# 95

# Network-on-Chip Design for Gigascale Systems-on-Chip

Davide Bertozzi
*University of Bologna*

Luca Benini
*University of Bologna*

Giovanni De Micheli
*Stanford University*



## 95.1 Introduction

The increasing integration densities made available by shrinking device geometries will have to be exploited to meet the computational requirements of parallel applications such as multimedia processing, automotive, multi-window TV, ambient intelligence, etc.

As an example, systems designed for ambient intelligence will be based on high-speed digital signal processing with computational loads ranging from 10 MOPS for lightweight audio processing, 3 GOPS for video processing, 20 GOPS for multilingual conversation interfaces, and up to 1 TOPS for synthetic video generation. This computational challenge will have to be addressed at manageable power levels and affordable costs [1].

Such a performance cannot be provided by a single processor, but requires a heterogeneous on-chip multi-processor system containing a mix of general-purpose programmable cores, application-specific processors, and dedicated hardware accelerators.

In this context, performance of gigascale *Systems-on-Chip* (SoCs) will be communication dominated, and only an interconnect-centric system architecture will be able to cope with this problem. Current on-chip interconnects consist of low-cost shared arbitrated buses, based on the serialization of bus access requests; only one master at a time can be granted access to the bus. The main drawback of this solution is its lack of scalability, which will result in unacceptable performance degradation for complex SoCs (more than a dozen of integrated cores). Moreover, the connection of new blocks to a shared bus increases its associated load capacitance, resulting in more energy-consuming bus transactions.

0-8493-1985-4/05/$0.00+$1.50

A scalable communication infrastructure that better supports the trend of SoC integration consists of an on-chip micro-network of interconnects, generally known as *Network-on-Chip* (NoC) architecture [2,6,7]. The basic idea is borrowed from the wide-area networks domain, and envisions router (or switch)-based networks on which on-chip packetized communication takes place, as depicted in Figure 95.1. Cores access the network by means of proper interfaces, and have their packets forwarded to destination through a certain number of hops.

The scalable and modular nature of NoCs and their support for efficient on chip communication potentially lead to NoC-based multi-processor systems characterized by high structural complexity and functional diversity. On the one hand, these features need to be properly addressed by means of new design methodologies [3], while on the other more efforts have to be devoted to modeling on-chip communication architectures and integrating them into a single modeling and simulation environment combining both processing elements and communication infrastructures [4,9,10]. These efforts are needed to include on-chip communication architecture in any quantitative evaluation of system design during design space exploration [8,11], to enable the assessment of the impact of the interconnect on achieving a target system performance.

An important design decision for NoCs regards the choice of topology. Several researchers [7,12,3,13] envision NoCs as regular tile-based topologies (such as mesh networks and fat trees), which are suitable for interconnecting homogeneous cores in a chip multiprocessor. However, SoC component specialization (used by designers to optimize performance at low power consumption and competitive cost) leads to the on-chip integration of heterogeneous cores having varied functionality, size, and communication requirements. If a regular interconnect is designed to match the requirements of a few communication-hungry components, it is bound to be largely overdesigned with respect to the needs of the remaining components. This is the main reason why most current SoCs use irregular topologies like bridged buses and/or dedicated point-to-point links [14].

This chapter introduces the basic principles and guidelines for the NoC design. At first, the motivation for the design paradigm shift of SoC communication architectures from shared buses to NoCs is examined. Then, the chapter goes into the details of NoC building blocks (switch, network interface, and switch-to-switch links), discussing the design guidelines and presenting a case study where some of the most advanced concepts in NoC design have been applied to a real NoC architecture (called *Xpipes* and developed at University of Bologna [15]).

Finally, the challenging issue of heterogeneous NoC design will be addressed, and the effects of mapping the communication requirements of an application onto a domain-specific NoC, instead of a network with regular topology, will be detailed by means of an illustrative example.

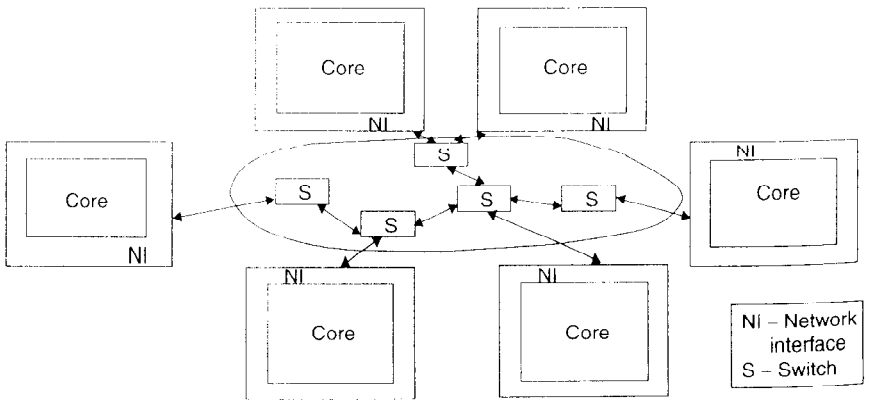



FIGURE 95.1 Example of network-on-chip architecture.

## 95.2 Design Challenges for On-Chip Communication Architectures

SoC design challenges that are driving the evolution of traditional bus architectures toward NoCs can be outlined as follows:

*Technology issues*: While gate delays scale down with technology, global wire delays typically increase or remain constant as repeaters are inserted. It is estimated that in 50 nm technology, at a clock frequency of 10 GHz, a global wire delay can be up to 6–10 clock cycles [2]. Therefore, limiting the on-chip distance traveled by critical signals will be key to guarantee the performance of the overall system, and will be a common design guideline for all kinds of system interconnects. On the contrary, other challenges posed by deep sub-micron technologies are leading to a paradigm shift in the design of SoC communication architectures. For instance, global synchronization of cores on future SoCs will be unfeasible due to deep submicron effects (clock skew, power associated with clock distribution tree, etc.), and an alternative scenario consists of self-synchronous cores that communicate with one another through a network-centric architecture [16]. Finally, signal integrity issues (cross-talk, power supply noise, soft errors, etc.) will lead to more transient and permanent failures of signals, logic values, devices, and interconnects, thus raising the reliability concern for on-chip communication [17]. In many cases, on-chip networks can be designed as regular structures, allowing electrical parameters of wires to be optimized and well controlled. This leads to lower communication failure probabilities, thus enabling the use of low swing signaling techniques [18], and to the capability of exploiting performance optimization techniques such as wavefront pipelining [19].

*Performance issues*: In traditional buses, all communication actors share the same bandwidth. As a consequence, performance does not scale with the level of system integration, but degrades significantly, although once the bus is granted to a master, access occurs with no additional delay. On the contrary, NoCs can provide a much better performance scalability. No delays are experienced for accessing the communication infrastructure, since multiple outstanding transactions originating from multiple cores can be handled at the same time, resulting in a more efficient network resources utilization. However, given a certain network dimension (e.g., the number of instantiated switches), large latency fluctuations for packet delivery could be experienced as a consequence of network congestion. This is unacceptable when hard real-time constraints of an application have to be met, and two solutions are viable: network overdimensioning (for NoCs designed to support best-effort traffic only) or implementation of dedicated mechanisms to provide guarantees for timing-constrained traffic (e.g., loss-less data transport, minimal bandwidth, bounded latency, minimal throughput, etc.) [20].

*Design productivity issues*: It is well known that synthesis and compiler technology development do not keep up with IC manufacturing technology development [21]. Moreover, times-to-market need to be kept as low as possible. Reuse of complex preverified design blocks is an efficient means to increase productivity, and considers both computation resources and the communication infrastructure [22]. It would be highly desirable to have processing elements that could be used in different platforms by means of a plug-and-play design style. To this end, a scalable and modular on-chip network represents a more efficient communication infrastructure compared with shared bus-based architectures. However, the reuse of processing elements is facilitated by the definition of standard network interfaces, which also make the modularity property of the NoC effective. The *Virtual Socket Interface Alliance* (VSIA) has attempted to set the characteristics of this interface industrywide [23]. OCP [24] is another example of standard interface sockets for cores. It is worth remarking that such network interfaces also decouple the development of new cores from the evolution of new communication architectures. The core developer will not have to make assumptions about the system, when the core will be plugged into. Similarly, designers of new on-chip interconnects will not be constrained by the knowledge of detailed interfacing requirements for particular legacy SoC components. Finally, let us observe that NoC components (e.g., switches or interfaces) can be instantiated multiple times in the same design as opposed to the arbiter of traditional shared buses (which is instance-specific) and reused in a large number of products targeting a specific application domain.

