Chapter 1
Introduction to Network-on-Chip Design

Computing technology affects every aspect of our modern society and is a major catalyst for innovation across different sectors. Semiconductor technology and computer architecture has provided the necessary infrastructure on top of which every computer system has been developed offering high performance for computationally-intensive applications and low-energy operation for less demanding ones. Innovation in the semiconductors industry provided more transistors for roughly constant cost per chip, while computer architecture exploited the available transistor budget and discovered innovative techniques to scale systems' performance.

We have reached a point where transistor integration capacity will continue to scale, though with limited performance and power benefit. Computer architects reacted to this challenge with multicore architectures. The first systems developed followed an homogeneous architecture, while recent ones move gradually to heterogeneous architectures that look like complex platform Systems-on-Chip (SoCs) integrating in the same chip latency-optimized cores, throughput optimized cores (like GPUs) and some specialized cores that together with the associated memory hierarchies and memory controllers (mostly for off-chip DRAM) allows them to cover the needs of many application domains. SoCs for mobile devices were heterogeneous from the beginning including various specialized components such as display controllers, camera interfaces, sensors, connectivity modules such as Bluetooth, WiFi, FM radio, GNSS (Global Navigation Satellite System), and multimedia subsystems. Programming such heterogeneous systems in a unified manner is still an open challenge. Nevertheless, any revolutionary development in heterogeneous systems programming should rely on a solid computation and communication infrastructure that will aid and not limit the system-wide improvements.

Scalable interconnect architectures form the solid base on top of which heterogeneous computing platforms and their unifying programming environments will be developed; parallelism is all about cooperation that cannot be achieved without the equivalent concurrency in communication. The interconnect implements

G. Dimitrakopoulos et al., *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, DOI 10.1007/978-1-4614-4301-8_1

the physical and logical medium for any kind of data transfer and its latency, bandwidth and energy efficiency directly affects overall system performance. Interconnect design is a multidimensional problem involving hardware and software components such as network interfaces, routers, topologies, routing algorithms and communication programming interfaces.

Modern heterogeneous multiprocessing systems have adopted a Network-on-Chip (NoC) technology that brings interconnect architectures inside the chip. The NoC paradigm tries to find a scalable solution to the tough integration challenge of modern SoCs, by applying at the silicon chip level well established networking principles, after suitably adapting them to the silicon chip characteristics and to application demands (Dally and Towles 2001; Benini and Micheli 2002; Arteris 2005). While the seminal idea of applying networking technology to address the chip-level interconnect problem has been shown to be adequate for current systems (Lecler and Baillieu 2011), the complexity of future computing platforms demands new architectures that go beyond physical-related requirements and equally participate in delivering high-performance, quality of service, and dynamic adaptivity at the minimum energy and area overhead (Bertozzi et al. 2014; Dally et al. 2013).

The NoC is expected to undertake the expanding demands of the ever increasing numbers of processing elements, while at the same time technological and application constraints increase the pressure for increased performance and efficiency with limited resources. Although NoC research has evolved significantly the last decade, crucial questions remain un-answered that call for fresh research ideas and innovative solutions. Before diving in the details of the router microarchitecture that is the focus of this book, we will briefly present in this chapter the technical issues involved in the design of a NoC as a whole and how it serves its goal for offering efficient system-wide communication.

1.1 The Physical Medium

The available resources that the designer has at the physical level are transistors and wires. Using them appropriately the designer can construct complex circuits that are designed at different abstraction levels, following either custom or automated design methodologies. Interconnect architectures should use these resources in the most efficient manner offering a globally optimum communication medium for the components of the system.

