# Network-on-chip architectures and design methods

**Article** *in* IEE Proceedings - Computers and Digital Techniques · April 2005

DOI: 10.1049/ip-cdt:20045100 · Source: IEEE Xplore



**2 authors:**

Luca Benini
University of Bologna
**1,162** PUBLICATIONS **42,340** CITATIONS

SEE PROFILE

Davide Bertozzi
University of Ferrara
**173** PUBLICATIONS **4,829** CITATIONS

SEE PROFILE

# Network-on-chip architectures and design methods

L. Benini and D. Bertozzi

**Abstract:** Performance and power of gigascale systems-on-chip (SoCs) is increasingly communication-dominated. Designers have to accommodate the communication needs of an increasing number of integrated cores while preserving overall system performance under tight power budgets. State-of-the-art SoC communication architectures start facing scalability as well as modularity limitations, and more advanced bus specifications are emerging to deal with these issues at the expense of silicon area and complexity. Communication architecture evolutions mainly regard bus protocols (to better exploit available bandwidth) and bus topologies (to increase bandwidth). In the long run, more aggressive solutions are needed to overcome the scalability limitation, and networks-on-chip (NoCs) are currently viewed as a 'revolutionary' approach to provide a scalable, high performance and robust infrastructure for on-chip communication. The paper aims at surveying the evolution of the field, moving from SoC buses to forward-looking NoC research prototypes. The elements of continuity, as well as the key differences, will be captured, in an effort to extract general guiding principles in a fast-evolving domain.

## 1 Introduction

Increasing integration densities made possible by shrinking device geometries will have to be fully exploited to meet the computational requirements of applications in domains such as multimedia processing, automotive, ambient intelligence For instance, the computational load of typical ambient intelligence tasks will be 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. These workloads will have to be delivered with tightly constrained power levels (from a few watts for wall-plugged appliances, to a few milliwatts for portable and wearable devices), affordable cost and high reliability [1]. System architecture and design technology must adapt to the critical challenges posed by both the large scale of integration and the small features of elementary devices.

To tackle the application and integration complexity challenges and the ensuing design productivity gap, SoCs are and will increasingly be designed by reusing large-scale programmable components, such as microprocessors, microcontrollers and media processors, as well as large embedded memory macros and numerous standard peripherals and specialised coprocessors. Design methodologies have to support component reuse in a plug-and-play fashion in order to be effective. In this reuse-dominated context, there is little doubt of the fact that the most critical factor in system integration will be the scalability of the communication fabric among components. This conclusion is further strengthened if we focus on the 'challenges of the small' posed by the unrelenting pace of scaling. Whereas computation and storage power-delay products (i.e. energy) benefit from device scaling (smaller gates, smaller memory cells), the energy for global communication does not scale down, hence, propagation delays on global wires will greatly exceed the clock period, and the power consumed to drive wires will dominate the power breakdown. Moreover, estimating delays accurately will become increasingly harder, as wire geometries may be determined late in the design flow. Electrical noise due to crosstalk, delay variations and synchronisation failures will be likely to produce bit upsets. Thus, the transmission of digital values on wires will be slow, power-hungry and inherently unreliable.

Network-on-chips (NoCs) are viewed by many researchers and designers as a response to the 'interconnect showstopper'. The basic premise of the network-on-chip revolution is fundamentally simple; the on-chip interconnect fabric should be designed using the same guiding principles that drive the development of macroscopic communication networks, which have demonstrated sustainable scalability, exponentially improving performance, remarkable robustness and reliability over many years of rapid evolution. The NoC literature has flourished in the last three years, with many strong contributions on development, analysis and implementation. This work does not attempt a complete overview, but it aims at providing a survey on the evolution of the field, moving from state-of-the-art communication fabrics (SoC buses), to forward-looking NoC research prototypes. We will underline the many elements of continuity as well as the key differences between SoC buses and NoCs in an effort to extract some general guiding principles in a very dynamic landscape.

## 2 On-chip buses

On-chip buses have originally been architected to mimic their off-chip counterparts, relying on the analogy between building a board with commodity components and building a system-on-chip with IP cores. Ultimately, buses rely on shared communication channels and on an arbitration mechanism which is in charge of serialising bus access requests (time division multiplexing). This widely adopted



*IEE Proceedings* online no. 20045100

doi: 10.1049/ip-cdt:20045100

Paper first received 29th July and in revised form 3rd November 2004

The authors are with the Dip. di Elettronica, Informatica e Sistemistica, Università di Bologna, Bologna, 40136, Italy

E-mail: lbenini@deis.unibo.it

solution obviously suffers from power and performance scalability limitations, but it has the advantage of low complexity and reduced area for the interface, communication and control logic.

Besides scalability, another limitation of early on-chip buses is poor decoupling between core interfaces and bus protocols, which greatly weakens modularity and composability of complex designs. To better understand this issue we can use the example of LAN/WAN interfaces in traditional computer networks, where the access protocol to the network is completely standardised (TCP/IP) and independent from the physical implementation (e.g. a shared medium, as in wireless networks, or complex multistage networks on cable or fibre). In order to test the level of decoupling between interconnect access protocol and core interfaces, a simple conceptual test can be performed, using a communication initiator (also called 'master') and a target (also called 'slave'). If master and slave can be connected directly to each other (i.e. with a point-to-point connection), then obviously the topology and internal protocol of the interconnect are completely decoupled from the core interface. We call these network interfaces PP (point-to-point). PP is a very desirable property from the design integration viewpoint, because it completely decouples communication fabric design from core design.

On-chip buses have evolved in an effort to address the above-mentioned limitations. We can distinguish three main directions of evolution: (i) enhancements in the parallelism and efficiency of the bus access protocol, which help in fully exploiting the bandwidth of the available interconnect resources; (ii) enhancements in topology, to increase the available interconnect bandwidth; and (iii) redefinition and standardisation of PP target and initiator interfaces. We will follow these trends with the help of a case study, namely AMBA, which is probably the most widely deployed on-chip communication protocol.

### 2.1 Tracking the evolutionary path: the AMBA bus

AMBA (Advanced Micro-Controller Bus Architecture) is a bus standard which was originally conceived by ARM to support communication among ARM processor cores [2]. The AMBA specification provides standard bus protocols for connecting on-chip components, custom logic and specialised functions.

AMBA defines a segmented bus architecture, where bus segments are connected with each other via a bridge that buffers data and drives the control signals across segments. A system bus is defined, which provides a high-speed, high-bandwidth communication channel between embedded processors and high-performance peripherals. Two system buses are actually specified: the AMBA high-speed-bus (AHB) and the advanced system bus (ASB).

Moreover, a low-performance and low-power peripheral bus (called advanced peripheral bus, APB) is specified, which accommodates communication with general purpose peripherals and is connected to the system bus via a bridge, acting as the only APB master. The overall AMBA architecture is illustrated in Fig. 1.

Even though AMBA defines three bus protocols, we will focus only on the most advanced one, namely AMBA AHB. The main features of AMBA AHB can be summarised as follows:

- *Non-tristate implementation*. AMBA AHB implements a separate read and write data bus in order to avoid the use of tristate drivers. In particular, master and slave signals are multiplexed onto the shared communication resources (read and write data buses, address bus, control signals).
- *Support for multiple initiators*. A control logic block, called 'arbiter', ensures that only one bus master is active on the bus and also that when no masters are requesting the bus a default master is granted. A simple request-grant mechanism is implemented between the arbiter and each bus master.
- *Pipelined and burst transfers*. Address and data phases of a transfer occur during different clock periods. In fact, the address phase of any transfer occurs during the data phase of the previous transfer. Overlapping of address and data enables full exploitation of bus bandwidth if slaves are fast enough to respond in a single clock cycle.
- *Wide data bus configurations*. Support for high-bandwidth data-intensive applications is provided using wide on-chip memories. System buses support 32, 64 and 128-bit data-bus implementations with a 32-bit address bus, as well as smaller byte and half-word designs.