The developments of NoC architectures and protocols are fueled by the aforementioned arguments, in spite of the challenges represented by the need for new design methodologies and an increased complexity of system design.

## 95.3 Related Work

The need to progressively replace on-chip buses with micro-networks was extensively discussed in [2,7]. A number of NoC architectures have been proposed in the literature so far.

*Sonics MicroNetwork* [25] is an on-chip network making use of communication architecture-independent interface sockets. The *MicroNetwork* is an example of evolutionary solutions [26], which move from a physical implementation as a shared bus, and propose generalizations to support a higher bandwidth (such as partial and full crossbars).

*STBUS* interconnect from STMicroelectronics is another example of evolutionary architecture, which provides designers with the capability to instantiate both shared bus or partial or full crossbar interconnect configurations.

Even though these architectures provide a bandwidth higher than simple buses, addressing the wiring delay and scalability challenge in the long term requires more radical solutions.

One of the earliest contributions in this area is the *Maia* heterogeneous signal processing architecture, proposed by Zhang et al., [27] based on a hierarchical mesh network. Unfortunately, *Maia*'s interconnect is fully instance-specific. Furthermore, routing is static at configuration time: network switches are programmed once and for all for a given application (as in an FPGA). Thus, communication is based on circuit switching, as opposed to packet switching.

In this direction, Dally and Lacy sketch the architecture of a VLSI multi-computer using 2009 technology [35]. A chip with 64 processor-memory tiles is envisioned. Communication is based on packet switching. This seminal work draws upon past experiences in designing parallel computers and reconfigurable architectures (FPGAs and their evolutions) [28–30].

Most proposed NoC platforms are packet switched and exhibit a regular structure. An example is a mesh interconnection, which can rely on a simple layout and the switch independence on the network size. The *NOSTRUM* network described in [3] adopts this approach: the platform includes both a mesh architecture and the design methodology. The *Scalable Programmable Integrated Network (SPIN)* described in [31] is another regular, fat-tree-based network architecture. It adopts cut-through switching to minimize message latency and storage requirements in the design of network switches. The *Linkoeping SoCBUS* [32] is a two-dimensional mesh network, which uses a *packet connected circuit* (PCC) to set up routes through the network: a packet is switched through the network locking the circuit as it goes. This notion of virtual circuit leads to deterministic communication behavior but restricts routing flexibility for the rest of the communication traffic.

The need to map communication requirements of heterogeneous cores may lead to the adoption of irregular topologies. The motivation for such architectures lies in the fact that each block can be optimized for a specific application (e.g., video or audio processing), and link characteristics can be adapted to the communication requirements of the interconnected cores. Supporting heterogeneous architectures requires a major design effort and leads to coarser-granularity control of physical parameters. Many recent heterogeneous SoC implementations are still based on shared buses (such as the single chip MPEG-2 codec reported in [33]), but the growing complexity of customizable media-embedded processor architectures for digital media processing will soon require NoC-based communication architectures and proper hardware/software development tools. The *Aethereal* NoC design framework presented in [34] aims at providing a complete infrastructure for developing heterogeneous NoC with end-to-end quality of service guarantees. The network supports *guaranteed throughput* (GT) for real-time applications and *best-effort* (BE) traffic for timing unconstrained applications.

Support for heterogeneous architectures requires highly configurable network building blocks, customizable at instantiation time for a specific application domain. For instance, the *Proteo* NoC [36]

consists of a small library of predefined, parameterized components that allow the implementation of a large range of different topologies, protocols, and configurations.

*Xpipes* interconnect [15] and its synthesizer *XpipesCompiler* [41] push this approach to the limit, by instantiating an application-specific NoC from a library of composable soft macros (network interface, link, and switch). The components are highly parameterizable and provide a reliable and latency-insensitive operation.

## 95.4 Network-on-Chip Architecture

Most of the terminology for on-chip packet switched communication is adapted from computer network and multiprocessor domain. Messages that have to be transmitted across the network are usually partitioned into fixed-length packets. Packets in turn are often broken into message flow control units called *flits*. In the presence of channel width constraints, multiple physical channel cycles can be used to transfer a single flit. A *phit* is the unit of information that can be transferred across a physical channel in a single step. Flits represent logical units of information, as opposed to phits that correspond to physical quantities. In many implementations, a flit is set to be equal to a phit. The basic building blocks for packet-switched communication across NoCs are:

I. network link
II. switch
III. network interface

and will be described hereafter.

### Network Link

The performance of interconnect is a major concern in scaled technologies. As geometries shrink, gate delay improves much faster than the delay in long wires. Therefore, the long wires increasingly determine the maximum clock rate, and hence performance, of the entire design. The problem becomes particularly serious for domain-specific heterogeneous SoCs, where the wire structure is highly irregular and may include both short and extremely long switch-to-switch links. Moreover, it has been estimated that only a fraction of the chip area (between 0.4 and 1.4%) will be reachable in one clock cycle [42].

A solution to overcome the interconnect-delay problem consists of pipelining interconnects [37,38]. Wires can be partitioned into segments (or relay stations, which have a function similar to the one of latches on a pipelined data path), whose length satisfies predefined timing requirements (e.g., desired clock speed of the design). In this way, link delay is changed into latency, but data introduction rate is not bounded by the link delay any more. Now, the latency of a channel connecting two modules may end up being more than one clock cycle. Therefore, if the functionality of the design is based on the sequencing of the signals and not on their exact timing, then link pipelining does not change the functional correctness of the design. This requires the system to be composed of modules whose behavior does not depend on the latency of the communication channels (latency-insensitive operation). As a consequence, the use of interconnect pipelining can be seen as a part of a new and more general methodology for *deep submicron* (DSM) designs, which can be envisioned as synchronous distributed systems composed of functional modules that exchange data on communication channels according to a latency-insensitive protocol. This protocol ensures that functionally correct modules behave correctly independent of the channel latencies [37]. The effectiveness of the latency-insensitive design methodology is strongly related to the ability of maintaining a sufficient communication throughput in the presence of increased channel latencies.

The *International Technology Roadmap for Semiconductors* (ITRS) 2001 [16] assumes that interconnect pipelining is the strategy of choice in its estimates of achievable clock speeds for MPUs. Some industrial designs already make use of interconnect pipelining. For instance, the NETBURST micro-architecture of Pentium 4 contains instances of a stage dedicated exclusively to handle wire delays: in fact, a so-called drive stage is used only to move signals across the chip without performing any computation and, therefore, can be seen as a physical implementation of a relay station [39].

*Xpipes* interconnect makes use of pipelined links and of a latency-insensitive operation in the implementation of its building blocks. Switch-to-switch links are subdivided into basic segments whose length guarantees that the desired clock frequency (i.e., the maximum speed provided by a certain technology) can be used. In this way, the system operating frequency is not bound by the delay of long links. According to the link length, a certain number of clock cycles are needed by a flit to cross the interconnect. If network switches are designed in such a way that their functional correctness depends on the flit arriving order and not on their timing, input links of the switches can be different and of any length. These design choices are the bases of latency-insensitive operation of the NoC and allow the construction of an arbitrary network topology and hence support for heterogeneous architectures.

Figure 95.2 illustrates the link model, which is equivalent to a pipelined shift register. Pipelining has been used both for data and control lines. The figure also illustrates how pipelined links are used to support latency-insensitive link-level error control, ensuring robustness against communication errors. The retransmission of a corrupted flit between two successive switches is represented. Multiple outstanding flits propagate across the link during the same clock cycle. When flits are correctly received at the destination switch, an ACK is propagated back to the source. After at least $2N$ clock cycles since transmission (where $N$ is the length of the link expressed in number of repeater stages), the flit will be discarded from the buffer of the source

