checkers for functional safety (see section 9.3.3.1) and firewalls for security (see section 9.3.3.2). What if we could coordinate all this data together and analyze it for patterns that can detect or predict problems in performance and QoS, safety and security? What if we could use these inferences to then change settings within the NoC to alleviate issues, without requiring coordination with the system-level software? And what if we could give information to the system-level software about what is going on within the SoC dataflow so that the software can make more intelligent decisions for better use of chip resources?

This is one of next frontiers of NoC technology. These technologies, once combined in a useful format that is similar in concept to Waze or Google’s live traffic maps, can be used to provide information to regulate the dataflow throughout the NoC, including links to the system level in order to control and synchronize the behavior of IP blocks transmitting data onto the NoC. As the number and types of initiators and targets attached to a NoC increase, we create a highly complex system that becomes

impossible to understand without looking at it in its entirety. Integrating advanced NoC instrumentation and analytics at the system level will provide this necessary capability.

9.6. Summary and conclusion

The purpose of this chapter has been to explain how multi-processor SoCs have been enabled by NoC technology, first by showing the history and evolution of NoCs and the current state of the art, then by describing how the technology will progress as SoC architecture requirements change. Networks-on-chip play the critical role in on-chip dataflow, which affects the achievable bandwidth and latency of a chip. Managing this data traffic at the system level becomes the most important SoC engineering issue as chips scale in size and the numbers and types of processing elements increase. In the past, individual blocks of IP, such as processors and hardware accelerators, dominated the time and imaginations of engineers creating the latest generations of SoCs. Soon, the NoC interconnect will be widely accepted as most important IP required to create these large chips.

9.7. References

- ARM (1999). AMBA specification (Rev 2.0) [Online]. Available: https://static.docs.arm.com/ihi0011/a/IHI0011A_AMBA_SPEC.pdf.
- Arteris IP (2020). Arteris IP – The Network-on-Chip (NOC) interconnect company [Online]. Available: <http://www.arteris.com/>.
- Benini, L. and De Micheli, G. (2002). Networks on chips: A new SoC paradigm. *Computer*, 35(1), 70–78.
- De Micheli, G. and Benini, L. (2017). Networks on chips: 15 years later. *Computer*, 50(5), 10–11.
- Hennessy, J.L. and Patterson, D.A. (2019). Interconnection networks. *Book companion – Computer Architecture: A Quantitative Approach*, 6th edition. Morgan Kaufmann Publishers Inc., F1–F118.
- ISO26262-10 (2018). Road vehicles – Functional safety – Part 10: Guidelines on ISO 26262, ISO standard 26262-10 ISO26262-10:2018, ISO – International Organization for Standardization [Online]. Available: <https://www.iso.org/standard/68392.html>.
- ISO26262-11 (2018). Road vehicles – Functional safety – Part 11: Guidelines on application of ISO 26262 to semiconductors, ISO standard 26262-11 ISO26262-11:2018, ISO – International Organization for Standardization [Online]. Available: <https://www.iso.org/standard/69604.html>.
- Jerger, M.E., Krishna, T., and Peh, L.-S. (2017). *On-Chip Networks: Second Edition. Synthesis Lectures on Computer Architecture*. Morgan & Claypool Publishers.

- JESD86A (2006). Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices, Standard No. 89A JESD89A, JEDEC [Online]. Available: <https://www.jedec.org/standards-documents/docs/jesd-89a>.
- Leclerc, J.-J. and Baillieu, G. (2011). Application driven network-on-chip architecture exploration & refinement for a complex soc. *Des. Autom. Embedded Syst.*, 15(2), 133–158. Available: <https://doi.org/10.1007/s10617-011-9075-5>.
- Renesas (2020). Microcontroller abstraction layer (MCAL) [Online]. Available: <https://www.renesas.com/us/en/solutions/automotive/technology/autosar/autosar-mcal.html>.
- Sorin, D., Hill, M.D., and Wood, D.A. (2011). *A Primer on Memory Consistency and Cache Coherence. Synthesis Lectures on Computer Architecture*. Morgan & Claypool Publishers.