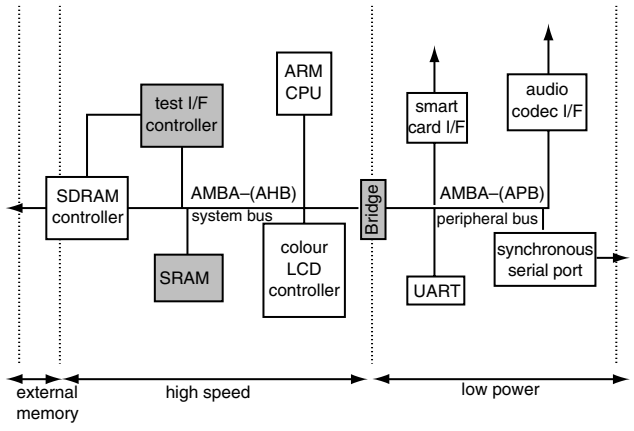


**Fig. 1** *AMBA bus architecture*

In a normal bus transaction, the arbiter grants the bus to the master until the transfer completes and the bus can then be handed over to another master. However, in order to avoid excessive arbitration latencies, the arbiter can break up a burst. In that case, the master must re-arbitrate for the bus in order to complete the remaining data transfers.

A basic AHB transfer consists of four clock cycles. During the first one, the request signal is asserted, and in the best case at the end of the second cycle a grant signal from the arbiter can be sampled by the master. Then, address and control signals are asserted for slave sampling on the next rising edge, and during the last cycle the data phase is carried out (read data bus driven or information on the write data bus sampled). A slave may insert wait states into any transfer, thus extending the data phase, and a ready signal is available for this purpose.

Four-, eight- and sixteen-beat bursts are defined in the AMBA AHB protocol, as well as undefined-length bursts. During a burst transfer, the arbiter rearbitrates the bus when the penultimate address has been sampled, so that the asserted grant signal can be sampled by the relative master at the same point where the last address of the burst is sampled. This makes bus master handover at the end of a burst transfer very efficient.

For long transactions, the slave can decide to split the operation, warning the arbiter that the master should not be granted access to the bus until the slave indicates it is ready to complete the transfer. This transfer splitting mechanism is supported by all advanced on-chip interconnects, since it

prevents high latency slaves from keeping the bus busy without performing any actual transfer of data.

As a result, split transfers can significantly improve bus efficiency, i.e. reduce the number of bus busy cycles used just for control (e.g. protocol handshake) and not for actual data transfers. Advanced arbitration features are required in order to support split transfers, as well as more complex master and slave interfaces.

The main limitations of the AHB protocol are: (i) no complete support for multiple outstanding transactions and out-of-order completion, which greatly limit bandwidth in case of slow slaves (if a slave is not ready to respond, no other transaction can bypass the blocked one, and the bus is unused during the wait cycles); (ii) no PP interface definition (for instance, initiators have to drive directly the arbitration request signals, hence they are directly exposed to the interconnect-specific time division multiplexing protocol); and (iii) limited intrinsic scalability caused by the presence of shared single master-to-slave and slave-to-master channels.

## 2.2 AMBA evolutions

To address the limitations outlined in the previous Section, advanced specifications of the AMBA bus have been proposed, featuring increased performance and better link utilisation. In particular, the multilayer AHB and the AMBA AXI interconnect schemes will be briefly reviewed in the following Subsections. Multilayer AHB can be seen as an evolution of bus topology while keeping the AHB protocol unchanged. In contrast, AMBA AXI represents a significant advancement of the protocol. It should be observed that all these interconnect performance improvement can be achieved at the expense of silicon area and complexity.

*2.2.1 Multilayer AHB:* The multilayer AHB specification aims at increasing the overall bus bandwidth and providing a more flexible interconnect architecture with respect to AMBA AHB. This is achieved by using a more complex interconnection matrix (also called a crossbar) which enables parallel access paths between multiple masters and slaves in a system [3].

Therefore, the multilayer bus architecture allows the interconnection of unmodified standard AHB master and slave modules with an increased available bus bandwidth. The resulting architecture becomes very simple and flexible; each AHB layer only has one master and no arbitration and master-to-slave muxing is needed. Moreover, the interconnect protocol implemented in these layers can be very simple; it does not have to support request and grant, nor retry or split transactions.

The additional hardware needed for this architecture with respect to the AHB is a multiplexer to connect the multiple masters to the peripherals and some arbitration is also required when more than one master tries to access the same slave simultaneously.

Figure 2 shows a schematic view of the multilayer concept. The interconnect matrix contains a decoding stage for every layer in order to determine which slave is required during the transfer. The multiplexer is used to route the request from the specific layer to the desired slave.

The arbitration protocol decides the sequence of accesses of layers to slaves based on a priority assignment. The layer with lowest priority has to wait for the slave to be freed by higher priority layers. Different arbitration schemes can be used, and every slave port has its own arbitration. Input layers can be served in a round-robin fashion, changing every transfer or every burst transaction, or based on a fixed

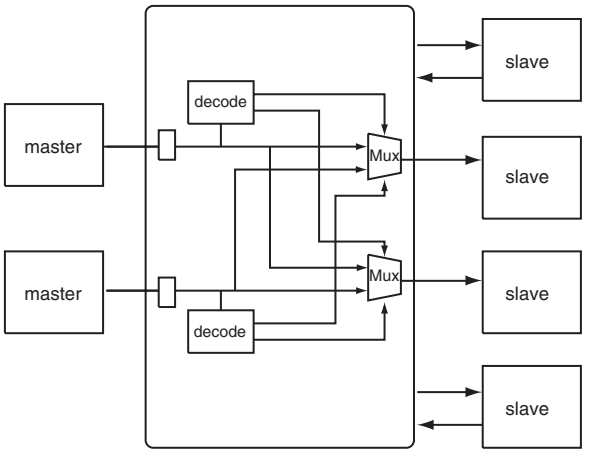


**Fig. 2** *Schematic view of the multilayer AHB interconnect*

priority scheme. It is also interesting to outline the capability of this topology to support multiport slaves. Some devices, such as SDRAM controllers, work much more efficiently when processing transfers from different layers in parallel.

The number of input/output ports on the interconnect matrix is completely flexible and can be adapted to suit to system requirements. However, as the number of masters and slaves in the system increases, the complexity of the crossbar interconnect rapidly becomes unmanageable. In essence, while a shared bus has limited scalability in terms of available bandwidth, crossbars do not scale well in hardware complexity (which impacts silicon area, cycle time, and power). To limit crossbar complexity blowup, some optimisation techniques have to be used, such as defining multiple masters on a single layer, multiple slaves appearing as a single slave to the interconnect matrix or defining local slaves to a particular layer.

*2.2.2 AMBA AXI:* AXI is the most recent evolution of the AMBA interface. It significantly enhances protocol performance and it also includes optional extensions for low-power operation [4]. This high-performance protocol provides flexibility in the implementation of interconnect architectures while still keeping backward-compatibility with existing AHB and APB interfaces.