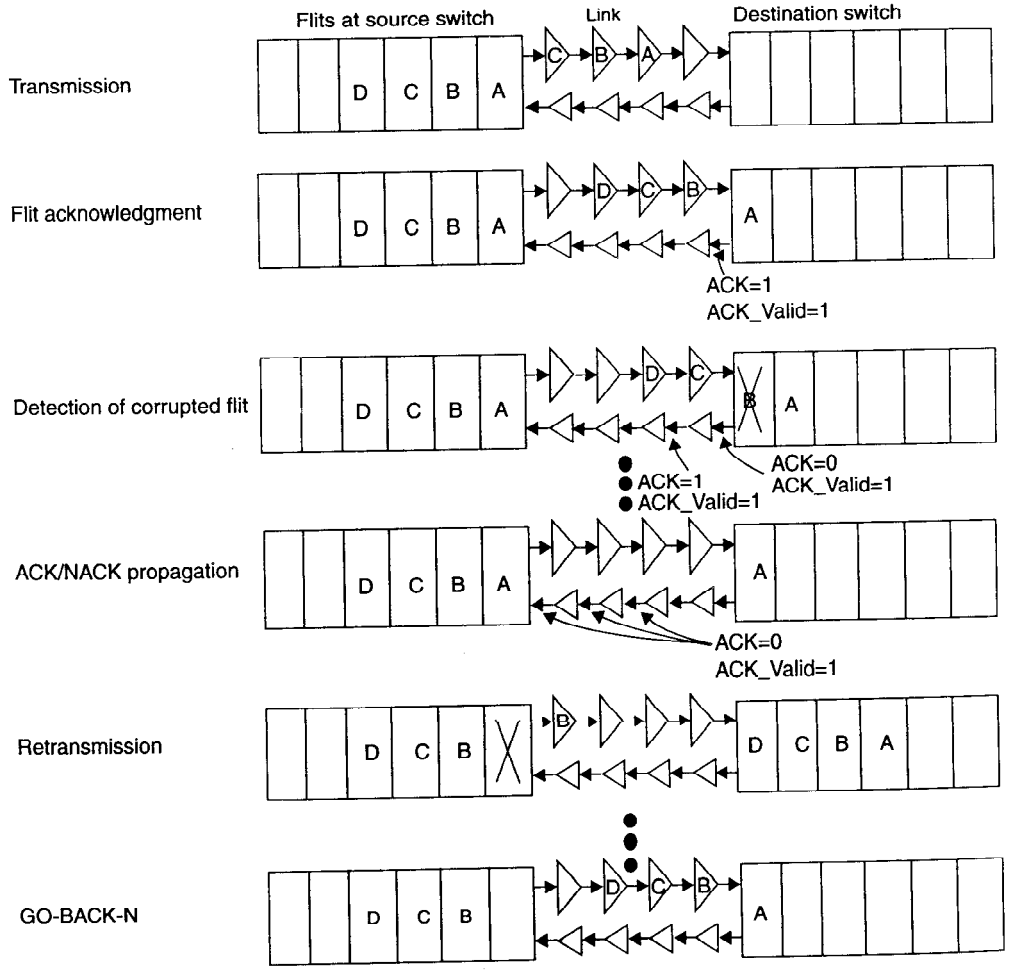



FIGURE 95.2 Pipelined link model and latency-insensitive link-level error control.

switch. On the contrary, a corrupted flit is NACKed and will be retransmitted in due time. The implemented retransmission policy is GO-BACK-N, to keep the switch complexity as low as possible.

## Switch

The task of the switch is to carry packets injected into the network to their final destination, following a statically defined or dynamically determined routing path. The switch transfers packets from one of its input ports to one or more of its output ports.

Switch design is usually characterized by a power-performance trade-off: power-hungry switch memory resources can be required by the need to support high-performance on-chip communication. A specific design of a switch may include both input and output buffers or only one type of buffers. Input queuing uses fewer buffers, but suffers from head-of-line blocking. Virtual output queuing has a higher performance, but at the cost of more buffers.

Network flow control (or routing mode) specifically addresses the limited amount of buffering resources in switches. Three policies are feasible in this context [5].

In *store-and-forward routing*, an entire packet is received and entirely stored before being forwarded to the next switch. This is the most demanding approach in terms of memory requirements and switch latency. Also, *virtual cut-through routing* requires buffer space for an entire packet, but allows lower latency communication, in that a packet is forwarded as soon as the next switch guarantees that the complete packet will be accepted. If this is not the case, the current router must be able to store the whole packet.

Finally, a *wormhole routing* scheme can be used to reduce switch memory requirements and to permit low latency communication. The first flit of a packet contains routing information, and header flit decoding enables the switches to establish the path and subsequent flits simply follow this path in a pipelined manner by means of switch output port reservation. A flit is passed to the next switch as soon as enough space is available to store it, even though there is not enough space to store the whole packet. If a certain flit faces a busy channel, subsequent flits have to wait at their current locations and are therefore spread over multiple switches, thus blocking the intermediate links. This scheme avoids buffering the full packet at one switch and keeps end-to-end latency low, although it is more sensitive to deadlock and may result in low link utilization.

Guaranteeing quality of service in switch operation is another important design issue, which needs to be addressed when time-constrained (hard or soft real time) traffic is to be supported. Throughput guarantees or latency bounds are examples of time-related guarantees.

Contention-related delays are responsible for large fluctuations of performance metrics, and a fully predictable system can be obtained only by means of contention-free routing schemes. With *circuit switching*, a connection is set up over which all subsequent data are transported. Therefore, contention resolution takes place at setup at the granularity of connections, and time-related guarantees during data transport can be given. In *time division circuit switching* (see [25] for an example), bandwidth is shared by time division multiplexing connections over circuits.

In packet switching, contention is unavoidable since packet arrival cannot be predicted. Therefore, arbitration mechanisms and buffering resources must be implemented at each switch, thus delaying data in an unpredictable manner and making it difficult to provide guarantees. Best-effort NoC architectures can mainly rely on network overdimensioning to bound fluctuations of performance metrics.

The *Aethereal* NoC architecture makes use of a router that tries to combine GT and BE services [34]. The GT router subsystem is based on a time-division multiplexed circuit switching approach. A router uses a *slot table* to (i) avoid contention on a link, (ii) divide up bandwidth per link between connections, and (iii) switch data to the correct output. Every slot table $T$ has $S$ time slots (rows), and $N$ router outputs (columns). There is a logical notion of synchronicity: all routers in the network are in the same fixed-duration slot. In a slot $s$, at most one *block* of data can be read/written per input/output port. In the next slot, the read blocks are written to their appropriate output ports. Blocks thus propagate in a store and forward manner. The latency a block incurs per router is equal to the duration of a slot and bandwidth is guaranteed in multiples

of block size per $S$ slots. The BE router uses packet switching, and it has been showed that both input queuing with wormhole routing or virtual cut-through routing and virtual output queuing with wormhole routing are feasible in terms of buffering cost. The BE and GT router subsystems are combined in the *Aethereal* router architecture of Figure 95.3. The GT router offers a fixed end-to-end latency for its traffic, which is given the highest priority by the arbiter. The BE router uses all the bandwidth (slots) that has not been reserved or used by GT traffic. GT router slot tables are programmed by means of BE packets (see the arrow "program" in Figure 95.3). Negotiations, resulting in slot allocation, can be done at compile time, and be configured deterministically at runtime. Alternatively, negotiations can be done at runtime.

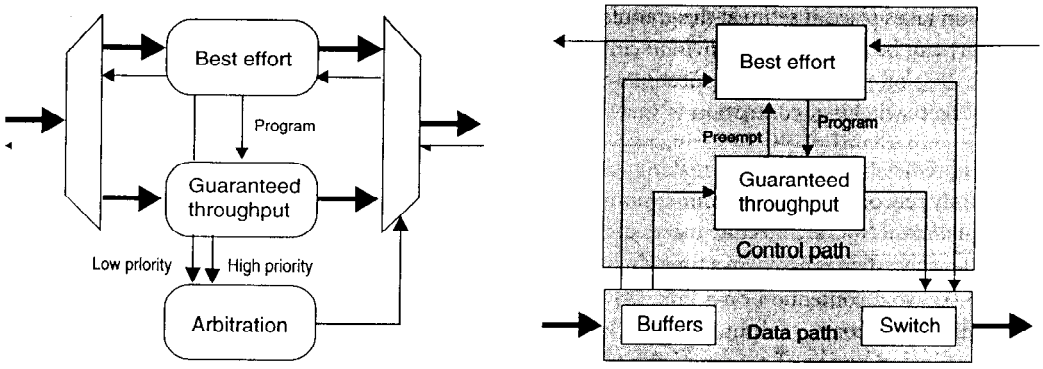
A different perspective has been taken in the design of the switch for the best-effort *Xpipes* NoC. Figure 95.4 shows an example configuration with four inputs, four outputs and two virtual channels multiplexed across the same physical output link. A physical link is assigned to different virtual channels on a flit-by-flit basis, thereby improving network throughput. Switch operation is latency insensitive, in that correct operation is guaranteed for arbitrary link pipeline depth. In fact, as explained above, network links in *Xpipes* interconnect are pipelined with a flexible number of stages, thereby decoupling link data introduction rate from its physical length.

For a latency-insensitive operation, the switch has virtual channel registers to store $2N + M$ flits, where $N$ is the link length (expressed as the number of basic repeater stages) and $M$ is a switch architecture-related contribution (12 cycles in this design). The reason is that each transmitted flit has to be acknowledged before being discarded from the buffer. Before an ACK is received, the flit has to travel across the link ($N$ cycles), an ACK/NACK decision has to be taken at the destination switch (a portion of $M$ cycles), and the ACK/NACK signal has to be propagated back ($N$ cycles) and recognized by the source switch (remaining portion of $M$ cycles). During this time, other $2N + M$ flits are transmitted but not yet ACKed.

