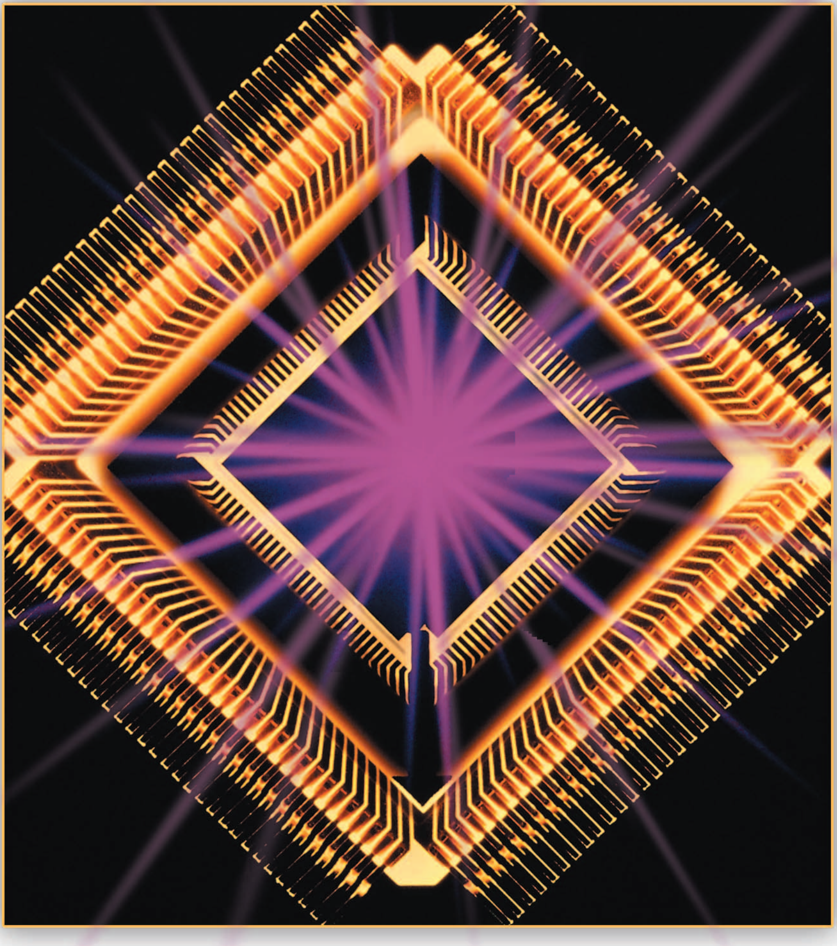
Feature

Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip

Davide Bertozzi and Luca Benini



Abstract

The growing complexity of embedded multi-processor architectures for digital media processing will soon require highly scalable communication infrastructures. Packet switched Networks-on-Chip (NoC) have been proposed to support the trend for Systems-on-Chip integration. In this paper, an advanced NoC architecture, called Xpipes, targeting high performance and reliable communication for on-chip multi-processors is introduced. It consists of a library of soft macros (switches, network interfaces and links) that are design-time composable and tunable so that domain-specific heterogeneous architectures can be instantiated and synthesized. Links can be pipelined with a flexible number of stages to decouple link throughput from its length and to get arbitrary topologies. Moreover, a tool called XpipesCompiler, which automatically instantiates a customized NoC from the library of soft network components, is used in this paper to test the Xpipes-based synthesis flow for domain-specific communication architectures.

Keywords: Systems-on-Chip, Networks-on-chip, pipelining, reliability, soft macros, network instantiation

Introduction

The integration of an entire system onto the same silicon die (*System-on-Chip, SoC*) has become technically feasible as an effect of the increasing integration densities made available by deep sub-micron technologies and of the computational requirements of the most aggressive applications in the multimedia, automotive and ambient intelligence domain.

SoCs represent high-complexity, high-value semiconductor products that incorporate building blocks from multiple sources (either in-house made or externally supplied): in particular, general-purpose fully programmable processors, co-processors, DSPs, dedicated hardware accelerators, memory blocks, I/O blocks, etc. Even though commercial products currently exhibit only a few integrated cores (e.g., a RISC general purpose MIPS core and a VLIW Trimedia processor for the Philips Nexperia platform for multimedia applications), in the next few years technology will allow the integration of thousands of cores, making a large computational power available.

In contrast to past projections, which assumed that technology advances only needed to be linear and that all semiconductor products would deploy them, today the introduction of new technology solutions is increasingly application driven [1]. As an example, let us consider ambient intelligence, which is considered the new paradigm for consumer electronics. 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 [2], and a single processor will not suffice, thus driving the development of more and more complex *multi-processor SoCs (MPSoCs)*.

In this context, the performance of gigascale SoCs will be limited by the ability to efficiently interconnect pre-designed and pre-verified functional blocks and to accommodate their communication requirements, i.e. it will be communication—rather than computation- dominated. Only an interconnect-centric system architecture will be able to cope with these new challenges.

Current on-chip interconnects consist of low-cost shared communication resources, where an arbitration logic is needed for the serialization of bus access requests: only one master at a time can drive the bus. In spite of its low complexity, the main drawback of this solution is its lack of scalability, which will result in unacceptable performance degradation (e.g., contention-related delays for bus accesses) when the level of SoC integration will exceed a dozen of cores. Moreover, the connection of new blocks to a shared bus increases its associated load capacitance, resulting in more energy consuming bus transactions.

State-of-the-art communication architectures make use of evolutionary approaches, such as full or partial crossbars, allowing a higher degree of parallelism in accessing communication resources, but in the long term more aggressive solutions are required.

A scalable communication infrastructure that better supports the trend of SoC integration consists of an on-chip packet-switched micro-network of interconnects, generally known as *Network-on-Chip* (NoC) architecture [3, 4, 5]. The basic idea is borrowed from traditional large-scale multi-processors and the wide-area networks domain, and envisions on-chip router (or switch)-based networks on which packetized communication takes place, as depicted in Fig. 1. Cores access the network by means of proper interfaces, and have their packets forwarded to destination through a multi-hop routing path.

The scalable and modular nature of NoCs and their support for efficient on-chip communication potentially leads to NoC-based multi-processor systems characterized by high structural complexity and functional diversity.