The wires are used as the physical medium for transferring information between any two peers. On-chip wires are implemented in multiple metal layers that are organized in groups (Weste and Harris 2010), as shown in Fig. 1.1. Each group satisfies a specific purpose for the on-chip connectivity. The first metal layers are tailored for local connectivity and are optimized for on-chip connections spanning up to several hundreds of μm. They offer highly dense connections that allow thousands of bits to be transferred in close distance. Upper metal layers, are built

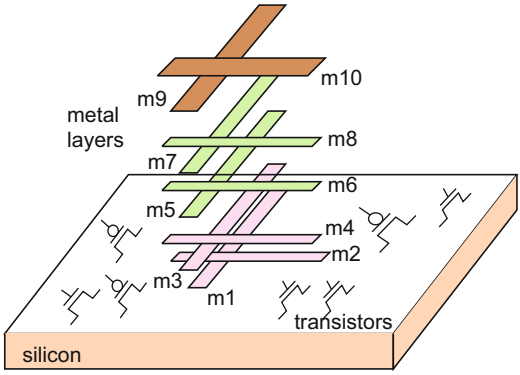

Fig. 1.1 The transistor and the metal layers of an integrated circuit

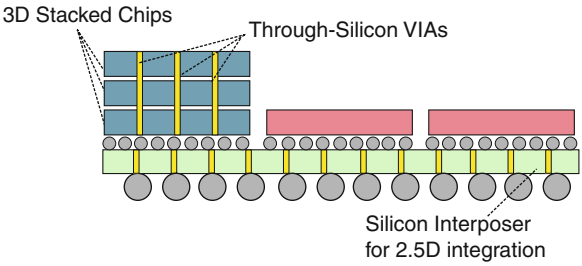

Fig. 1.2 2.5D and 3D integration possibilities for large SoCs

with larger cross sections, that offer lower resistance, and allow transferring bits in longer distance with lower delay. Due to manufacturing limitations upper metal layers should be placed further apart and should have a larger minimum width thus limiting the designer to use less wires per connection bus. Still, the wires that belong to the upper metal layers can be a very useful resource since they allow crossing several mms of on-chip distance very fast (Golander et al. 2011; Passas et al. 2010). In every case, using wisely the density of the upper and the lower metal layers allows for the design of high-bandwidth connections between any two components (Ho et al. 2001).

Technology improvement provides the designer with more connectivity. For example 2.5D integration offers additional across-chip wires with good characteristics allowing fast connections within the same package using the vertical through-silicon vias of a silicon interposer (Maxfield 2012) as depicted in Fig. 1.2. On the other hand, 3D integration promises even more dense connectivity by allowing vertical connections across different chips that are stacked on top of each other offering multiple layers of transistor and metal connections. Instead of allowing stacked chips to communicate using wired connections, short-distance wireless connectivity can be used instead, using, either inductive, or capacitive data transfer across chips (Take et al. 2014). Finally, instead of providing more wiring

connections as is the mainstream approach followed so far, several other research efforts try to provide a better communication medium for the on-chip connections utilizing on-chip optical connections (Bergman et al. 2014).

1.2 Flow Control

At the system level, using only a set of data wires in a communication channel between two peers (a sender and a receiver module) is not enough. The receiver should be able to distinguish the old data sent by the sender from the new data that it sees at its input. Also, the sender should be informed if the data that has sent has been actually accepted by the receiver or not. Therefore, some additional form of information needs to be conveyed across the sender and the receiver that would allow them to understand when a transaction between them has been completed successfully. Such information is transferred both in the forward and in the backward direction, as depicted in Fig. 1.3, and constitutes the flow-control mechanism.

The flow control mechanism can be limited at the borders of a single wire (called link-level flow control) or it can be expanded between any source and destination possibly covering many links and thus called end-to-end flow control (Gerla and Kleinrock 1980). Figure 1.3 tries to explain graphically the difference between the local and the global flow control mechanisms. While link-level flow control is explicitly implemented by the additional flow control wires of the link, end-to-end flow control can be either explicitly or implicitly implemented in a NoC environment. Explicit implementation requires several flow control wires arriving at each node from different destinations, that each one would describe the status of the corresponding connection. Implicit implementation means that any source or destination node has a mechanism to understand the status of the other side using the normal or special messages transmitted between them. Message transmission in this case, would have used all the intermediate links between the source and destination pair.