Output buffering was chosen for *Xpipes* switches, and the resulting architecture is reported in Figure 95.5. It consists of a replication of the same output module, accepting all input ports as its own inputs. Flow control signals generated by each output block are directed to a centralized module, which takes care of generating proper ACKs or NACKs for the incoming flits from the different input ports.

Each output module is deeply pipelined (seven pipeline stages) to maximize the operating clock frequency of the switch. Architectural details on the pipelined output module are illustrated in Figure 95.6. Forward flow control is used and a flit is transmitted to the next switch only when adequate storage is available. The CRC decoders for error detection work in parallel with the switch operation, thereby concealing their impact on switch latency.

The first pipeline stage checks the header of incoming packets on different input ports to determine whether those packets have to be routed through the output port under consideration. Only matching packets are forwarded to the second stage, which resolves contention based on a round-robin policy. Arbitration is carried out against receipt of the tail flits of preceding packets, so that all other flits of a




(a) Conceptual view

(b) Hardware view

FIGURE 95.3 A combined GT–BE router.

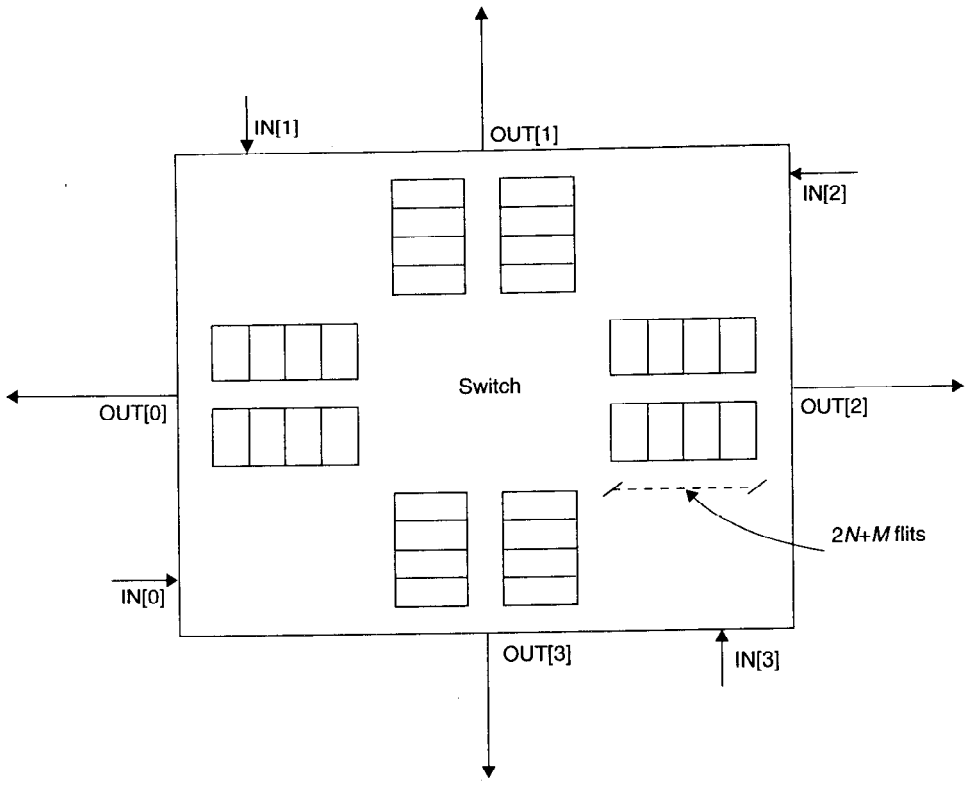



FIGURE 95.4 Example of switch configuration with two virtual channels.

packet can be propagated without contention resolution at this stage. A NACK for flits of nonselected packets is generated. The third stage is just a multiplexer, which selects the prioritized input port. The following arbitration stage retains the status of virtual channel registers and decides whether the flits can be stored in the registers or not. A header flit is sent to the register with more free locations, followed by successive flits of the same packet. The fifth stage is the actual buffering stage, and the ACK/NACK message at this stage indicates whether a flit has been successfully stored or not. The following stage takes care of forward flow control and finally a last arbitration stage multiplexes the virtual channels on the physical output link.

Finally, the switch is highly parameterizable. Design parameters are: number of I/O ports, flit width, number of virtual channels, length of switch-to-switch links, and size of output registers.

## Network Interface

The most relevant tasks of the *network interface* (NI) are: (i) concealing the details about the network communication protocol to the cores, so that they can be developed independent of the communication infrastructure, (ii) communication protocol conversion (from end-to-end to network protocol), and (iii) data packetization (packet assembly, delivery, and disassembly).

The former objective can be achieved by means of standard interfaces. For instance, the VSIA vision [23] is to specify open standards and specifications that facilitate the integration of software and hardware virtual components from multiple sources. Different complexity interfaces are described in the standard, from Peripheral *Virtual Complexity Interfaces* (VCI) to Basic VCI and Advanced VCI.

Another example of standard socket to interface cores to networks is represented by *Open Core Protocol* (OCP) [24]. Its main characteristics are a high degree of configurability to adapt to the core's

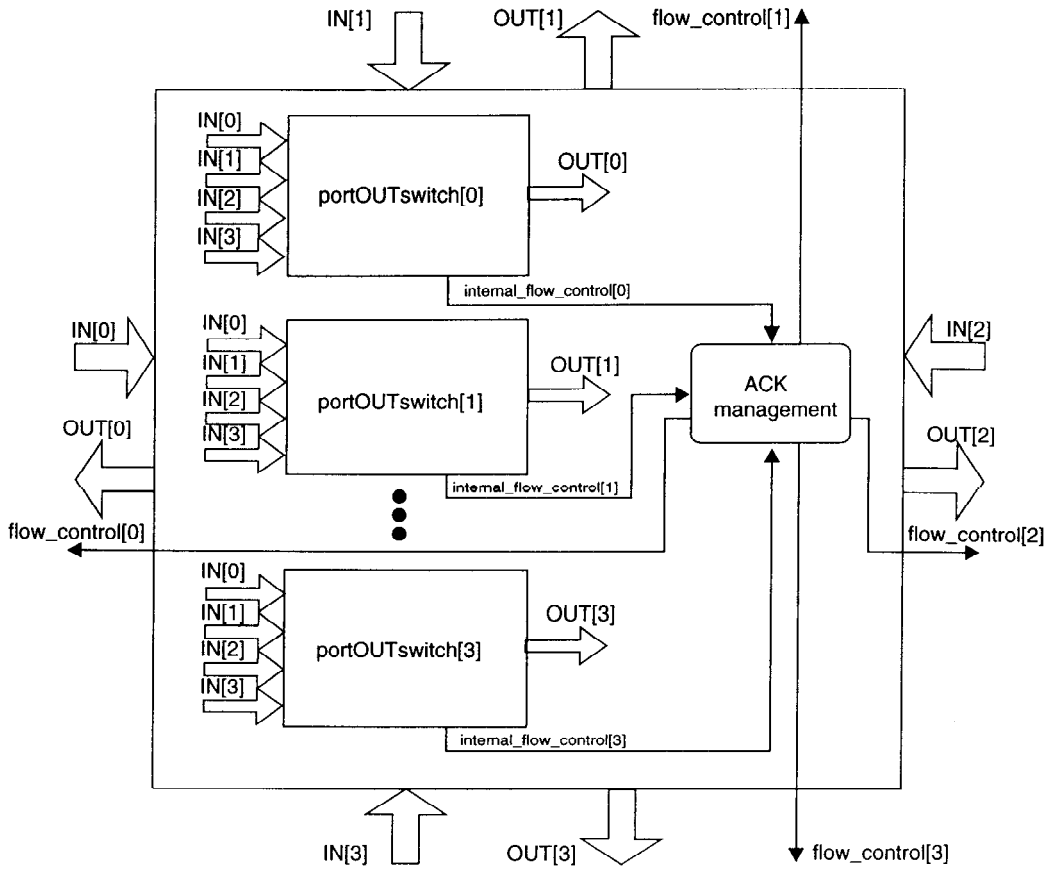



FIGURE 95.5 Architecture of output-buffered *Xpipes* switch.

functionality and the independence of request and response phases, thus supporting multiple outstanding requests and pipelining of transfers.