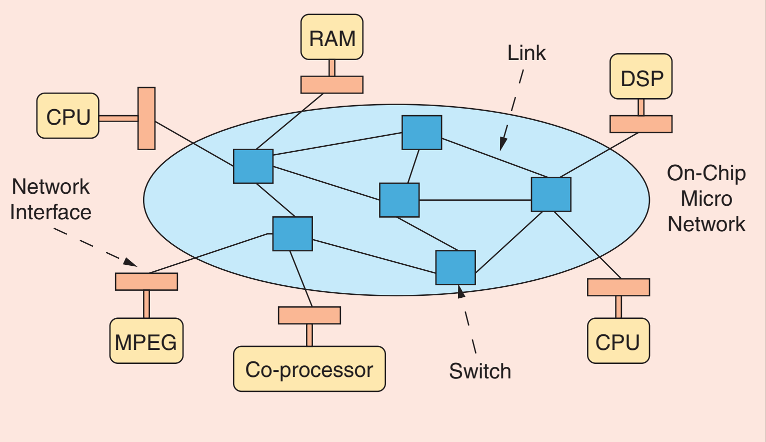


Figure 1. Example of Network-on-Chip architecture.

Davide Bertozzi and Luca Benini are with the Dipartimento Elettronica Informatica Sistemistica, University of Bologna, Via Risorgimento, 2, 40136 Bologna, Italy. E-mail: {dbertozzi,lbenini}@deis.unibo.it.

An important design decision for NoCs concerns topology selection. Several researchers [5, 6, 7, 8] 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. Of course, the network topology depends on system functional requirements, hence on the size and the placement of the integrated modules. In general, SoC integration potentially encompasses heterogeneous technologies and specialized components (used by designers to optimise performance at low power consumption and competitive cost), leading to the integration of heterogeneous cores having varied functionality, size and communication requirements [9]. This might motivate the adoption of custom, domain-specific irregular topologies [10].

As an example, consider the implementation of an MPEG4 decoder [11], depicted in Fig. 2(b), where blocks are drawn roughly to scale and links represent inter-block communication. First, the embedded memory (SDRAM) is much larger than all other cores and it is a critical communication bottleneck. Block sizes are highly non-uniform and the floorplan does not match the regular, tile-based floorplan showed in Fig. 2(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 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 under-utilizing 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 [12].

This paper illustrates an embodiment of the concept of NoC represented by the *Xpipes* NoC architecture developed at University of Bologna [13]. *Xpipes* aims at providing high performance and reliable communication for gigascale SoCs. While describing the basic building blocks of this solution, the main NoC design principles will be briefly explained, detailing the specific contribution of *Xpipes* to the fast-evolving research activity in the domain of NoC-based communication architectures.

Xpipes has been designed as a library of highly parameterised soft macros (network interface, switch and switch-to-switch link), which are design-time tunable and composable. Therefore, arbitrary topologies can be instantiated and the challenging issue of heterogeneous NoC design can be addressed. In fact, an illustrative example will show how *Xpipes* can be employed in a design space exploration framework, wherein the ability of several customized NoC topologies to efficiently accommodate the communication requirements of an application is investigated and compared.

Design Challenges for On-Chip Communication Architectures

Designing communication architectures for highly integrated deep sub-micron SoCs is a non-trivial task that needs to take into account the main challenges posed by technology scaling and by exponentially increasing system complexity. A few relevant SoC design issues are hereafter discussed:

- *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 [3]. Therefore, limiting the on-chip distance travelled 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. Synchronization of cores on future SoCs will be unfeasible due to deep sub-micron effects (clock skew, power associated with clock distribution trees, etc.), and an alternative scenario consists of self-synchronous cores that communicate with one another through a network-centric architecture [14]. 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

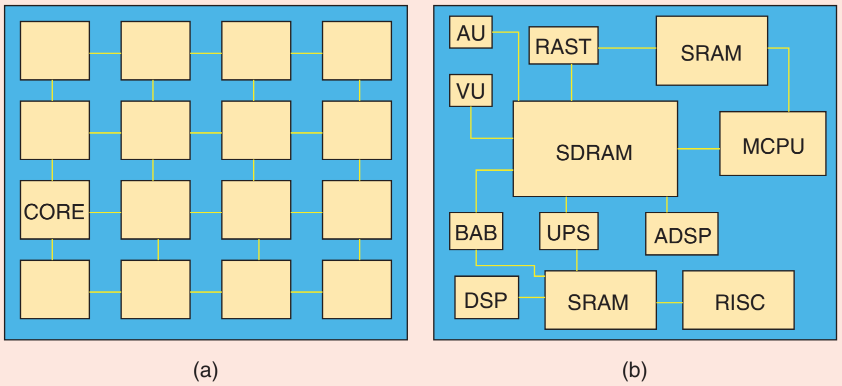


Figure 2. Homogeneous versus heterogeneous architectural template. (a) Tile-Based On-Chip Multi-Processor. (b) MPEG-4 MPSoC.

interconnects, thus raising the reliability concern for on-chip communication [15]. In many cases, on-chip networks can be designed as regular structures, allowing electrical parameters of wires to be optimised and well controlled. This leads to lower communication failure probabilities, thus enabling the use of low swing signalling techniques [16], and to the capability of exploiting performance optimisation techniques such as wavefront pipelining [17].

- *Performance issues.* In traditional busses, all communication actors share the same bandwidth. As a consequence, performance does not scale with the level of system integration, but degrades significantly. Though, once the bus is granted to a master, access occurs with no additional delay. On the contrary, NoCs can provide much better performance scalability. No delays are experienced for accessing the communication infrastructure, since multiple outstanding transactions originated by multiple cores can be handled at the same time, resulting in a more efficient network resource utilization. However, given a certain network dimension (e.g., 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.) [18].
- *Design productivity issues.* It is well known that synthesis and compiler technology development does not keep up with IC manufacturing technology development [19]. Moreover, time-to-market needs to be kept as low as possible. The reuse of complex pre-verified design blocks is efficient means to increase productivity, and regards both computation resources and the communication infrastructure [20]. It would be highly desirable to have processing elements that could be employed in different platforms by means of a plug-and-play design style. To this purpose, 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 industry-wide [21]. OCP [21] 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. 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 busses, which is instance-specific) and reused in a large number of products targeting a specific application domain.