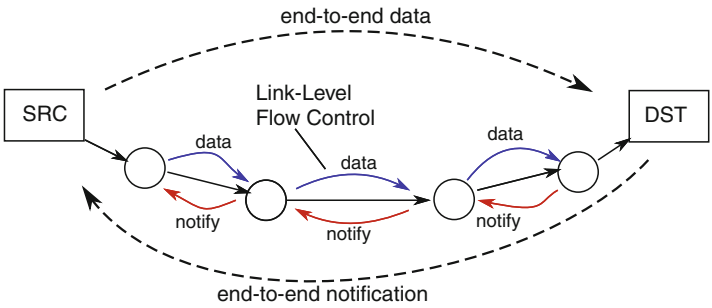

Fig. 1.3 Link-level and end-to-end flow control

Flow-control strategies are connected in one or in another way with the availability of buffering positions either at the other end of the link or at the destination. Therefore, the semantics of the flow-control protocol lead to various constraints regarding the implementation of the buffering alternatives.

The messages transferred across any two peers depend on the applications running on the system. Therefore, it is very common the granularity of the messages that are transferred at the application level to be different from the physical wiring resources available on the links. The selection of the channel width depends on a mix of constraints that span from application-level requirements down to physical chip-level integration limitations.

The messages between a source and a destination can be short and fit the channel width or can be longer and need to be serialized to many words that traverse the link in multiple cycles. This attribute should be also reflected to the flow-control mechanism that decides the granularity to which it allocates the channels and the buffers at the receive side. Coarse-grained flow control treats each message (or packet) as an atomic entity, while fine-grained flow control mechanisms operate at the sub-message (sub-packet) level, allowing parts of the message to be distributed to several stages.

1.3 Read–Write Transactions

Besides simple data transfers between two IP cores on the same chip, the exchange of information across multiple IP cores requires the implementation of multiple interfaces between them that would allow them to communicate efficiently and implement high-level protocol semantics. In widely accepted interfaces such as AMBA AXI and OCP-IP, each core should implement distinct and independently flow-controlled interfaces for writing, and reading from another IP core, including also interfaces for transferring additional notification messages (ARM 2013; Accelera 2013). An example of two connected cores via a single channel, where each core implements the full set of interfaces needed by AXI is shown in Fig. 1.4.

In most cases, where such address-based load/store transactions are used for the communication of two IPs the interfaces shown in Fig. 1.4 suffice to describe the needed functionality. The implementation of these interfaces and respecting the rules that come with the associated interconnect protocol, e.g., AXI, constitute the transaction layer of the network-on-chip communication architecture. Every transaction is initiated by a core (called the master for this transaction) via the request interface (read or write) and completed via the corresponding reply interface, while it may include an additional transaction response. Each transaction always involves a master and a slave core (that receives and services the request), while the two peers of a transaction are identified by the address used in the request and reply interfaces. Transaction-layer communication is an end-to-end operation between a master and slave and its definition, besides the support for the necessary physical interfaces, does not constrain the designer on how to implement it.

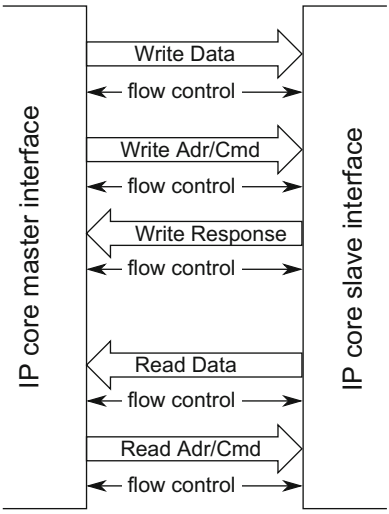

Fig. 1.4 An example of master and slave interfaces as needed by the AXI transaction protocol

1.4 Transactions on the Network: The Transport Layer

Directly supporting all the interfaces of the transaction layer in all links of the system is an overkill that requires an enormous number of wiring resources. Following the encapsulation principle followed by any network, the required interfaces can be substituted by transport layer interfaces that exchange packets of information that include in their headers the information delivered by each encapsulated interface (Mathewson 2010). Each packet can be either a read or a write packet consisting of a header word and some payload words.The packet header encodes the read/write address of the transaction and all other transaction parameters and control signals included in the original transaction-layer interface. Also, the header signal should include the necessary identification information that would guide the packet to its appropriate destination.

1.4.1 Network Interfaces

Interfacing between the transport and the transaction layer of communication is done at the network interfaces (NIs), located at the NoC periphery. The NI is responsible for both sending packets to the network as well as receiving packets from the network and after the appropriate manipulation to present it to the connected IP core according to the semantics of the transaction-layer interface (Saponara et al. 2014).