Data packetization is a critical task for the network interface, and has an impact on the communication latency, besides the latency of the communication channel. The packet preparation process consists of building packet header, payload, and packet tail. The header contains the necessary routing and network control information (e.g., source and destination address). When source routing is used, the destination address is ignored and replaced with a route field that specifies the route to the destination. This overhead in terms of packet header is counterbalanced by the simpler routing logic at the network switches: they simply have to look at the route field and route the packet over the specified switch output port. The packet tail indicates the end of a packet and usually contains parity bits for error-detecting or error-correcting codes.

An insight into the *Xpipes* network interface implementation will provide an example of these concepts. It provides a standardized OCP-based interface to network nodes. The NI for cores that initiate communication (initiators) needs to turn OCP-compliant transactions into packets to be transmitted across the network. It represents the slave side of an OCP end-to-end connection, and it is therefore referred to as *network interface slave* (NIS). Its architecture is shown in Figure 95.7.

The NIS has to build the packet header, which has to be spread over a variable number of flits depending on the length of the path to the destination node. In fact, *Xpipes* relies on a static routing algorithm called *street sign routing*. Routes are derived by the network interface by accessing a lookup table based on the destination address. Such information consists of direction bits read by each switch and indicating the output port of the switch that flits belonging to a certain packet have to be directed to.

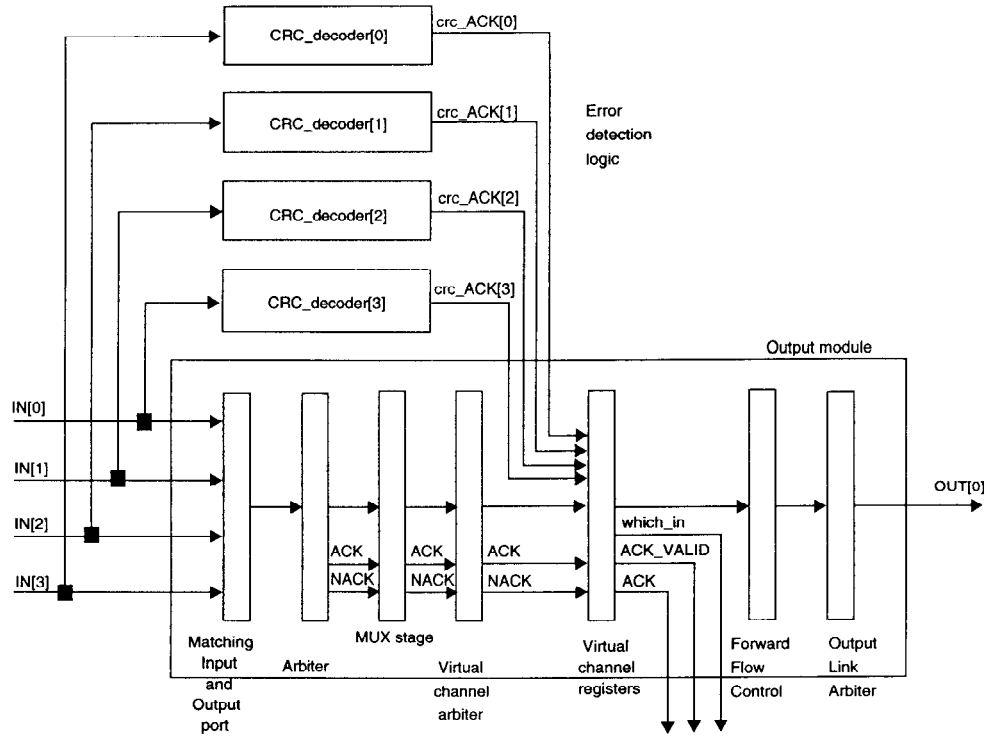



FIGURE 95.6 Architecture of an *Xpipes* switch output module.

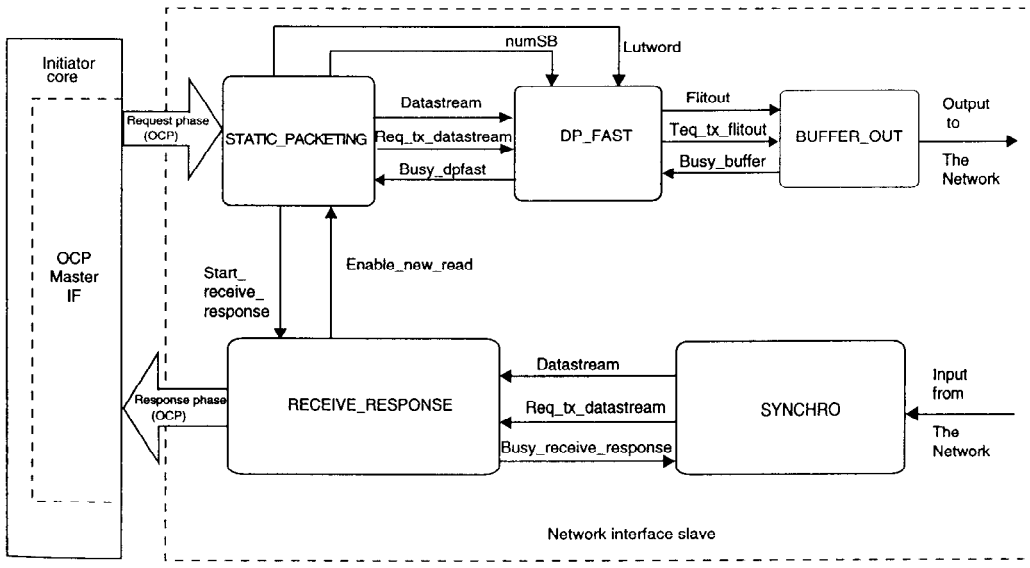



FIGURE 95.7 Architecture of the *Xpipes* network interface slave.

The lookup table is accessed by the *STATIC_PACKETING* block, a finite state machine that forwards the routing information *numSB* (number of hops to destination) and *lutword* (word read from the lookup table) as well as the request-related information *datastream* from the initiator core to the *DP_FAST* block, provided the enable signal *busy_dpfast* is not asserted.