The above-mentioned issues are extensively discussed in [3, 5]. Some commercial SoC interconnects, such as the *Sonics MicroNetwork* [23] and the *STBUS* interconnect from STMicroelectronics are examples of evolutionary architectures which provide designers with a degree of freedom to instantiate different bus-based topologies. More radical solutions have been explored. Dally and Lacy sketch the architecture of a VLSI multi-computer using 2009 technology [24], where communication is based on packet switching. This seminal work draws upon the past experiences in designing parallel computers and reconfigurable architectures (FPGAs and their evolutions) [12, 25–27]. Most of the proposed NoC platforms are packet switched and exhibit regular structure. Examples are mesh interconnections [7, 28] and fat-trees [29].

The need to map communication requirements of heterogeneous cores may lead to the adoption of irregular topologies. The *Aethereal* NoC design framework presented in [30] aims at providing a complete infrastructure for developing heterogeneous NoCs 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, customisable at instantiation time for a specific application domain. For instance, the *Proteo* NoC [31] consists of a small library of predefined, parameterised components that allow the implementation of a large range of different topologies, protocols and configurations.

The *Xpipes* interconnect [13] and its synthesizer *XpipesCompiler* [32] push this approach to the limit, by instantiating an application specific NoC from a library of composable components providing reliable and latency insensitive operation.

NoC Architecture

In general, the most relevant tasks of a *network interface*

(NI) are: (i) hiding the details about network communication protocol to the cores, so that they can be developed independently of the communication infrastructure, (ii) communication protocol conversion (from end-to-end to network protocol), (iii) data packetization (packet assembly, delivery and disassembly).

The former objective can be achieved by means of standard interfaces, such as VCI [21] and OCP [22], whose main characteristics are a high degree of configurability to adapt to the core's functionality and the independence of the 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, in addition to the latency of the communication channel. 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 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., the 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 destination. This overhead in terms of packet header is counterbalanced by a simpler routing logic at the network switches: they simply have to look at the routing fields in the packets and route them 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. Packetizing data deals effectively with communication errors. In fact, packet boundaries contain the effect of errors and allow error recovery on a packet-by-packet basis.

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.

The 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 [33].

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 employed 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 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 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 set-up at the granularity of connections, and time-related guarantees during data transport can be given. In *time division circuit switching* (see [23] 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.

Finally, the design of communication (switch-to-switch and network interface-to-switch) links represents a critical issue with respect to the system performance. 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. Techniques have to be devised to decouple link throughput from its length and to allow functionally correct operation of the switches in presence of mis-aligned inputs due to the unequal length of the input links.

Next, implementation details about *Xpipes* network building blocks are provided, illustrating the specific design choices and how the above-mentioned issues have been addressed.

XPipes Architecture Overview

Xpipes targets Multi-GHz heterogeneous packet-switched NoCs, thanks to the design of high performance network building blocks and to their instantiation time flexibility. The *Xpipes* design process was therefore partitioned into two relevant stages: (i) development of highly parameterizable network components, written in synthesizable SystemC; (ii) definition of a NoC instance by specifying parameters of the composable soft macros and an arbitrary network topology. A *XpipesCompiler* tool has been developed for this purpose.

The high degree of parameterisation of *Xpipes* network building blocks regards both global network-specific parameters and block-specific parameters. The former ones include flit size, degree of redundancy of error control logic, address space of the cores, maximum number of hops between any two nodes, maximum number of bits for end-to-end flow control, etc. On the contrary, parameters specific to the network interface are: type of interface (master, slave or both), flit buffer size at the output port, content of routing tables for source-based routing, other interface parameters to the cores such as number of address/data lines, maximum burst length, etc. Parameterisation of the switches mainly regard the number of I/O ports, the number of virtual channels for each physical output link and the link buffer size. Finally, the length of each individual link can be specified in terms of number of repeater stages, as will be discussed next.

Xpipes network interfaces use OCP as point-to-point communication protocol with the cores, and take care of protocol conversion to adapt to the network protocol. Data packetization results in the packet partitioning procedure illustrated in Fig. 3. A *flit type* field allows to identify the head and the tail flit, and to distinguish between header and payload flits.

The NoC backbone relies on a wormhole switching technique and makes use of 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 that flits belonging to a certain packet have to be directed to. This routing algorithm allows a lightweight switch implementation as no dynamic decisions have to be taken at the switching nodes.

One of the main *Xpipes* design guidelines was the support for communication reliability. It was achieved by means of distributed error detection with link level retransmission as error recovery technique. Although a distributed approach to error detection causes a higher area overhead at the network nodes compared with an end-to-end solution, it is better able to contain the effects of error propagation, for instance preventing packets with corrupted header from being directed to the wrong path. In order to counterbalance this overhead, error detection with retransmission of incorrectly received data was preferred to error correction, since this latter requires complex energy-inefficient decoders. If average bit error rate is not high, performance penalty caused by retransmissions as perceived from the application can be neglected and

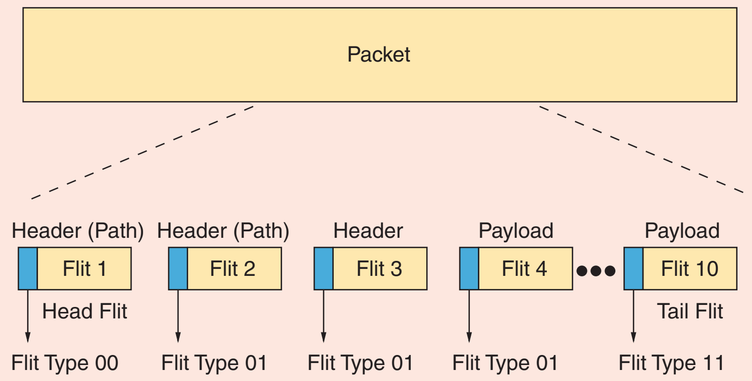


Figure 3. Packet partitioning into flits.

error detecting schemes have been showed to minimize average energy-per-bit [15]: as a consequence, they were the reference choice to provide communication reliability in *Xpipes*.