AMBA AXI is a fully PP connection. It decouples masters and slaves from the underlying interconnect, by defining only master interfaces and symmetric slave interfaces. This approach, besides allowing backward compatibility and interconnect topology independence, has the advantage of simplifying the handshake logic of attached devices, which only need to manage a point-to-point link.

To provide higher parallelism, four different logical monodirectional channels are provided in AXI interfaces; an address channel, a read channel, a write channel and a write response channel. Activity on different channels is mostly asynchronous (e.g. data for a write can be pushed to the write channel before or after the write address is issued to the address channel), and can be parallelised, allowing multiple outstanding read and write requests, with out-of-order completion.

Figure 3*a* shows how a read transaction uses the read address and read data channels. The write operation over the write address and write data channels is presented in Fig. 3*b*. Data is transferred from the master to the slave using a write data channel, and it is transferred from the slave to the master using a read data channel. In write transactions,

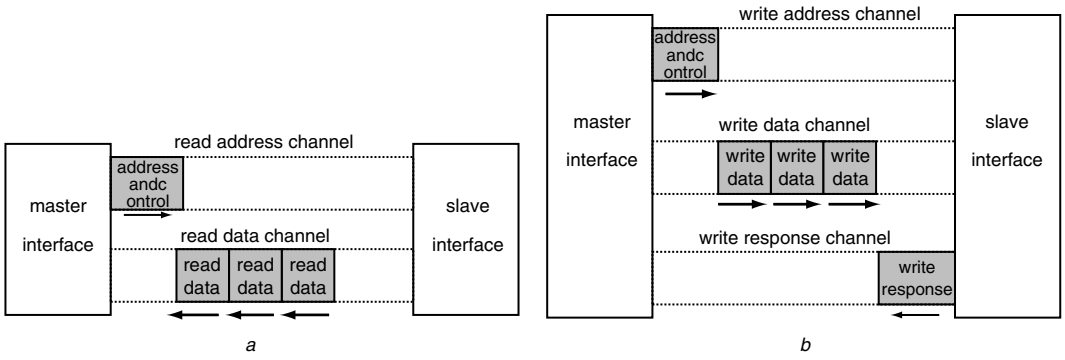


**Fig. 3** *Architecture of transfers*

*a* Read operation
*b* Write operation

where all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction. The rationale of this split-channel implementation is based upon the observation that usually the required bandwidth for addresses is much lower than that for data (e.g. a burst requires a single address but maybe four or eight data transfers). Thus, it might be possible to allocate more interconnect bandwidth to the data bus than the address bus.

The mapping of channels, as visible by the interfaces, to interconnect resources is decided by the interconnect designer; single resources might be shared by all channels of a certain type in the system, or a variable amount of dedicated wires may be available, up to a full crossbar.

Concluding this Section, we observe that on-chip buses have come a long way. On one hand, PP protocols act now fully as network interfaces, while, on the other hand, multilayer topologies can provide much higher bandwidth than a single shared channel. We used AMBA as a case study for these trends, but the landscape of evolutionary interconnects is very diverse, and many alternatives do exist [5, 6].

Still, these evolutionary approaches do not address in full the fundamental scalability limitation of any single-hop interconnect. Networks-on-chip, as described in Section 4, aim precisely at providing sustainable scalability by making it possible to define multihop topologies and providing efficient support to switching, routing and flow control.

## 3 Quantitative analysis

This Section focuses on providing some quantitative evidence of the performance benefits provided by enhanced protocols and high-bandwidth topologies. At first, scalability of evolving interconnect fabric protocols is assessed. Then, we will focus on speed enhancements due to multichannel topologies.

### 3.1 Protocol efficiency

SystemC models of AMBA AHB and AMBA AXI (provided within the Synopsys CoCentric/Designware® [7] suites) are used within the framework of the MPARM simulation platform [8–10].

The simulated on-chip multiprocessor consists of a configurable number of ARM cores attached to the system interconnect. Traffic workload and pattern can easily be tuned by running different benchmark code on the cores, by scaling the number of system processors, or by changing the amount of processor cache, which leads to different amounts of cache refills.

An AMBA AHB link and a more advanced, but also more expensive, AMBA AXI interconnect with shared bus topology are tested under heavy load. Figure 4 shows an example of the efficiency improvements made possible by advanced interconnects in the test case of slave devices having two wait states, with three system processors and four-beat burst transfers. AMBA AHB has to pay two cycles of penalty per transferred datum. AMBA AXI is capable of interleaving transfers by sharing data channel ownership in time. Under conditions of peak load, when transactions always overlap, AMBA AHB is limited to a 33% efficiency (transferred words over elapsed clock cycles), while AMBA AXI can theoretically reach a 100% throughput.

In order to assess interconnect scalability, a benchmark is independently but concurrently run on every system processor performing accesses to its private slave (involving bus transactions). This means that, while producing real functional traffic patterns, the test setup was not constrained by bottlenecks due to shared slave devices. Private memories are assumed to introduce one wait state before responses.

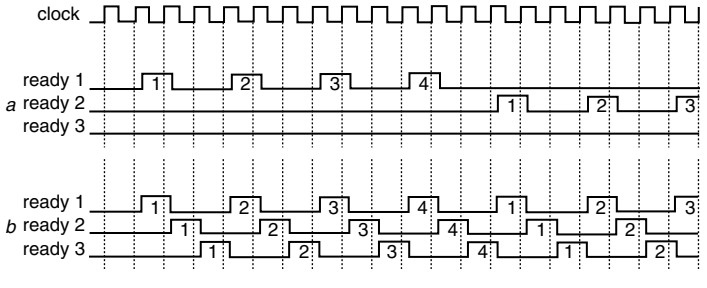


**Fig. 4** *Concept waveforms showing burst interleaving for AMBA AHB and AXI interconnects*

*a* AMBA AHB
*b* AMBA AXI

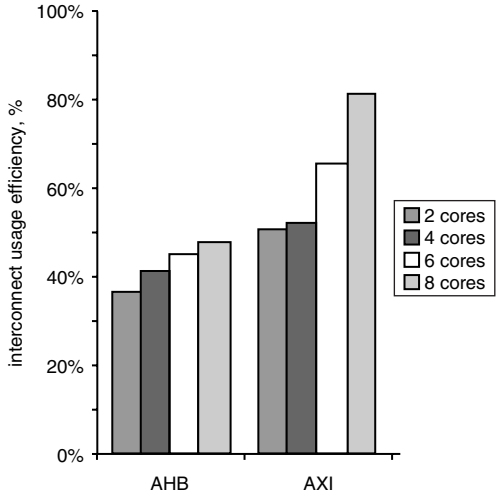


**Fig. 5** *Execution times with 256B caches*

Scalability properties of the system interconnects can be observed in Fig. 5, reporting the execution time variation with an increasing number of cores attached to a single shared interconnect under heavy traffic load. Core caches are kept very small (256 bytes) in order to cause many cache misses and therefore significant levels of interconnect congestion. Execution times are normalised against those for a two-processor system, trying to isolate the scalability factor alone. The heavy bus congestion case is considered here because the same analysis performed under light traffic conditions (e.g. with 1 kB caches) shows that both interconnects perform very well, with AHB showing a moderate performance decrease of 6% when moving from two to eight running processors.