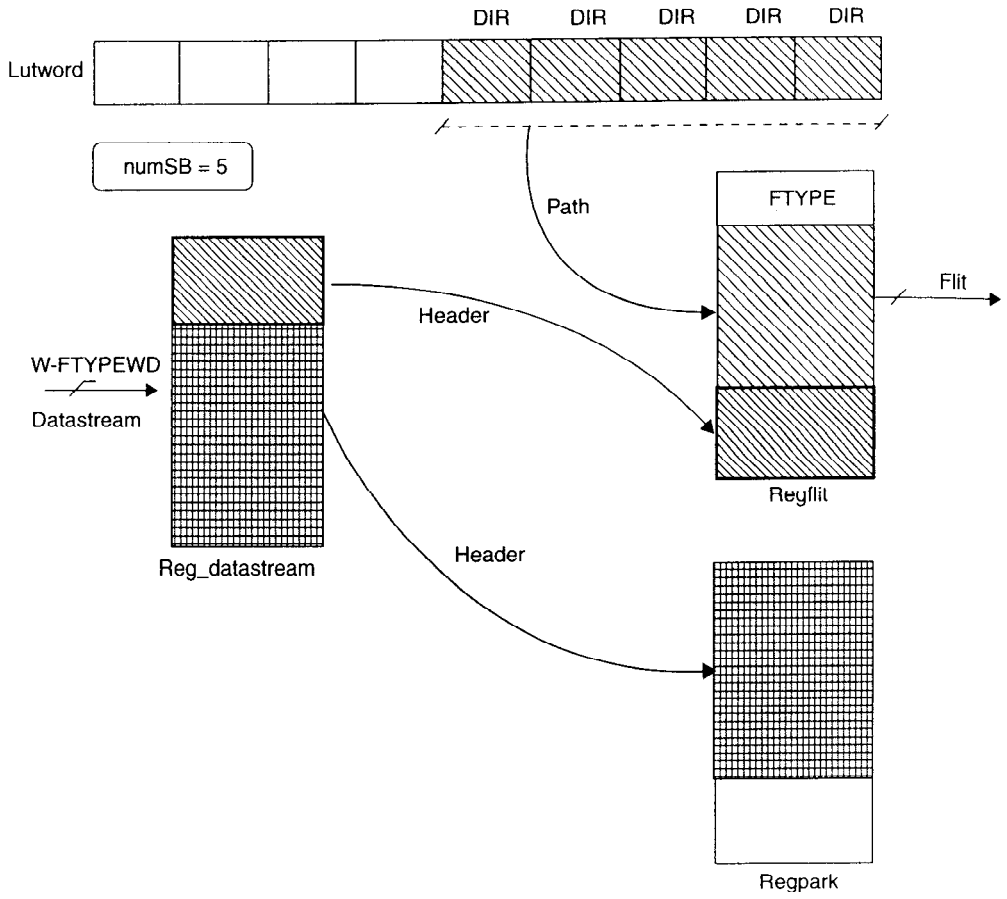


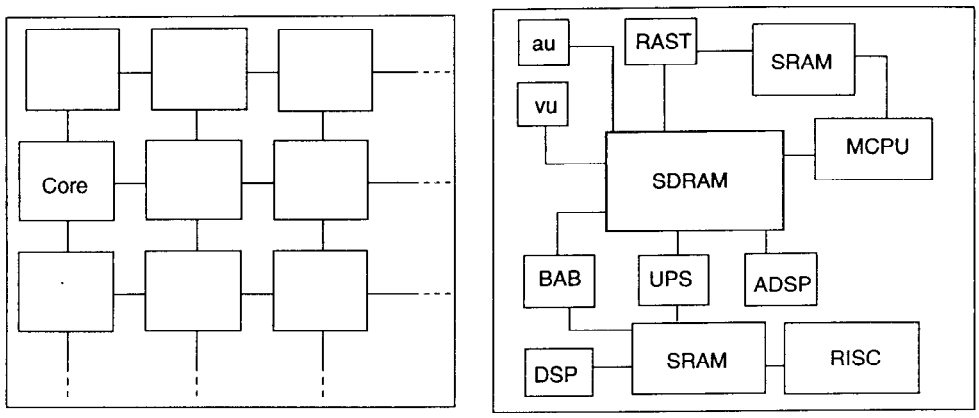

FIGURE 95.8 Mechanism for building header flits.

Based on the input data, module *DP_FAST* has the task of building the flits to be transmitted via the output buffer *BUFFER_OUT*, according to the mechanism illustrated in Figure 95.8. Let us assume that a packet requires *numSB* = 5 hops to get to the destination, and that the direction to be taken at each switch is expressed by *DIR*. Module *DP_FAST* builds the first flit by concatenating the flit-type field with path information. If there is some space left in the flit, it is filled with header information derived from the input *datastream*. The unused part of the *datastream* is stored in a *regpark* register, so that a new datastream can be read from the *STATIC_PACKETING* block. The following header and/or payload flits will be formed by combining data stored in *regpark* and *reg_datastream*. No partially filled flits are transmitted to make transmission more efficient.

Finally, module *BUFFER_OUT* stores flits to be sent across the network, and allows the NIS to keep preparing successive flits when the network is congested. The size of this buffer is a design parameter.

The response phase is carried out by means of two modules. *SYNCHRO* receives incoming flits and reads out only useful information (e.g., it discards route fields). At the same time, it contains buffering resources to synchronize the network's requests to transmit remaining packet flits with the core consuming rate. The *RECEIVE_RESPONSE* module translates useful header and payload information into OCP-compliant response fields.

When a read transaction is initiated by the master core, the *STATIC_PACKETING* block asserts a *start_receive_response* signal that triggers the waiting phase of the *RECEIVE_RESPONSE* module for the requested data. As a consequence, the NIS supports only one outstanding read operation to keep interface




(a) Tile-based on-chip multi-processor (b) MPEG-4 SOC

FIGURE 95.9 Homogeneous versus heterogeneous architectural template.

complexity low. Although no read after read transactions can be initiated unless the previous one has been completed, an indefinite number of write transactions can be carried out after an outstanding read has been initiated.

The architecture of a network interface master is similar to the one just described, and is not reported here due to lack of space. At instantiation time, the main network interface-related parameters to be set are: total number of core blocks, flit width, and maximum number of hops across the network.

## 95.5 Network-on-Chip Topology

The individual components of SoCs are inherently heterogeneous with widely varying functionality and communication requirements. The communication infrastructure should optimally match communication patterns among these components accounting for the individual component needs.

As an example, consider the implementation of an MPEG4 decoder [40], depicted in Figure 95.9 (b), where blocks are drawn roughly to scale and links represent interblock communication. First, the embedded memory (SDRAM) is much larger than all other cores and it is a critical communication bottleneck. Block sizes are highly nonuniform and the floorplan does not match the regular, tile-based floorplan shown in Figure 95.9 (a). Second, the total communication bandwidth to/from the embedded SDRAM is much larger than that required for communication among the other cores. Third, many neighboring blocks do not need to communicate. Even though it may be possible to implement MPEG4 onto a homogeneous fabric, there is a significant risk of either underutilizing many tiles and links, or, at the opposite extreme, of achieving poor performance because of localized congestion. These factors motivate the use of an application-specific on-chip network [27].

With an application-specific network, the designer is faced with the additional task of designing network components (e.g., switches) with different configurations (e.g., different I/Os, virtual channels, buffers) and interconnecting them with links of uneven length. These steps require significant design time and the need to verify network components and their communications for every design.

The library-based nature of network building blocks seems the more appropriate solution to support domain-specific custom NoCs. Two relevant examples have been reported in the open literature: *Proteo* and *Xpipes Interconnects*. *Proteo* consists of a fully reusable and scalable component library where the components can be used to implement networks from very simple bus emulation structures to complex packet networks. It uses a standardized VCI interface between the functional cores and the communication network. *Proteo* is described using synthesizable VHDL and relies on an interconnect node architecture that

targets flexible on-chip communication. It is used as a testing platform when the efficiency of network topologies and routing schemes are investigated for on-chip environments. The node is constructed from a collection of parameterized and reusable hardware blocks, including components such as FIFO buffers, routing controllers, and standardized interface wrappers. A node can be tuned to fulfill the desired characteristics of communication by properly selecting the internal architecture of the node itself.

*Xpipes* NoC adopts a similar approach. As described throughout this chapter, its network building blocks have been designed as highly configurable and design-time composable soft macros described in SystemC at the cycle-accurate level.

An optimal system solution will also require an efficient mapping of high-level abstractions onto the underlying platform. This mapping procedure involves optimizations and trade-offs between many complex constraints, including the quality of service, real-time response, power consumption, area, etc. Tools are urgently needed to explore this mapping process, and assist and automate optimization where possible.

The first challenge for these tools is to bridge the gap in building custom NoCs that optimally match the communication requirements of the system. The network components they build should be highly optimized for that particular NoC design, providing large savings in area, power, and latency with respect to standard NoCs based on regular structures.

In the following section, an example of design methodology for heterogeneous SoCs is briefly illustrated. It is relative to Xpipes interconnect and relies on a tool that automatically instantiates an application-specific NoC for heterogeneous on-chip multi-processors (called *XpipesCompiler* [41]).

## Domain Specific Network-on-Chip Synthesis Flow

The complete *XpipesCompiler*-based NoC design flow is depicted in Figure 95.10. From the specification of an application, the designer (or a high-level analysis and exploration tool) creates a high-level view of the SoC floorplan, including nodes (with their network interfaces), links and switches. Based on clock speed target and link routing, the number of pipeline stages for each link is also specified. The information on the network architecture is specified in an input file for the *XpipesCompiler*. Routing tables for the network interfaces are also specified. The tool takes as additional input the SystemC library of soft network components. The output is a SystemC hierarchical description, which includes all switches, links,

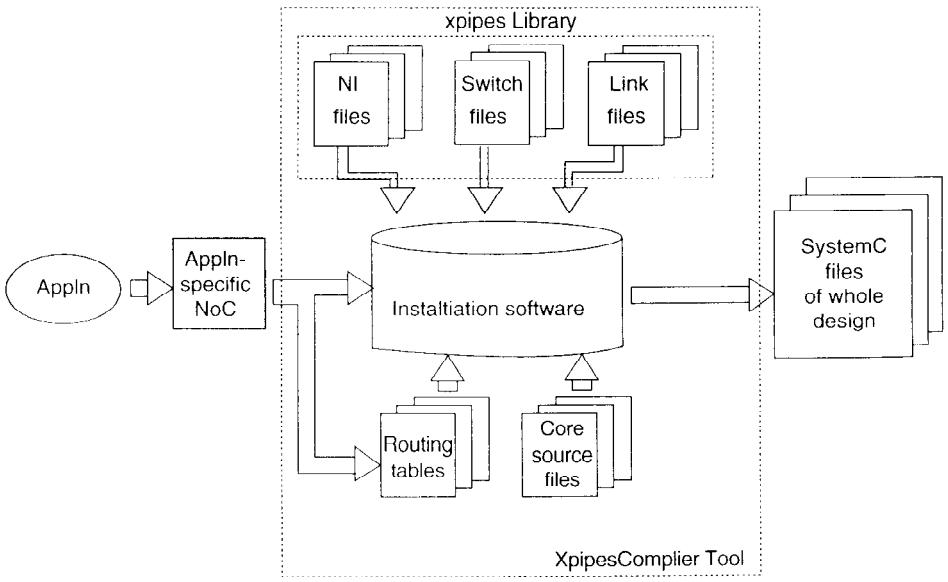



FIGURE.95.10 NoC synthesis flow with *XpipesCompiler*.