The retransmission policy implemented in *Xpipes* is *GO-BACK-N*. Flits are transmitted continuously and the transmitter does not wait for an ACK after sending a flit. Such an ACK is received after a round-trip delay. When a NACK is received, the transmitter backs up to the flit that is negatively acknowledged and re-sends it in addition to the *N* succeeding flits that were transmitted during the round-trip delay. At the receiver, the *N-1* received flits following the corrupted one are discarded regardless of whether they were received error-free or not. GO-BACK-N trades-off inefficiency in bus usage (retransmission of many error-free flits) with a moderate implementation cost, and was preferred to a selective-repeat scheme, wherein only those flits that are negatively acknowledged are retransmitted but more buffering resources are required at the destination switch.

In *Xpipes* switches, different error detecting decoders can be instantiated at design time, as a function of their latency and redundancy (additional parity lines) overhead and of their error detection capability, that has to be compared with the estimated (technology-dependent) bit error rate and the required *mean time between failures (MTBF)*. The considered error detecting codes were several versions of the *Hamming code* and of the *Cyclic Redundancy Check (CRC) Code*, each one characterized by a different minimum distance and hence error detection capability. The ultimate objective was to select, for each supply voltage and flit size, coding schemes providing a MTBF of at least 1 year, and to successively select among them a scheme that minimizes decoding latency and/or redundancy.

Finally, it is worth noting that the support for high performance communication was provided in *Xpipes* by means of proper design techniques for the basic network components (such as link pipelining, deeply pipelined switches, latency insensitive component design). Implementation details about the network building blocks are provided below.

Network Link

A solution to overcome the interconnect-delay problem consists of pipelining interconnects [34, 35]. Wires can be partitioned into segments (or relay stations, which have a function similar to the one of latches in a pipelined data path) whose length satisfies pre-defined timing requirements (e.g., the desired clock speed of the design). In this way, link delay is changed into latency, but the data introduction rate is not bound by the link delay any more. Now the latency of a channel connecting two modules may end up being more than one clock cycle. This requires the system to be composed of modules whose behavior does not depend on the latency of input communication channels [34].

Xpipes interconnect makes use of pipelined links and of latency-insensitive operation of its building blocks. Switch-to-switch links are subdivided into basic segments whose length can be tailored to the desired clock frequency that needs to be achieved in the design. 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 is 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, the input links of the switches can be different and of

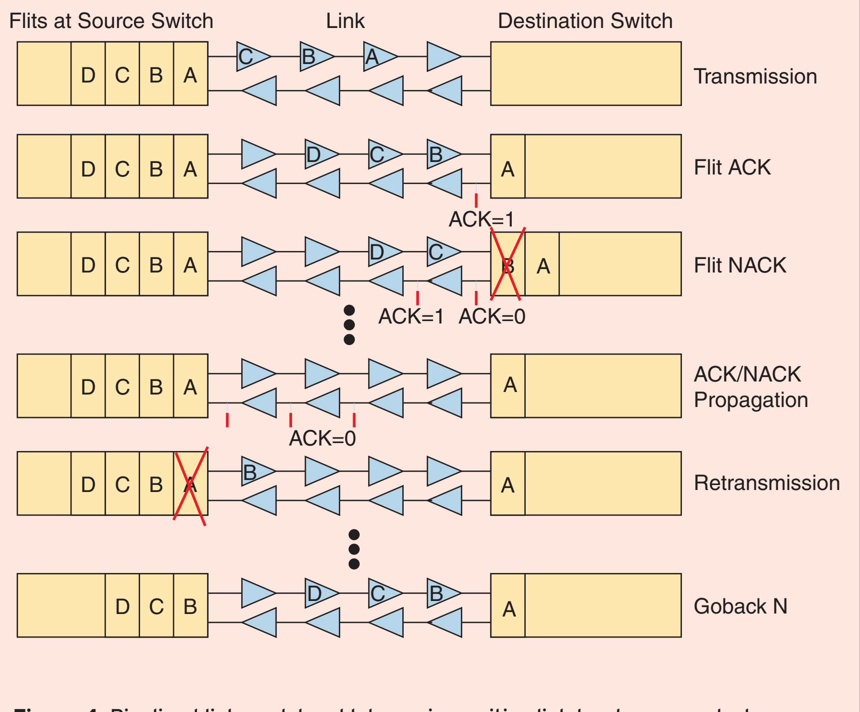


Figure 4. Pipelined link model and latency-insensitive link-level error control.

any length. 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.

Fig. 4 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 affect the operation of latency-insensitive link-level error control: 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, and after N clock cycles (where N is the link length expressed in terms of 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.

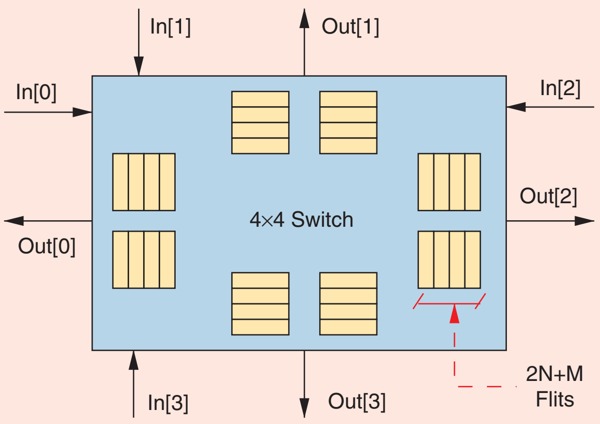


Figure 5. Example of a 4×4 switch configuration with 2 virtual channels.