With 256 B caches, the resulting execution times, as Fig. 5 shows, get 77% worse for AMBA AHB when moving from two to eight cores, while AXI manages to stay within 12% and 15%. The reason behind the behaviour pointed out in Fig. 5 is that under heavy traffic load and with many processors, interconnect saturation takes place. This is clearly indicated in Fig. 6, which reports the fraction of cycles during which some transaction was pending on the bus with respect to total execution time.

In such a congested environment, as Fig. 7 shows, AMBA AXI can achieve transfer efficiencies (defined as data actually moved over bus contention time) of up to 81%,

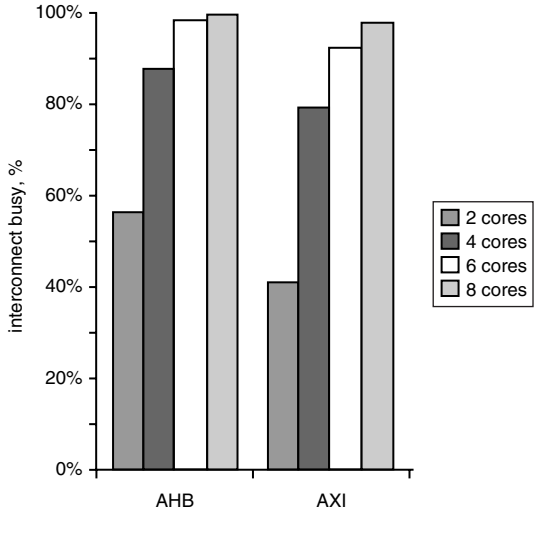


**Fig. 6** *Bus busy time with 256B caches*

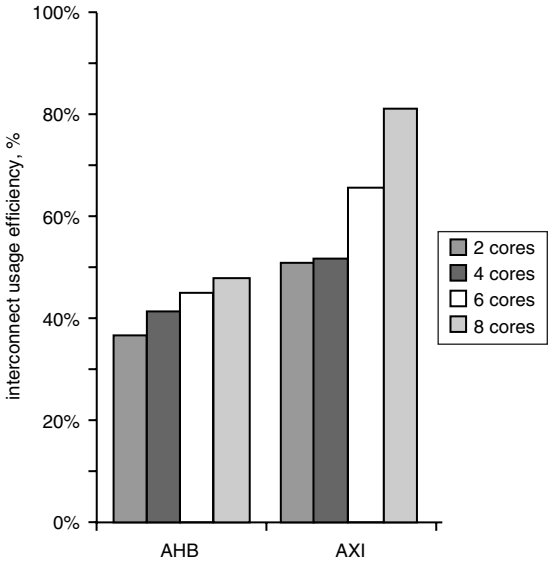


**Fig. 7** *Bus usage efficiency with 256B caches*

while AMBA AHB reaches 47% only; near to its maximum theoretical efficiency of 50% (one wait state per data word). These plots stress the impact that comparatively low-area-overhead optimisations can sometimes have in complex systems.

It must be pointed out, however, that protocol improvements alone cannot overcome the intrinsic performance limitations due to the shared nature of the interconnect resources. While protocol features can push the saturation boundary further, and get near to a 100% efficiency, traffic loads taking advantage of more parallel topologies will always exist. The charts reported here already show some traces of saturation even for the most advanced protocols.

### 3.2 Multichannel topologies

Topology enhancement can provide additional steam to bandwidth-saturated buses. To illustrate this point with some experimental evidence, we performed a test based on functional simulation of a complete multiprocessor architecture with an AMBA AHB compliant interconnect and eight ARM7 cores. We run two applications on the platform, namely independent matrix multiply, and independent matrix multiply with semaphore synchronisation upon completion. In the first application, each processor performs matrix multiplication and it is completely independent from the other processors. Matrices are stored in different memories, which are connected as slaves to the system interconnect, hence there is no contention among processors for the same memory slave. However, execution time is heavily impacted by contention for the shared communication resource. In the second application, data processing is exactly the same as for the first application, but processors synchronise after computing every element of the product matrix using a counting semaphore which is contained in a dedicated slave device.

We ran the two applications with different interconnects, namely a shared bus, a multilayer implementation based on a full crossbar and a bridged solution that splits the bus in two segments. Results are summarised in Fig. 8. Data transfers have no destination conflicts, hence, very significant speedups can be achieved by advanced topologies. Notice that for the independent matrix multiply benchmark, a speedup of more than a factor of two is achieved. The bridged bus gives lower speedup. This is expected

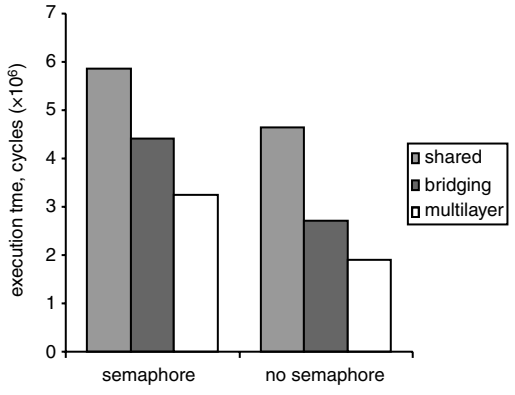


**Fig. 8** *Application execution time with different topologies*

because splitting the bus into two segments gives a maximum theoretical throughput enhancement of two, in the case of no traffic between segments, while theoretical crossbar speedup is $N$ (where $N$ is the number of independent channels).

Speedups are inferior for the synchronised application. This demonstrates that the application-level speedup provided by a given topology strongly depends on the nature of the traffic generated by applications. In this case, there is a traffic bottleneck created by the single counting semaphore in shared memory; all processors will contend for semaphore access, and this application-level contention significantly impacts both execution time and speedups. The bottleneck is even more evident for the bridged solution, because many transactions have to cross the bridge (namely, all semaphores accessed by processors on the opposite side of the bridge).

As a conclusive note, it is important to stress that these application-level speedups are achieved at the price of significantly increased cost in crossbar logic. In fact, crossbar complexity scales quadratically with $N$, and a full crossbar would not be usable in practice for $N$ much larger than ten. The bridged solution, which is the most commonly used in today's designs, has even more limited scalability, and it is highly sensitive to traffic patterns; in fact it can perform even worse than a single shared bus if many transactions have to traverse the bridge. Hence, this solution should be evaluated very carefully for possible traffic mismatches.

## 4 Packet-switched interconnection networks

The previous Sections have described evolutionary communication architectures. We now focus on a more revolutionary approach to on-chip communication, known as 'network-on-chip (NoC)' [11, 12]. NoCs are packet-switched, multihop interconnection networks integrated onto a single chip. Cores access the network by means of PP interfaces, and have their packets forwarded to destinations through a number of hops. NoCs differ from wide area networks in their local proximity and because they exhibit less nondeterminism. Local, high-performance networks, such as those developed for large-scale multiprocessors, have similar requirements and constraints. However, some distinctive features, such as energy constraints and design-time specialisation, are unique to SoC networks.

Given the degrees of freedom of a multihop architecture, topology selection for NoCs is a critical design issue. The main issue is how efficiently communication requirements of an application can be mapped onto a certain topology, and by physical level considerations. In fact, regular topologies, such as meshes, can be designed with better control of electrical parameters and, therefore, on communication noise sources (such as crosstalk), although they might result in link underutilisation or localised congestion from an application viewpoint. On the contrary, irregular topologies have to deal with more complex physical design issues but are more suitable to implement customised, domain-specific communication architectures.