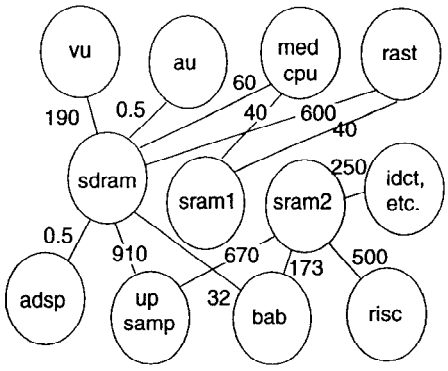



FIGURE 95.11 Core graph representation of an MPEG4 design with annotated average communication requirements.

TABLE 95.1 Area and power estimates for the MPEG4-related NoC configurations

| NoC Configuration | Area ($mm^2$) | Ratio mesh/cust | Power (mW) | Ratio mesh/cust |
|---|---|---|---|---|
| Mesh | 1.31 | | 114.36 | |
| Custom 1 | 0.86 | 1.52 | 110.66 | 1.03 |
| Custom 2 | 0.71 | 1.85 | 93.66 | 1.22 |

network nodes and interfaces, and specifies their topological connectivity. The final description can then be compiled and simulated at the cycle-accurate and signal-accurate level. At this point, the description can be fed to back-end RTL synthesis tools for silicon implementation.

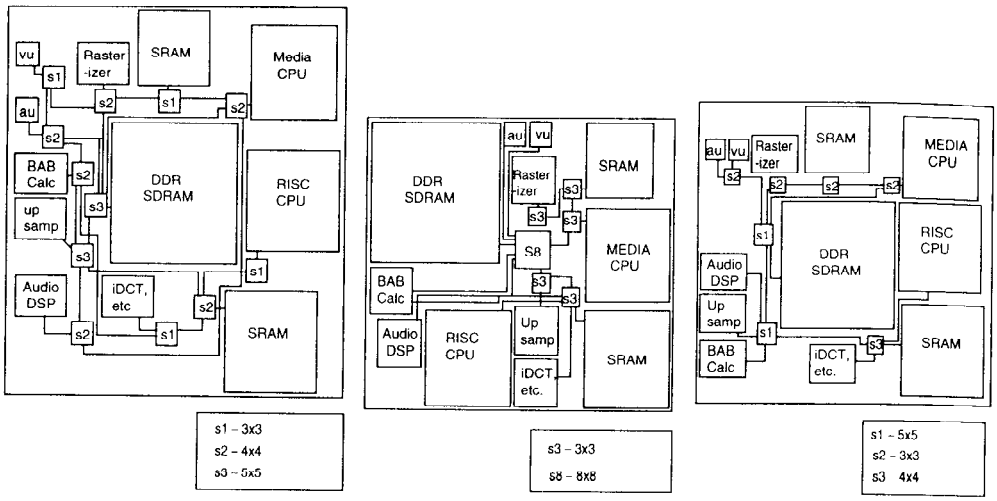
In a nutshell, the *XpipesCompiler* generates a set of network component instances that are custom-tailored to the specification contained in its input network description file. This tool allows a very instructive comparison of the effects (in terms of area, power, and performance) of mapping applications on customized domain-specific NoCs and regular mesh NoCs.

Let us focus on the MPEG4 decoder already introduced in this chapter. Its core graph representation together with its communication requirements are reported in Figure 95.11. The edges are annotated with the average bandwidth requirements of the cores in MB/s. Customized application-specific NoCs that closely match the application's communication characteristics have been manually developed and compared to a regular mesh topology. The different NoC configurations are reported in Figure 95.12. In the MPEG4 design considered, many of the cores communicate with each other through the shared SDRAM. Thus, a large switch is used for connecting the SDRAM with other cores (Figure 95.12 (b)), while smaller switches are connected to less communication-intensive cores. An alternate custom NoC is also considered (Figure 95.12 (c)): it is an optimized mesh network, with superfluous switches and switch I/Os removed.

Area (in 0.1 $\mu$m technology) and power estimates for the different NoC configurations are reported in Table 95.1. Since all cores communicate with many other cores, many switches are needed and therefore area savings are not extremely significant for custom NoCs.

Based on the average traffic through each network component, the power dissipation for each NoC design has been calculated. Power savings for the custom solutions are not very significant, as most of the traffic traverses the larger switches connected to the memories. As power dissipation on a switch increases nonlinearly with an increase in switch size, there is more power dissipation in the switches of custom NoC1 (that has an 8×8 switch) than the mesh NoC. However, most of the traffic traverses short links in this custom NoC, thereby giving marginal power savings for the whole design.

Figure 95.13 reports the variation of average packet latency (for 64B packets, 32-bit flits) with link bandwidth. Custom NoCs, as synthesized by *XpipesCompiler*, have lower packet latencies as the average number of switches and link traversals is lower. At the minimum plotted bandwidth value, almost 10% latency saving is achieved. Moreover, the latency increases more rapidly with the mesh NoC as the link




(a) Mesh NoC (b) AppIn-specific NoC1 (c) AppIn-specific NoC2

**FIGURE 95.12** NoC configurations for MPEG4 decoder.

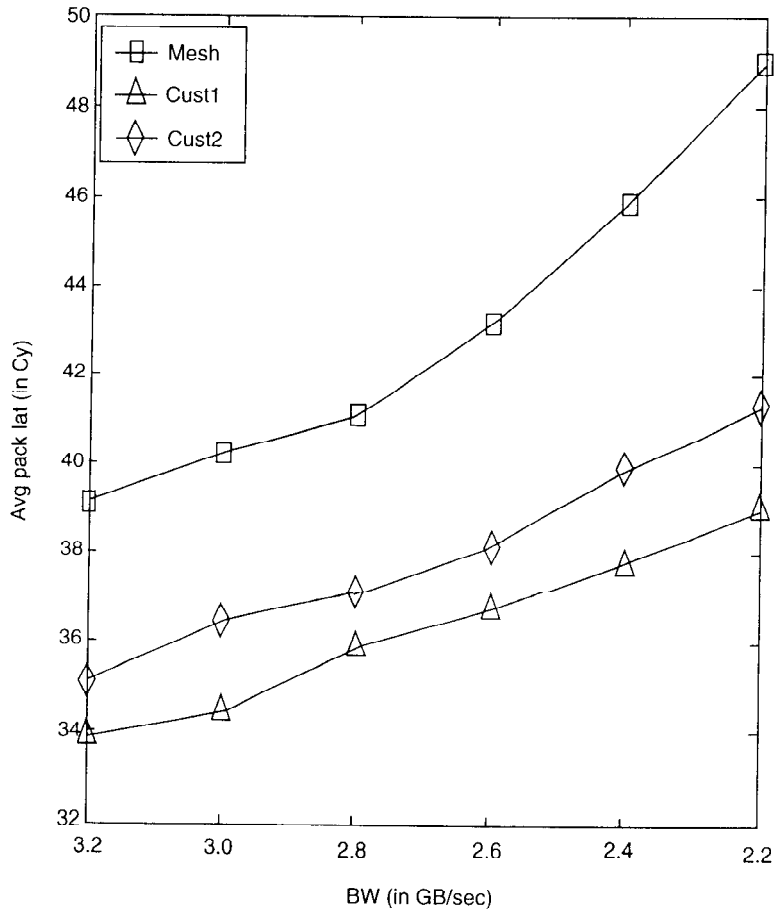



**FIGURE 95.13** Average packet latency as a function of the link bandwidth.

bandwidth decreases. Also, custom NoCs have better link utilization: around 1.5 times the link utilization of a mesh topology.

Area, power, and performance optimizations by means of custom NoCs turn out to be more difficult for MPEG4 than other applications such as Video Object Plane Decoders and Multi-Window Displayer [41].

## 95.6 Conclusions

This chapter describes the motivation for packet-switched networks as communication paradigm for deep submicron SoCs. After an overview of NoC proposals from the open literature, this chapter goes into the details of NoC architectural components (switch, network interface and point-to-point links), introducing the *Xpipes* library of composable soft macros as a case study. Finally, the challenging issue of heterogeneous NoC design is addressed, showing an example NoC synthesis flow and detailing area, power, and performance metrics of customized application-specific NoC architectures with respect to regular mesh topologies. The chapter aims at highlighting the main guiding principles and open issues for NoC design on gigascale multi-processor SoCs.