Switch

A schematic representation of a switch for the *Xpipes* NoC is illustrated in Fig. 5. The example configuration exhibits 4 inputs, 4 outputs and 2 virtual channels multiplexed across the same physical output link. Switch operation is latency insensitive, in that correct operation is guaranteed for arbitrary link pipeline depth. For

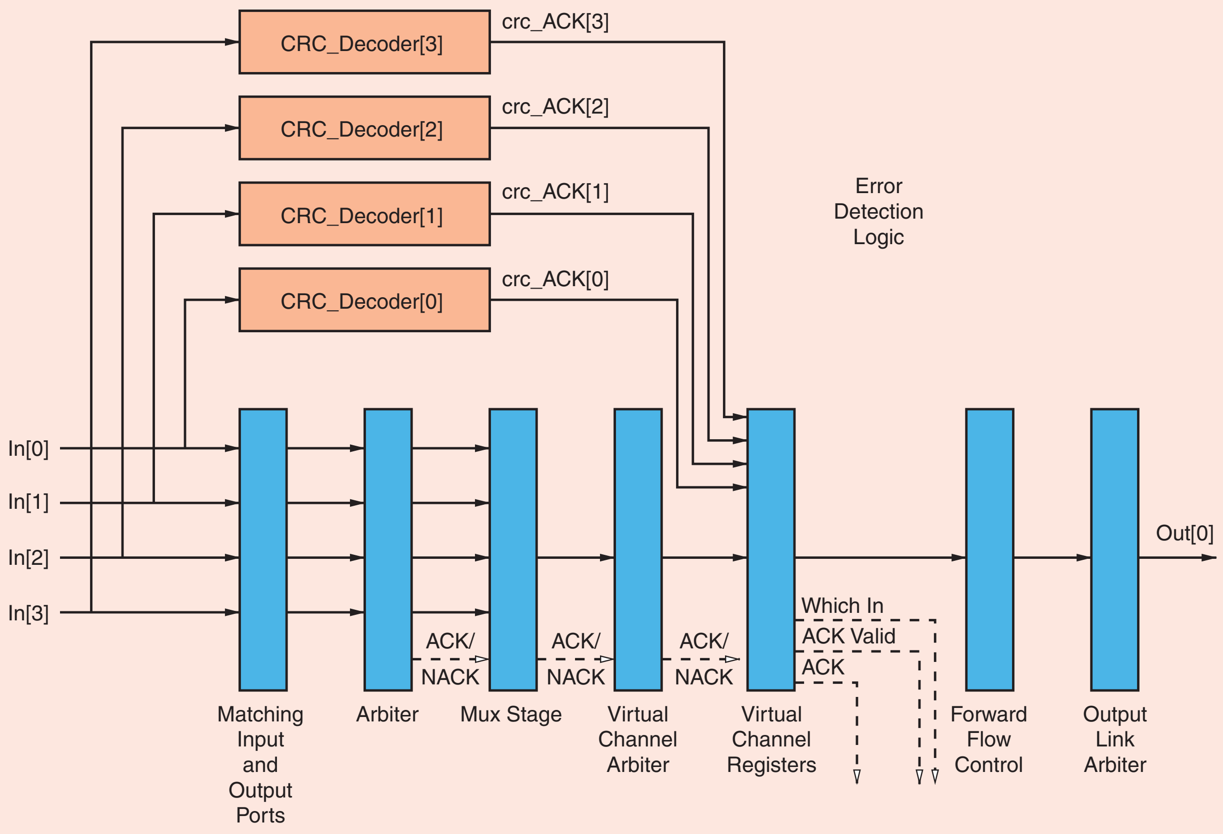


Figure 6. Architecture of the output module for each *Xpipes* switch output port.

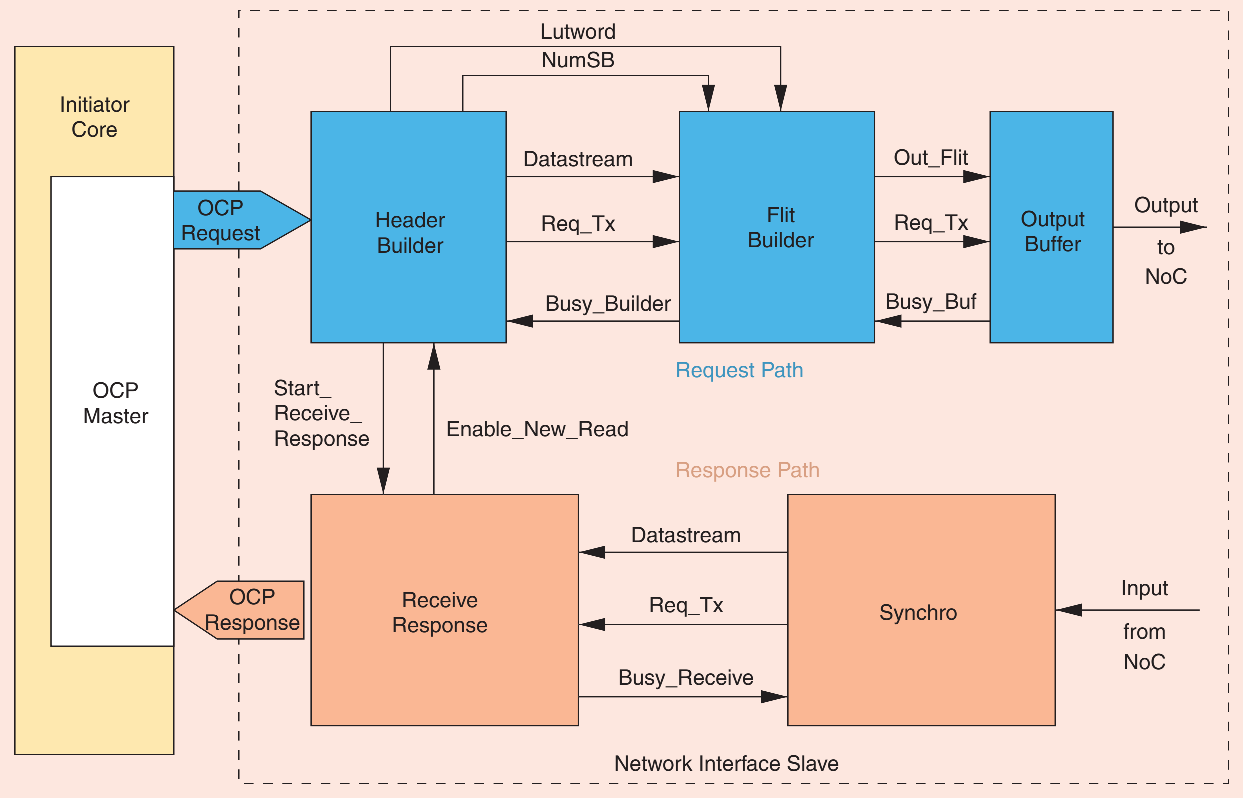


Figure 7. Architecture of *Xpipes* network interface slave.

latency insensitive operation, the switch has virtual channel registers to store $2N + M$ flits, where N is the link length (expressed as 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 propagate across the link (N cycles), an ACK/NACK decision has to be taken at the destination switch (a portion of M cycles), 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 consists of multiple replications of the same output module reported in Fig. 6, one for each switch output port. All switch input ports are connected to the each module's inputs. Flow control signals generated by each module (such as ACK and NACK for incoming flits) are collected by a centralized switch unit, which directs them back to the proper source switch.