The scalable and modular nature of NoCs and their support for efficient on-chip communication potentially lead to NoC-based multicore systems characterised by high structural complexity and functional diversity. On one hand, these features need to be properly addressed by means of new design methodologies, while on the other hand more efforts have to be devoted to modelling on-chip communication architectures and integrating them into a single modelling and simulation environment combining both processing elements and communication architectures. The development of NoC architectures and their integration into a complete MPSoC design flow is the main focus of an ongoing worldwide research effort [13–15].

In this direction, Dally and Lacy sketch the architecture of a VLSI multicomputer using 2009 technology [16]. 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) [17–19]. Most proposed NoC platforms are packet-switched and exhibit regular structure. An example is a mesh interconnection, which can rely on a simple layout and the switch independence on the network size, such as the NOSTRUM network [20]. The Linkoeping SoCBUS [21] 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 behaviour but restricts routing flexibility for the following traffic. The scalable programmable integrated network (SPIN) described in [22] is another regular, fat-tree-based network architecture. It adopts cut-through switching to reduce message latency.

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 optimised 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. Many recent heterogeneous SoC implementations are still based on shared buses (such as the single-chip MPEG-2 codec reported in [23].) The Aethereal NoC design framework presented in [13] 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 applications with soft or unspecified constraints. Support for heterogeneous architectures requires highly configurable network building blocks, which can be customised at instantiation time for a specific application domain. For instance, the Proteo NoC [24] consists of a small library of predefined, parameterised components that allow the implementation of a large range of different topologies, protocols and configurations. Xpipes interconnect [25] and its synthesiser XpipesCompiler [26] push this approach to the limit, by instantiating an

application specific NoC from a library of soft macros (network interface, link and switch). The components are highly parameterisable and provide reliable and latency insensitive operation.

### 4.1 NoC architecture

Messages that have to be transmitted across the network are 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: network link, switch, and network interface.

*4.1.1 The link:* The performance of interconnects is a major concern in scaled technologies As geometries shrink, gate delay improves much faster than the delay in long wires. 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 [27]. 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.

A solution to overcome the interconnect-delay problem consists of pipelining interconnects [28]. Wires can be partitioned into segments bounded by relay stations, which have a function similar to the one of latches on a pipelined data path. Segment 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 becomes decoupled from the link delay. This requires the system to be made of modules whose behaviour 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 by functional modules that exchange data on communication channels according to a latency-insensitive protocol. This protocol ensures that functionally correct modules behave correctly independently of the channel latencies [28]. 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 [29] assumes that interconnect pipelining is the strategy of choice in its estimates of achievable clock speeds. Some industrial designs already make use of interconnect pipelining. For instance, the NETBURST microarchitecture 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 [30].

The Xpipes NoC supports pipelined links and 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. According to the link length, a certain number of clock cycles is needed by a flit to cross the interconnect. These design choices are at the basis of latency-insensitive operation of the NoC and allow the construction of an arbitrary network topology and, hence, support for heterogeneous architectures, without creating clock cycle bottleneck on long links.

The link model is equivalent to a pipelined shift register. Pipelining has been used both for data and control lines, hence also for ACK lines used by ACK flits to propagate from the destination switch back to the source one. This architecture impacts the way link-level error control is performed in order to ensure robustness against communication errors. In fact, 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, and after $N$ clock cycles (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 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.

*4.1.2 Switch architecture:* 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 characterised 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 buffer. 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 specifically addresses the limited amount of buffering resources to switches. Several approaches have been explored in this context [31]. In store-and-forward flow control, an entire packet is received and stored before being forwarded to the next switch. Virtual cut-through 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* flow-control scheme can be employed to reduce switch memory requirements with low latency communication. The first flit of a packet contains routing information, and header flit decoding enables the switches to establish the path, while subsequent flits simply follow this path in a pipelined fashion 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 utilisation.

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 requirements. 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 is transported. Therefore, contention resolution takes place during connection setup, and time-related guarantees during data transport can be given. In time division circuit switching, 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 oversizing to bound fluctuations of performance metrics.

The Aethereal NoC architecture makes use of a router that tries to combine guaranteed throughput (GT) and best effort (BE) services [13]. 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 connection; 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 fashion. 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 shown that both input queuing with wormhole flow control or virtual cut-through routing and virtual output queuing with wormhole flow control are feasible in terms of buffering cost. The BE and GT router subsystems are combined in the Aethereal router architecture. 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. Negotiations, resulting in slot allocation, can be done at compile time, and be configured deterministically at run time.

A different perspective has been taken in the design of the switch for the best effort Xpipes NoC. Figure 9 shows an example configuration with four inputs, four outputs. 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 latency-insensitive operation, the switch must have a channel buffer, implemented as a circular buffer, capable of storing $2N + M$ flits, where $N$ is the link length (expressed as number of basic repeater stages) and $M$ is a switch-architecture-related contribution (two 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 (one cycle), the ACK/NACK signal has to be propagated back ($N$ cycles) and recognised by the source switch (one cycle). During this time, other $2N + M$ flits are transmitted, but not yet ACKed. Output buffering was chosen for Xpipes switches. Flow-control signals generated by each output block are fed back to the upstream switches. The switch is pipelined (two stages) so to maximise the operating clock frequency of the switch. Error control at the link level is supported. Finally, the switch is highly parameterisable. Design parameters are: number of I/O ports, flit width, number of virtual channels, length of switch-to-switch links, size of output registers.

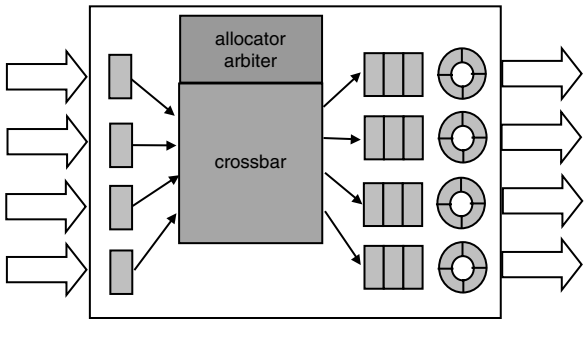


**Fig. 9** *Xpipes latency-insensitive switch*

A comparison between the hardware implementation of the Xpipes switch and the Aethereal switch is quite instructive. Both switches have been targeted to similar 130 nm technologies. A Xpipes switch with four port and 64 b flits uses 0.19 $mm^2$ of silicon area and can be clocked at 800 MHz. An Aethereal switch with five ports (32 b phits) uses 0.26 $mm^2$ of silicon area and is clocked at 500 MHz. The internal buffering in the Xpipes switch is 6 flits per output port while Aethereal is 24 phits per input port. We note that the Aethereal switch achieves better buffer density, mainly because it uses custom-designed FIFO macros, while the Xpipes switch is fully synthesised. However, Xpipes is faster. This is probably due to the QoS support in Aethereal, which impacts control complexity and ultimately cycle time.

*4.1.3 Network interface:* The network interface (NI) is entrusted with several critical tasks: (i) providing a standardised set of PP transactions to cores; (ii) efficient mapping of PP transactions into a (possibly large) set of network transactions; and (iii) interfacing with the packet-based network fabric (packet assembly, delivery and disassembly).