## Acknowledgement

This work was supported in part by MARCO/DARPA Gigascale Silicon Research Center.

## References

[1] Boekhorst, F., *Ambient Intelligence, the Next Paradigm for Consumer Electronics: How will it Affect Silicon?,* ISSCC 2002, Vol. 1, Feb. 2002, pp. 28–31.

[2] Benini, L. and G. De Micheli, *Networks On Chips: a New SoC Paradigm,* IEEE Computer, Vol. 35, Issue 1, pp. 70–78, 2002.

[3] Kumar, S., A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Oeberg, K. Tiensyrja, and A.Hemani, *A Network on Chip Architecture and Design Methodology,* IEEE Symposium on VLSI ISVLSI02, April 2002, pp. 105–112.

[4] Benini, L., D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, *SystemC Cosimulation and Emulation of Multiprocessor SoC Designs,* IEEE Computer, Vol. 36, Issue 4, pp. 53–59, 2003.

[5] Duato, J., S. Yalamanchili, and L. Ni, *Interconnection Networks: an Engineering Approach,* IEEE Computer Society Press, Silver Spring, MD, 1997.

[6] Wielage, P. and K. Goossens, *Networks on Silicon: Blessing or Nightmare?,* Proceedings of the Euromicro Symposium on Digital System Design DSD02, Sept. 2002, pp. 196–200.

[7] Dally, W.J. and B. Towles, *Route Packets, not Wires: On-Chip Interconnection Networks,* Design and Automation Conference DAC01, June 2001, pp. 684–689.

[8] Blume, H., H. Huebert, H.T. Feldkaemper, and T.G. Noll, *Model-based Exploration of the Design Space for Heterogeneous Systems on Chip,* IEEE Conference On Application-Specific Systems, Architectures and Processors ASAP02, pp. 29–40, 2002.

[9] Nugent, S., D.S. Wills, and J.D. Meindl, *A Hierarchical Block-Based Modeling Methodology for SoC in GENESYS.* IEEE ASIC/SOC Conference, Sept. 2002, pp. 239–243.

[10] Gerin, P., S. Yoo, G. Nicolescu, and A.A. Jerraya, *Scalable and Flexible Cosimulation of SoC Designs with Heterogeneous Multi-Processor Target Architecture,* Proceedings of the ASP-DAC 2001, Jan./Feb. 2001, pp. 63–68.

[11] Paulin, P.G., C. Pilkington, and E. Bensoudane, *StepNP: a System-level Exploration Platform for Network Processors,* IEEE Design and Test of Computers, Vol. 19, Issue 6, pp. 17–26, Nov–Dec. 2002.

[12] Guerrier, P. and A. Greiner, *A Generic Architecture for On-Chip Packet Switched Interconnections,* Design, Automation and Testing in Europe DATE00, March 2000, pp. 250–256.

[13] Lee, S.J. et al., *An 800 MHz Star-Connected On-Chip Network for Application to Systems on a Chip,* ISSCC03, Vol. 1, pp. 468–469, Feb. 2003.

[14] Yamauchi, H. et al., *A 0.8 W HDTV Video Processor with Simultaneous Decoding of Two MPEG2 MP@HL Streams and Capable of 30 Frames/s Reverse Playback,* ISSCC02, Vol.1, Feb. 2002, pp. 473–474.

[15] Dall'Osso, M., G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, *Xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs,* ICCD03, pp. 536–539. Oct. 2003.

[16] ITRS 2001, Http://public.itrs.net/Files/2001ITRS/Home.htm

[17] Bertozzi, D., L. Benini, and G. De Micheli, *Energy-Reliability Trade-off for NoCs,* in *Networks on Chip,* Jantsch, A. and Tenhunen, H. Eds., Kluwer, Dordrecht, 2003, pp. 107–129.

[18] Zhang, H., V. George, and J.M. Rabaey, *Low-Swing On-Chip Signaling Techniques: Effectiveness and Robustness,* IEEE Transactions on VLSI Systems, Vol. 8, Issue 3, pp. 264–272, 2000.

[19] Xu, J. and W. Wolf, *Wave Pipelining for Application-Specific Networks-on-Chips,* CASES02, Oct. 2002, pp. 198–201.

[20] Goossens, K., J. Dielissen, J. van Meerbergen, P. Poplavko, A. Radulescu, E. Rijpkema, E. Waterlander, and P. Wielage, *Guaranteeing the Quality of Services in Networks on Chip,* in *Networks on Chip,* Jantsch, A. and Tenhunen, H. Eds., Kluwer, Dordrecht, 2003, pp. 61–82.

[21] ITRS 1999, Http://public.itrs.net/files/1999_SIA_Roadmap/

[22] Jantsch, A. and H. Tenhunen, *Will Networks on Chip Close the Productivity Gap?,* in *Networks on Chip,* Jantsch, A. and Tenhunen, H. Eds., Kluwer, Dordrecht, 2003, pp. 3–18.

[23] VSI Alliance, Virtual Component Interface Standard, 2000.

[24] OCP International Partnership, Open Core Protocol Specification, 2001.

[25] Wingard, D., *MicroNetwork-based Integration for SoCs,* Design Automation Conference DAC01, June, 2001, pp. 673–677.

[26] Flynn, D., *AMBA: Enabling Reusable on-chip Designs,* IEEE Micro, Vol. 17, Issue 4, pp. 20–27, 1997.

[27] Zhang, H. et al., *A 1V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing,* IEEE Journal of Solid-State Circuits, Vol. 35, Issue 11, pp. 1697–1704, 2000.

[28] Culler, D., J.P. Singh, and A. Gupta, *Parallel Computer Architecture, a Hardware/Software Approach,* Morgan Kaufmann, Los Altos, CA, 1999.

[29] Compton, K. and S. Hauck, Reconfigurable Computing: a Survery of System and Software, ACM Computing Surveys, Vol. 34, Issue 2, pp. 171–210, 2002.

[30] Tessier, R. and W. Burleson, *Reconfigurable Computing and Digital Signal Processing: a survey,* Journal of VLSI Signal Processing, Vol. 28, Issue 3, pp. 7–27, 2001.

[31] Walrand, J. and P. Varaja, *High Performance Communication Networks,* Morgan Kaufmann, San Francisco, 2000.

[32] Liu, D. et al., *SoCBUS: The Solution of High Communication Bandwidth on Chip and Short TTM,* invited paper in Real Time and Embedded Computing Conference, Sept. 2002.

[33] Ishiwata, S. et al., *A Single Chip MPEG-2 Codec Based on Customizable Media Embedded Processor,* IEEE Journal of Solid-State Circuits, Vol. 38, Issue 3, pp. 530–540, 2003.

[34] Rijpkema, E., K. Goossens, A. Radulescu, J. van Meerbergen, P. Wielage, and E. Waterlander, *Trade Offs in the Design of a Router with both Guaranteed and Best-Effort Services for Networks on Chip,* Design Automation and Test in Europe DATE03, March 2003, pp. 350–355.

[35] Dally, W.J. and S. Lacy, *VLSI Architecture: Past, Present and Future,* Conference on Advanced Research in VLSI, 1999, pp. 232–241.

[36] Saastamoinen, I., D. Siguenza-Tortosa, and J. Nurmi, *Interconnect IP Node for Future Systems-on-Chip Designs,* IEEE Workshop on Electronic Design, Test and Applications, Jan., 2002, pp. 116–120.

[37] Carloni, L.P., K.L. McMillan, and A.L. Sangiovanni-Vincentelli, *Theory of Latency-Insensitive Design,* IEEE Transactions on CAD of ICs and Systems, Vol. 20, Issue 9, pp. 1059–1076, 2001.

[38] Scheffer, L., *Methodologies and Tools for Pipelined On Chip Interconnects,* International Conference On Computer Design, 2002, pp. 152–157.

[39] Glaskowsky, P., *Pentium 4 (partially) previewed,* Microprocessor Report, Vol. 14, no. 8, pp. 10–13, 2000.

[40] Van der Tol, E.B. and E.G.T. Jaspers, *Mapping of MPEG4 Decoding on a Flexible Architecture Platform,* SPIE 2002, January 2002, pp. 1–13.

[41] Jalabert, A., Murali, S., Benini, L., De Micheli, G., *XpipesCompiler: a Tool for Instantiating Application Specific Networks on Chip,* DATE 2004, pp. 884–889, 2004.

[42] Agarwal, V., M.S. Hrishikesh, S.W. Keckler, and D. Burger, *Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures,* in Proceedings 27th Annual International Symposium on Computer Architecture, June 2000, pp. 248–250.