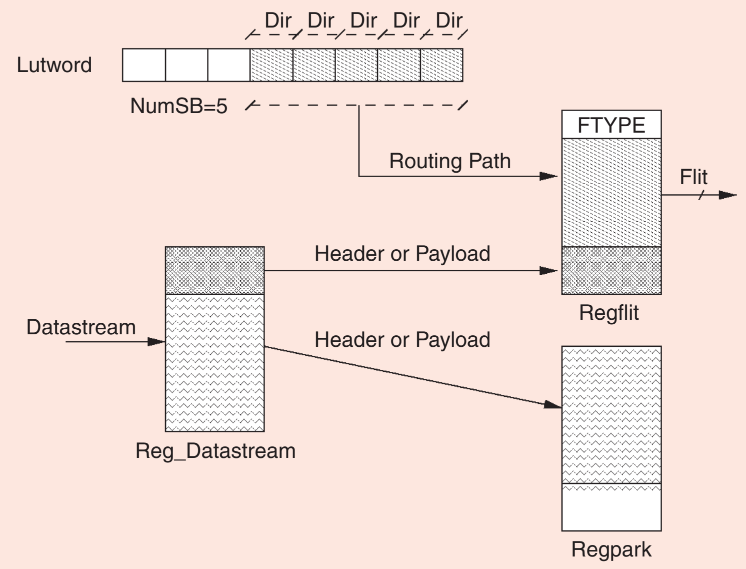


Figure 8. Mechanism for building header flits.

As can be observed from Fig. 6, each output module is deeply pipelined (7 pipeline stages) so to maximize the operating clock frequency of the switch. The CRC decoders for error detection work in parallel with switch operation, thereby hiding their latency.

The first pipeline stage checks the header of incoming packets on the 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 when the tail flit of the preceding packet is received, so that all other flits of a packet can be propagated without contention related delay at this stage. A NACK for flits of non-selected packets is generated. The third stage is just a multiplexer, which selects the prioritised input port. The following arbitration stage keeps the status of virtual channel registers and determines whether flits can be stored into the registers or not. A header flit is sent to the register with the greatest number of free locations, and followed by successive flits of the same packet. The fifth stage is the actual buffering stage, and the ACK/NACK response at this stage indicates whether a flit has been successfully stored or not. The following stage takes care of forward flow control: a flit is transmitted to the next switch only when adequate free register locations are available at the output port of interest of the destination switch. Finally, a last arbitration stage multiplexes the virtual channels on the physical output link on a flit-by-flit basis, thereby improving network throughput.

Network Interface

The *Xpipes* network interface provides a standardized OCP interface to networked cores. 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 (the master side being the initiator core), and it is therefore referred to as *network interface slave (NIS)*. Its architecture is showed in Fig. 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. The lookup table containing static routing information is accessed by the *HEADER_BUILDER* block and the destination address is used as table entry. In this way, two routing fields are extracted: *numSB* (the number of hops to destination) and *lutword* (the actual direction bits read from the look-up table). Together with the core transaction related information (*datastream*), they are forwarded to the *FLIT_BUILDER* block, provided the enable signal *busy_builder* is not asserted.

Based on the input data, the module *FLIT_BUILDER* has the task of building the flits to be transmitted via the output buffer *OUTPUT_BUFFER*, according to the mechanism illustrated in Fig. 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*. The module *FLIT_BUILDER* builds the

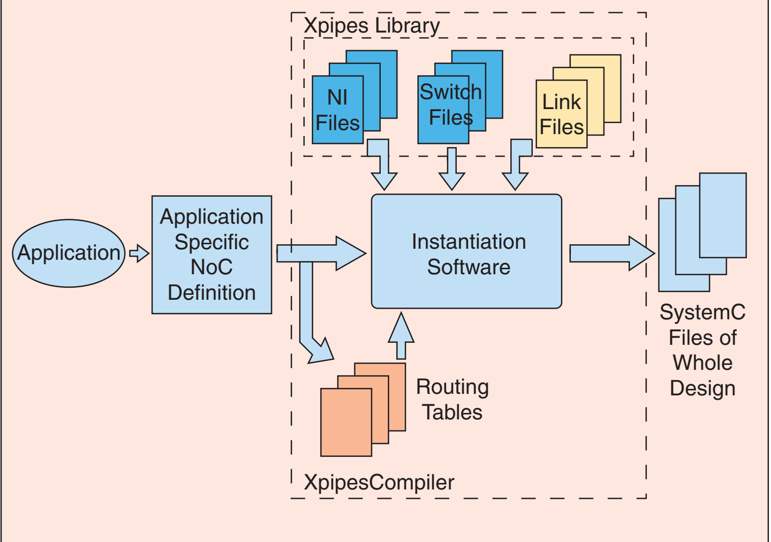


Figure 9. NoC synthesis flow with *XpipesCompiler*.

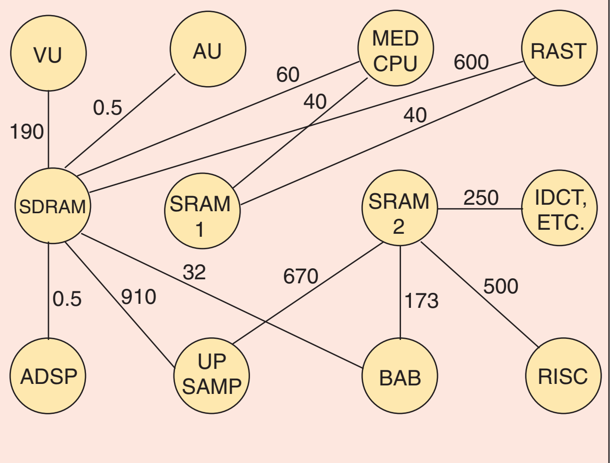


Figure 10. Core graph representation of an MPEG4 design with annotated average communication requirements.