The first objective requires the definition of a standardised PP interface. AMBA AXI is an example of such an interface, but its definition and evolution is controlled by a single company. To avoid the captivity risks associated with proprietary standards, several core interface standardisation initiatives have been promoted. For instance, the VSIA vision [32] 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 component 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) [33]. Its main characteristic are a high degree of configurability to adapt to the core's functionality and the independence of request and response phases, thus supporting multiple outstanding requests and pipelining of transfers (VCI and OCP have recently announced a merger).

Data packetisation 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 the 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 may contain redundant checksum bits for error-detecting or error-correcting codes.

An insight in the Xpipes network interface implementation will provide an example of these concepts. The Xpipes NI provides a standardised OCP-based interface to network nodes. The NI for cores that initiate communication (initiators) need to turn OCP-compliant transactions into packets to be transmitted across the network. It represents the slave side of an OCP PP connection, and it has to build the packet header, which is then embedded into the first flit of a packet. Xpipes relies on a static routing algorithm called street sign routing. Routes are derived by the network interface by accessing a look-up table based on the destination address. Such information consists of direction bits read by each switch and indicating the output port of the switch, which flits belonging to a certain packet have to be directed to. The main datapath units in the slave interface are OCP buffer, source routing table, flit-building logic, flit buffer and link interface. The NI can be decomposed in a front-end (containing OCP buffer, routing table and flit-building logic) and a back-end, containing flit buffer and link interface. Clearly, a complete initiator NI must provide a response block, whose purpose is to manage incoming packets, reassemble data in OCP-compliant format and manage the handshaking with the core. Again, the NI response block can be decomposed in front-end and back-end.

A very critical issue in the design of advanced network interfaces is the degree of support for multiple outstanding transactions. Given the multihop nature of NoCs, packet delivery latency (even without considering congestion-related latency) can be high. For instance, even if Xpipes is highly tuned for low-latency operation, each switch inserts two clock cycles latency, and pipelined links can significantly increase latency. If every PP transaction blocks the core interface until completion, network latency may seriously impact the bandwidth available to the cores, especially if cores issue posted transactions (i.e. transactions which do not require a response phase) or can initiate multiple outstanding transaction. It is important to notice, however, that supporting multiple outstanding transactions significantly increases control complexity and buffering in the NI, and the hardware cost is justified only when interfacing with advanced cores. For instance, Xpipes supports multiple posted writes, but it does not support, in its first version, multiple outstanding reads (or nonposted writes). This design choice is motivated by the fact that most of the cores available in the Xpipes simulation environment cannot exploit multiple outstanding transactions.

Before closing this Section on NoC architecture and components, we need to mention the important issue of synchronisation. Even though it may be possible to take the simplifying assumption that an entire SoC is synchronised by a single clock, in reality there is little doubt that all large scale SoCs and their communication fabric will need to support much more flexible synchronisation schemes. In fact, the cost (in terms of area, power, design effort) of distributing a single clock at the frequency needed to provide adequate performance is already unmanageable in current technology. Even though some authors are investigating fully asynchronous communication schemes, the most likely solution for NoC synchronisation is a globally asynchronous, locally synchronous (GALS) paradigm [34, 35]. Flexible GALS synchronisation provides an additional degree of freedom for NoC optimisation. For instance, if the NoC is clocked faster than the core, very wide data transfers from cores can be serialised in time over narrow network links [36]. This approach can greatly help in reducing wiring congestion, especially for crossbars and for NoCs with reduced number of hops and switches with a large number of input and output ports.

## 5 NoC design technology

NoC architectures are pushing the evolution of traditional IC design methodologies in order to more effectively deal with functional diversity and complexity. At the application level, the key design challenge is to expose task-level parallelism and to formally capture concurrent communication in models of computation. Then, high-level concurrent tasks have to be mapped to the underlying communication and computation resources. At this level, an abstract model of the hardware architecture is usually exposed to the mapping tool, so that area and power estimates can be given in the early design stage, and different objective functions (e.g. minimisation of communication energy) can be considered to evaluate the feasibility of alternative mappings.

For NoC-based MPSoCs, a critical step in communication mapping is the network topology selection for its significant impact on overall system performance, which is increasingly communication-dominated. In this area, we can distinguish two different approaches; namely, mapping onto predefined, regular topologies with homogeneous nodes and mapping onto ad hoc, application-specific topologies with heterogeneous nodes. The first approach can leverage a large body of research from traditional parallel computing, where the key problem is how to effectively map complex parallel applications on given regular topologies (which are typically used in highly parallel large-scale multiprocessors) and it is conceptually more tractable, because it decouples topology definition and instantiation from mapping. A few early approaches to this problem in a NoC setting have recently been proposed [37].

It is important to notice, however, that 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 [38], depicted in Fig. 10*b*, where blocks are drawn roughly to scale and links represent interblock communication. First, the embedded memory (SDRAM) is

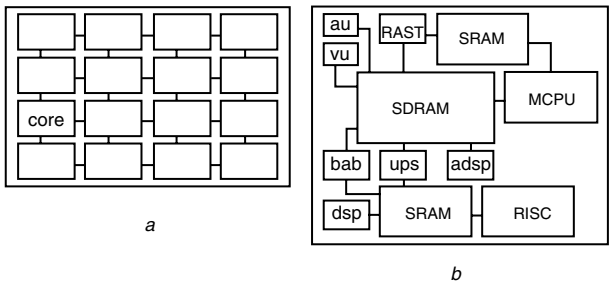


**Fig. 10** *Tile-based architecture and decoder implementation*

*a* Tile-based architecture (chip multiprocessor)
*b* MPEG4 decoder implementation (MPEG4 SoC)

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 Fig. 10*a*. Secondly, the total communication bandwidth to/from the embedded SDRAM is much larger than that required for communication among the other cores. Thirdly, many neighbouring 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 underutilising many tiles and links, or, at the opposite extreme, of achieving poor performance because of localised congestion. These factors motivate the use of an application-specific on-chip network.

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. The Xpipes NoC takes this approach. As described in the previous Section, 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 on to the underlying platform. This mapping procedure involves optimisations and trade-offs between many complex constraints, including quality of service, real-time response, power consumption, area, etc. Tools are urgently needed to explore this mapping process, and assist and automate optimisation 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 optimised for that particular NoC design, providing large savings in area, power and latency with respect to standard NoCs based on regular structures.

### 5.1 NoC synthesis case study: Xpipes

The design methodology has to partition the design problem into manageable tasks and to define the tools and practices for those tasks. In this Section we illustrate the challenges of NoC synthesis using an example NoC synthesis flow, called NetChip [39], for designing domain-specific NoCs and automating most of the complex and time-intensive design steps. NetChip provides design support for regular and custom network topologies, and therefore lends itself to the implementation of both homogeneous and heterogeneous system interconnects. NetChip assumes that the application has already been mapped onto cores by using pre-existing tools, and the resulting cores together with their communication requirements represent the inputs to the synthesis flow.

The tool-assisted design and generation of a customised NoC-based system is achieved by means of three major design activities: topology mapping, topology selection, and topology generation. NetChip leverages two tools: SUNMAP, which performs the network topology mapping and selection functions and XpipesCompiler, which performs the topology generation function. SUNMAP produces a mapping of cores onto various NoC topologies that are defined in a topology library. The mappings are optimised for the chosen design objective (such as minimising area, power or latency) and satisfy the design constraints (such as area or bandwidth constraints). SUNMAP uses floorplanning information early in the mapping process to determine the area–power estimates of a mapping and to produce feasible mappings (satisfying the design constraints). The tool supports various routing functions (dimension-ordered, minimum-path, traffic splitting across minimum paths, traffic splitting across all paths) and chooses the mapping onto the best topology from the library of available ones.