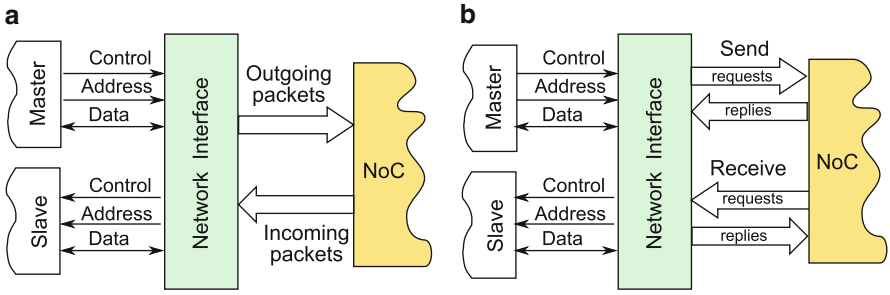

Fig. 1.5 Connection of the network interfaces using (**a**) simple connections to the network or (**b**) separate request and reply connections

For example as shown in Fig. 1.5a, the NI connected to a master implements a slave interface, while a NI connected to slave acts as a master to it. At the network's side, the send and receive paths at the edge of the NoC and the NI act as two independent flow-controlled channels that transfer packets according to the rules imposed by the transport layer.

The request and the responses of the transaction layer often assume that they are completely independent and isolated from each other thus eliminating any logical and architectural dependencies and allowing for deadlock-free operation at the transaction-protocol level. Enabling this separation by default at the transaction layers means that the transport and the physical layer provide a packet isolation mechanism. At the transport layer, this means that different packet classes such as request and reply packets should not interfere in the network in such a way that creates dependencies between them that may lead to a deadlock condition.

This can happen by imposing isolation either in space or in time. Isolation in space means that each packet class uses completely separated physical resources (separate request/reply channels, different switching mechanism), e.g., like adding different lanes on a road network for the different types of cars we don't want to interfere (see Fig. 1.5b) (Wentzlaff et al. 2007; Kistler et al. 2006). On the other hand, isolation in time means that different time slots are used by different packet classes. This time-sharing mechanism is equivalent to emulating the different lanes of a road network by virtual lanes, called virtual channels that each one appears at the physical channel in a different time instance (Dally 1992).

Any isolation mechanism implemented either in space or in time can be also used for providing deadlock-free routing for the packets travelling in the network. A routing deadlock can happen when a set of packets request access to already allocated channels and the chain of dependencies evolve in a cyclic manner that blocks any packet from moving forward (Duato et al. 1997).

1.4.2 The Network: The Physical Layer

The packets generated by the NIs reach their destination via a network of routers and links that are independently flow-controlled and form an arbitrary topology (Balfour and Dally 2006; Kim et al. 2007). Each router, in parallel to the network links, can connect to one or multiple NIs thus allowing to some of the cores of the system to communicate locally without their data to enter the network (Kumar et al. 2009).

At the network, the main issues that need to be resolved is handling connectivity and contention. Connectivity means that any two IP cores connected to the network via their NIs should be able to exchange information irrespective of their physical placement on the chip. Contention on the other hand is the result of offering connectivity via shared channels. Handling contention at the physical layer requires arbitration, multiplexing and buffering. In the example shown in Fig. 1.6, many IP cores are eligible to access the memory controller (RAM). However, in each clock cycle only one of them will actually transfer its data to it. The selection of the winning IP core is done by the arbitration logic and the movement of data is done via the switching multiplexers that exist inside each router. The IPs that lost in arbitration keep their data/packets in local buffers waiting to be selected in the next arbitration rounds.

While link-level flow control enables lossless operation across a sender and a receiver in a one-to-one connection, and arbitration and multiplexing enable sharing a link by many peers, real networks involve more complex switching cases that involve many to many connections. Each router should concurrently support all input-output permutations and solve the contention to all outputs at once respecting also the flow-control policy of the output links. Establishing a path between any source and destination of a complex network topology is a matter of the routing algorithm that is either implemented completely at the NIs or by the routers in a step-by-step and distributed manner.