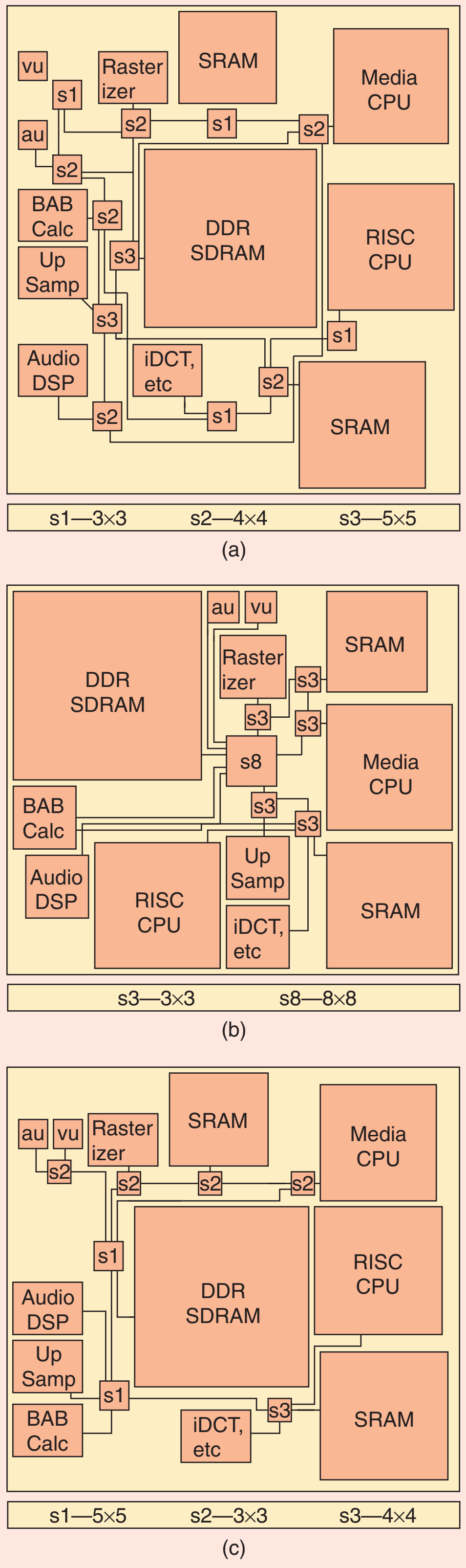


Figure 11. NoC configurations for the MPEG4 decoder. (a) MESH NoC. (b) Application-specific NoC No.1. (c) Application-specific NoC No.2.

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 *HEADER_BUILDER* 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, the module *OUTPUT_BUFFER* stores flits to be sent across the network, and allows the NIS to keep preparing successive flits also when the network is congested. 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 routing fields). At the same time, it contains buffering resources to synchronize the network's requests to transmit the remaining flits of a packet 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 initiator core, the *HEADER_BUILDER* 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 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 or write has been initiated.

The architecture of a *network interface master* is similar to the one just described, and is not reported here for lack of space.

Domain-Specific Network-on-Chip Synthesis Flow

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. Looking for the most efficient solution can be a (possibly automated) step of a more general design methodology of custom domain-specific NoCs, and *Xpipes* could be used within this framework as a library of components for the synthesis of the selected NoC configuration.

Therefore, for a Xpipes-based design methodology, a tool is needed to automatically instantiate network building blocks (switches, network interfaces and links) from the library of composable soft macros described in SystemC. *XpipesCompiler* [32] is the tool designed to automatically instantiate a *Xpipes*-based application-specific NoC for heterogeneous on-chip multi-processors.

The complete *XpipesCompiler*-assisted NoC design flow is depicted in Fig. 9. 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 the clock speed target and link routing, the number of pipeline stages for each link is also determined. The information on the network architecture is then specified in an input file for the *XpipesCompiler*. Routing tables for the network interfaces are also specified. The tool takes the *Xpipes* library of soft network components as an additional input. The output is a SystemC hierarchical description, which includes all switches, links, network nodes and interfaces and specifies their connectivity. Then the final description can 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.

As an example, let us focus on the MPEG4 decoder already introduced in this paper. Its core graph representation together with its average communication requirements are presented in Fig. 10. 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 with a regular mesh topology. The different NoC configurations are presented in Fig. 11. In the MPEG4 design considered, many of the cores communicate with each other through the shared SDRAM. So a large switch is used to connect the SDRAM (Fig. 11(b)), while smaller switches are instantiated for the other cores. An alternative custom NoC is also considered (Fig. 11(c)): it is an optimised mesh network, with superfluous switches and switch I/Os removed. We refer to this solution as a "distributed" custom NoC, as opposed to the "centralized" one.

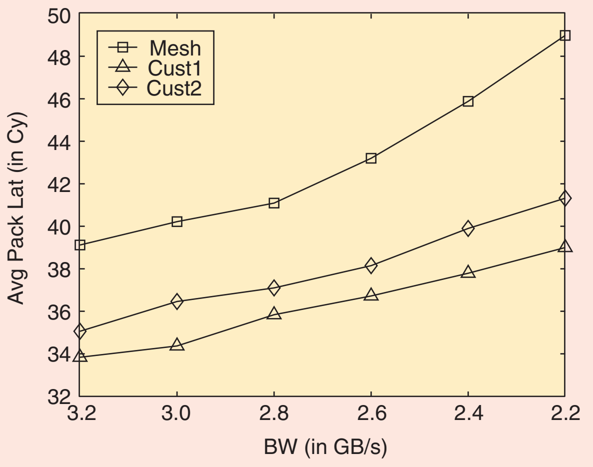


Figure 12. Average packet latency as a function of the link bandwidth.

Area (in 0.1 um technology) and power estimates for different NoC configurations are reported in Table I. As it can be observed, area and power savings are achieved with the custom configurations. However, the distributed solution turns out to be more effective, mainly because the centralized approach makes use of a very large switch. In fact, since power dissipation on a switch increases non-linearly with increase in switch size, there is more power dissipation on the switches of the custom NoC1 (that has an 8×8 switch) than in the mesh NoC. However, most of the traffic traverses short links in this custom NoC, thereby providing marginal power savings for the whole design. Power and area savings achieved by the custom NoC2 are more significant thanks to the use of smaller switches.