A design file describing the chosen topology is input to the XpipesCompiler, which automatically generates the SystemC description of the network components (switches, links and network interfaces) and their interconnection with the cores. A custom hand-mapped topology specification can also be accepted by the NoC synthesiser, and the network components with the selected configuration can be generated accordingly. The resulting SystemC code for the whole design can be simulated at the cycle-accurate and signal-accurate level.

The complete XpipesCompiler flow is summarised as follows. From the specification of an application, the designer (or a high-level analysis and exploration tool, like SUNMAP) 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, based on the architectural templates described in Section 4.1. The output is a SystemC hierarchical description, which includes all switches, links, 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 comparison of the effects (in terms of area, power and performance) of mapping applications on customised domain-specific NoCs and regular (e.g. mesh) NoCs.

As an example, let us focus on the MPEG4 decoder. Its core graph representation together with its communication requirements are reported in Fig. 11. The edges are annotated with the average bandwidth requirements of the cores in Mbyte/s. Customised 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 Fig. 12. In the MPEG4 design considered,

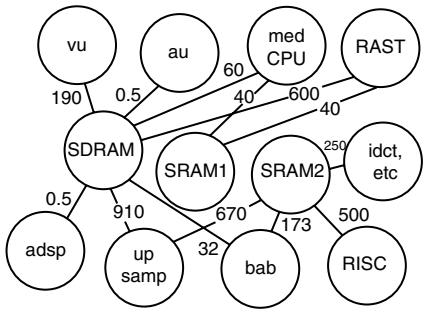


**Fig. 11** *Core graph representation of an example MPEG4 with average communication requirements*

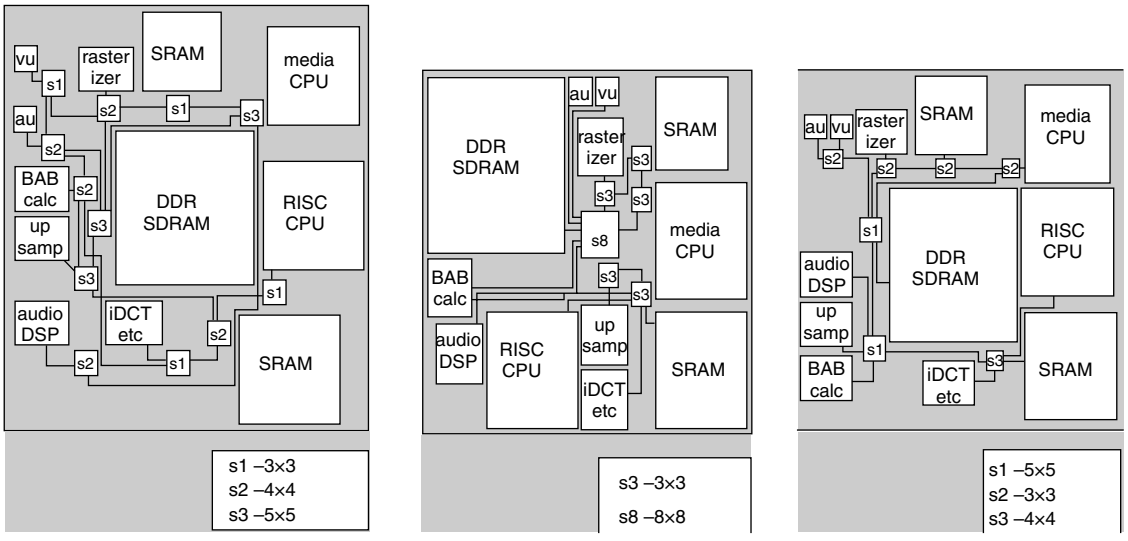


**Fig. 12** *NoC configurations for MPEG4 decoder*

**Table 1: Area and power estimates for NoC configurations**

| Instance | Area mm$^2$ | A ratio | Power mW | P ratio |
|---|---|---|---|---|
| MESH | 1.31 | 1 | 114.36 | 1 |
| COSTUM 1 | 0.86 | 1.52 | 110.66 | 1.03 |
| CUSTOM 2 | 0.71 | 1.85 | 93.66 | 1.22 |

many of the cores communicate with each other through the shared SDRAM. Therefore, a large switch is used for connecting the SDRAM with other cores (Fig. 12*b*), while smaller switches are employed for other cores. An alternate custom NoC is also considered (Fig. 12*c*); it is an optimised 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 1. The area calculations are based on analytical models of Xpipes switch area, including crossbar area, buffer and logic area. Although all cores communicate with many other cores and therefore many switches are needed, area savings for custom NoCs are significant.

The power dissipation for the NoC designs has been estimated using the analytical models proposed in [40]. These models account for the hardware complexity of the switches as well as the traffic passing through them. Power savings for the custom NoC1 are not relevant, as most of the traffic traverses the larger switches connected to the memories. As power dissipation on a switch increases nonlinearly with increase in switch size, there is more power dissipation in the switches of custom NoC1 (that has an $8 \times 8$ 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. In contrast, the NoC2 solution is much more power-efficient.

Figure 13 reports the variation of average packet latency (for 64B packets, 32 bit flits) with link bandwidth. Custom NoCs, as synthesised 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% savings in latency are achieved. Area, power and performance optimisations by means of custom

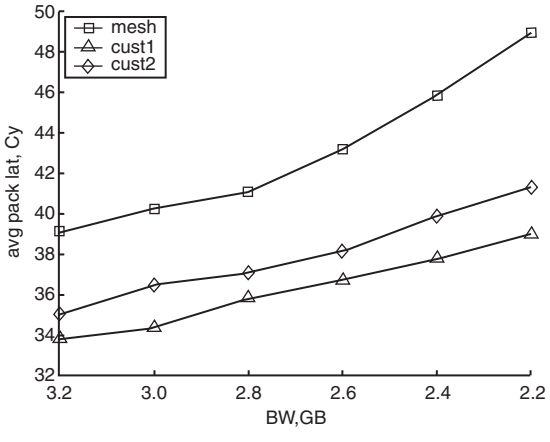


**Fig. 13** *Average packet latency as a function of the link bandwidth*

NoCs turn out to be more difficult for MPEG4 than for other applications, such as video object plane decoders and multiwindow displayer [26].

Concluding this Section, we observe that the custom NoC synthesis approach is viable and competitive only if supported by a complete and robust design flow and toolset. The Xpipes flow is the first attempt in this direction. Even though fully automated NoC synthesis enables reuse of predesigned components (i.e. the soft macros) without compromising flexibility, the quality of components synthesised starting from soft macros can be smaller than that of highly optimised custom-designed hard macros. Hence, much work has to be done, especially in the synthesis backend (RTL and logic optimisation, placement and routing), to fully demonstrate the advantages of this approach with respect to regular and homogeneous NoC architectures.

## 6 Conclusions

This paper has reviewed the guiding principles that are driving the evolution of SoC communication architectures from state-of-the-art shared buses to forward-looking NoC architectures. It shows how the large gap (in terms of design technology) between these two solutions is currently being

bridged by means of bus protocols, aiming at a better exploitation of the available bandwidth for on-chip communication, as well as bandwidth-enhancing bus topology evolutions. Finally, NoC design issues are discussed and some early research prototypes are described as case studies, pointing out the need for new design skills and methodologies in order to fully exploit the benefits of these architectures.