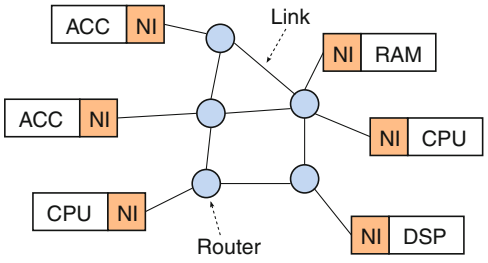

Fig. 1.6 A network-on-chip consisting of routers and links that reach the system's modules via the network interfaces (*NI*)

1.5 Putting It All Together

Initially assume that the CPU of the example system shown in Fig. 1.7 wants to read from an address that is stored in a memory in the other side of the chip. The NI of the CPU packetizes the read transaction including all the necessary control and addressing information that will allow the read request of the CPU to reach the memory controller. The NI acting as a packet source sends the read request packet to the first router. The router parses the header of the packet and understands to which output it should forward the incoming packet. Assuming that no other packet wants to leave from the same output and there is buffer space available to the next router, the first router forwards the packet to the next router. The following router will execute exactly the same tasks and finally the packet will reach the NI of the memory (RAM). The NI of the RAM parses the incoming packet and presents the read transaction to the slave memory controller. The memory (slave) produces the requested data and tries to send it back to the master that requested them. The NI of the RAM packetizes the reply data and using the network of routers allows the reply packet to reach the NI of the CPU (master). The CPU gets the necessary data in the appropriate interface of the transaction-layer protocol.

Using this network of routers multiple transactions could have completed in parallel between different master and slave pairs. When two or more packets want to move using the same link, the router solves the contention and serializes appropriately the requesting packets.

As in any network, the fundamental operation of a NoC is based on protocol layering that allows the decomposition of the network's functionality to simpler tasks, hides the implementation details of each layer and enables the network resources to be shared by allowing multiple transactions to execute on the same communication medium.

Following Fig. 1.8, each layer of the network acts as a service provider to the higher layers while it acts as a service user of the lower layers. Each layer can be implemented, optimized, and upgraded independently from the other layers thus allowing for maximum flexibility at network design and SoC integration phases. The main benefit of this layered design approach is that multiple different implementations of a layer can exist depending on the application domain and the

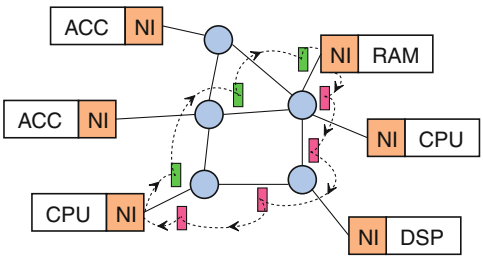

Fig. 1.7 Transfer of information across the network between two system's cores

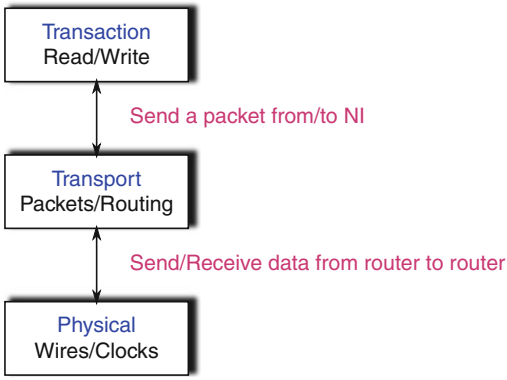

Fig. 1.8 Layered approach in network-on-chip design

technology node used for the system. For example, a network can employ different link widths and flow control mechanisms (Mishra et al. 2011) or even clocking at the physical layer without affecting the operation of the transport and transaction layers of communication.

1.6 Take-Away Points

The Network-on-chip paradigm solves the problem of on-chip communication by applying at the silicon chip level well established networking principles, after suitably adapting them to the silicon chip characteristics and to application demands. Network-on-chip design evolves in a layered approach that allows the transformation of abstract load/store transactions to packets of bits that travel in the network following the correct path from their source to their destination. The transformation between transactions and packets is done at the network interfaces, while the routers provide arbitrary lossless connectivity between inputs and outputs and allow for the implementation of arbitrary network topologies.