## 7 Acknowledgments

The authors acknowledge the contribution of a large group of coworkers which has made it possible to write this work. The group, led by Prof. G. De Micheli at Stanford University, in particular S. Murali and S. Stergiou, has made essential contributions in NoC design and design technologies. The authors also acknowledge coworkers in IMEC (F. Catthoor and P. Marchal), in the Technical University of Madrid (J. Ayala) and in Bologna (F. Poletti and F. Angiolini), which have massively contributed to the development of our NoC modelling and simulation platform. Work in Bologna is supported, in part, by a grant from STMicroelectronics.

## 8 References

1 Boekhorst, F.: 'Ambient intelligence, the next paradigm for consumer electronics: how will it affect silicon?'. Proc. ISSCC 2002, February 2002, Vol. 1, pp. 28–31
2 ARM, 'AMBA Specification,' v2.0, 1999
3 ARM, 'AMBA Multi-layer AHB overview,' 2001
4 ARM, 'AMBA AXI Protocol Specification,' 2003
5 Wodey, P., Camarroque, G., Barray, F., Hersemeule, R., and Cousin, J.P.: 'LO–TOS code generation for model checking of STBus based SoC: the STBus interconnection'. Proc. ACM and IEEE Int. Conf. on Formal Methods and Models for Co-Design, June 2003, pp. 204–213
6 Sonics, Inc.: 'Sonics μNetworks. Technical Overview,' 2002
7 Synopsys CoCentric. http://www.synopsys.com, 2004
8 Benini, L., Bertozzi, D., Bruni, D., Drago, N., Fummi, F., and Poncino, M.: 'SystemC cosimulation and emulation of multiprocessor SoC designs', *Computer*, 2003, **36**, (4), pp. 53–59
9 Poletti, F., Bertozzi, D., Bogliolo, A., and Benini, L.: 'Performance analysis of arbitration policies for SoC communication architectures', *Des. Autom. Embedded Syst.,* June/September 2003 (8), pp. 189–210
10 Loghi, M., Angiolini, F., Bertozzi, D., Benini, L., and Zafalon, R.: 'Analyzing on-chip communication in a MPSoC environment'. Proc. IEEE Design Automation and Test in Europe Conf., (DATE04), February 2004, pp. 752–757
11 Henkel, J., Wolf, W., and Chakradhar, S.: 'On-chip networks: a scalable, communication-centric embedded system design paradigm'. Proc. Int. Conf. on VLSI Design, January 2004, pp. 845–851
12 Benini, L., and De Micheli, G.: 'Networks on chips: a new SoC paradigm', *Computer*, 2002, **35**, (1), pp. 70–78
13 Rijpkema, E., Goossens, K., and Radulescu, A.: 'Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip'. Proc. Design Automation and Test in Europe, March 2003, pp. 350–355
14 Lee, K., *et al.*: 'A 51 mw 1.6 ghz on-chip network for low power heterogeneous SoC platform'. ISSCC Digest of Tech. Papers, 2004, pp. 152–154
15 Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A.: 'QNoC: QoS architecture and design process for network on chip', *J. Syst. Archit.*, 2005, **50**, pp. 105–128
16 Dally, W.J., and Lacy, S.: 'VLSI architecture: past, present and future.' Conf. on Advanced Research in VLSI, 1999, pp. 232–241
17 Culler, D., Singh, J.P., and Gupta, A.: 'Parallel computer architecture, a hardware/software approach' (Morgan Kaufmann, 1999)
18 Compton, K., and Hauck, S.: 'Reconfigurable computing: a Survey of system and software', *ACM Comput. Surv.*, 2002, **34**, (2), pp. 171–210
19 Tessier, R., and Burleson, W.: 'Reconfigurable computing and digital signal processing: a survey', *J. VLSI Signal Process.*, 2001, **28**, (3), pp. 7–27
20 Kumar, S. *et al.*: 'A network on chip architecture and design methodology'. IEEE Symp. on VLSI 2002, April 2002, pp. 105–112
21 Liu, D., *et al.*: 'Parallel computer architecture, a hardware/software approach' (Morgan Kaufmann, 1999)
22 Andriahantenaina, A., Charlery, H., Greiner, A., and Mortiez, L.: 'SPIN: a scalable, packet switched, on-chip micro-network'. Proc. Design Automation and Test in Europe, March 2003, pp. 70–73
23 Ishiwata, S., *et al.*: 'A single chip MPEG-2 codec based on customizable media embedded processor', *IEEE J. Solid-State Circuits*, 2003, **38**, (3), pp. 530–540
24 Saastamoinen, I., Siguenza-Tortosa, D., and Nurmi, J.: 'Interconnect IP node for future systems-on-chip designs'. Proc. IEEE Workshop on Electronic Design, Test and Applications, January 2002, pp. 116–120
25 Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., and Benini, L.: 'Xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs'. Proc. ICCD 2003, October 2003, pp. 536–539
26 Jalabert, A., Murali, S., Benini, L., and De Micheli, G.: 'XpipesCompiler: a tool for instantiating application specific networks on chip'. Proc. DATE 2004, 2004, pp. 884–889
27 Agarwal, V., Hrishikesh, M.S., Keckler, S.W., and Burger, D.: 'Clock rate versus IPC: the end of the road for conventional microarchitectures'. Proc. Int. Symp. Computer Architecture, June 2000, pp. 248–250
28 Carloni, L.P., McMillan, K.L., and Sangiovanni Vincentelli, A.L.: 'Theory of latency-insensitive design', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., September*, 2001, **20**, (9), pp. 1059–1076
29 ITRS 2001. http://public.itrs.net/Files/2001ITRS/Home.htm
30 Glaskowsky, P.: 'Pentium4 (partially) previewed', *Microprocess. Rep.*, 2000, **14**, (8), pp. 10–13
31 Duato, J., Yalamanchili, S., and Ni, L., 'Interconnection networks: an engineering approach' (IEEE Computer Society Press, 1997)
32 VSI Alliance: 'Virtual Component Interface Standard' 2000
33 OCP International Partnership, 'Open Core Protocol Specification', 2001
34 Lines, A.: 'Asynchronous interconnect for synchronous SoC design', *IEEE Micro*, 2004, **24**, (1), pp. 32–41
35 Muttersbach, J., Villiger, T., Kaeslin, H., Felber, N., and Fichtner, W.: 'Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems'. IEEE ASIC/SOC Conf., September 1999, pp. 317–321
36 Lee, S.J., *et al.*: 'An 800 MHz star-connected on-chip network for application to systems on a chip'. ISSCC Digest of Tech. Papers, 2003, pp. 468–469
37 Murali, S., and De Micheli, G.: 'SUNMAP: a tool for automatic topology selection and generation for NoCs'. Proc. Des. Autom. Conf., 2004, pp. 914–919
38 Van der Tol, E.B., and Jaspers, E.G.T.: 'Mapping of MPEG4 decoding on a flexible architecture platform', *Proc. SPIE-Int. Soc. Opt. Eng.*, 2001, **4674**, pp. 1–13
39 Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and De Micheli, G.: 'NoC synthesis flow for customized domain specific multiprocessor systems-on-Chip', *IEEE Trans. Parallel Distrib. Syst.*, Special issue on On-chip networks, to be published
40 Wang, H.S. *et al.*: 'Orion: A power-performance simulator for interconnection networks'. Proc. IEEE/ACM Int. Symp. on Microarchitecture, Istanbul, Turkey, Nov. 2002, pp. 294–305