Fig. 12 reports the variation of average packet latency (for 64 B 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 rapid-

Table 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

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

Overall, it should be observed that area, power and performance optimisations by means of custom NoCs turn out to be more difficult for MPEG4 than for other applications such as Video Object Plane Decoders (VOPD) and Multi-Window Displayers [32]. In fact, the MPEG4 core graph shows that almost all cores communicate with many other cores (thus requiring many switches) and most of the traffic traverses the larger switches connected to the memories (thus generating high power dissipation). On the contrary, a custom NoC can be generated for VOPD by observing that a half of the cores communicate with more than a single core. This motivates a configuration of the custom NoC, having less than half the number of switches than the mesh NoC. In this way, the custom NoC consumes about 5.7 times less area and 2.7 times less power than the corresponding mesh NoC.

These examples show that customized domain-specific NoCs exhibit large potentials for power minimization of gigascale on-chip multi-processors, and push the development of design exploration frameworks helping designers to assess the effectiveness of alternative custom NoC architectures. *Xpipes* and its synthesizer have been designed as key elements of such a NoC exploration and synthesis flow.

Conclusions

Current application and technology trends motivate a paradigm shift in on-chip interconnect architectures from bus-based solutions to packet-switched Networks-on-Chip. This paper analyses the key drivers for the development of packet-switched on-chip micro-networks and introduces a NoC architecture targeting high performance and reliable communication for domain-specific SoCs. *Xpipes* is a library of soft macros (switches, network interfaces and links) that are design-time composable and tunable, well suited for irregular topologies. An example NoC synthesis flow has also been illustrated, wherein a compiler tool (*XpipesCompiler*) is used to automatically instantiate customized network building blocks from the *Xpipes* library.

Xpipes and related NoC design efforts are mostly focused on revisiting the architectural foundations of the on-chip communication. This is, however, only a facet of a much more complex challenge, namely, the development of a comprehensive architectural, software and operating system infrastructure required to efficiently support the next generation large-scale multi-processor Systems-on-Chip.

Bibliography

[1] A. Allan, D. Edenfeld, W.H. Joyner, Jr, A.B. Kahng, M. Rodgers, and Y. Zorian, "2001 Technology Roadmap for Semiconductors," *IEEE Computer*, pp. 42–53, Jan. 2002 .

[2] F. Boekhorst, "Ambient intelligence, the next paradigm for consumer electronics: How will it affect silicon?," *ISSCC 2002*, vol. 1, pp. 28–31, Feb. 2002.

[3] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[4] P. Wielage and K. Goossens, "Networks on silicon: Blessing or nightmare?," in *Proc. Of the Euromicro Symp. on Digital System Design DSD02*, Sept. 2002, pp.196–200.

[5] W.J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," *Design and Automation Conf. DAC01*, Jun. 2001, pp. 684–689.

[6] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet switched interconnections," *Design, Automation and Testing in Europe DATE00*, Mar. 2000, pp. 250–256.

[7] S. Kumar, 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 Symp. on VLSI ISVLSI02*, Apr. 2002, pp. 105–112.

[8] S.J. Lee et al., "An 800 MHz Star-Connected On-Chip Network for Application to Systems on a Chip," *ISSCC03*, Feb. 2003.

[9] S. Ishiwata et al., "A single chip MPEG-2 codec based on customizable media embedded processor," *IEEE JSSC*, vol. 38, no. 3, pp. 530–540, Mar. 2003.

[10] H. Yamauchi 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.

[11] E. B. Van der Tol and E.G.T. Jaspers, "Mapping of MPEG4 decoding on a flexible architecture platform," *SPIE 2002*, Jan. 2002, pp. 1–13.

[12] H. Zhang et al., "A 1 V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE Journal of SSC*, vol. 35, no.11, pp. 1697–1704, Nov. 2000.

[13] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," *ICCD03*, Oct. 2003, pp. 536–539.

[14] ITRS 2001. Available: http://public.itrs.net/Files/2001ITRS/ Home.htm

[15] D. Bertozzi, L. Benini, G. De Micheli, "Energy-reliability trade-off for NoCs," in *Networks on Chip*, A. Jantsch and Hannu Tenhunen, eds., Dordrecht: Kluwer, 2003, pp. 107–129.

[16] H. Zhang, V. George, and J.M. Rabaey, "Low-swing on-chip signaling techniques: Effectiveness and robustness," *IEEE Trans. on VLSI Systems*, vol. 8, no. 3, pp. 264–272, Jun. 2000.

[17] J. Xu, and W. Wolf, "Wave pipelining for application-specific networks-on-chips," *CASES02*, Oct. 2002, pp. 198–201.

[18] K. Goossens, 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*, A. Jantsch and Hannu Tenhunen, eds. Dordrecht: Kluwer, 2003, pp. 61–82.

[19] ITRS 1999. Available: http://public.itrs.net/files/1999_SIA_Roadmap/

[20] A. Jantsch and H. Tenhunen, "Will networks on chip close the productivity gap?," in *Networks on Chip*, A. Jantsch and Hannu Tenhunen, eds. Dordrecht: Kluwer, 2003, pp. 3–18.

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

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

[23] D. Wingard, "MicroNetwork-based integration for SoCs," *Design Automation Conf. DAC01*, Jun. 2001, pp. 673–677.

[24] W.J. Dally and S. Lacy, "VLSI architecture: Past, present and future,"

Conf. Adv. Research in VLSI, 1999, pp. 232–241.
[25] D. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture, a Hardware/Software Approach,* San Moteo, MA: Morgan Kaufmann, 1999.
[26] K. Compton and S. Hauck, “Reconfigurable computing: A survery of system and software,” *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, Jun. 2002.
[27] R. Tessier and W. Burleson, “Reconfigurable computing and digital signal processing : A survey,” *Journal of VLSI Signal Processing*, vol. 28, no. 3, pp. 7–27, May–Jun. 2001.
[28] Dale Liu et al., “SoCBUS : The solution of high communication bandwidth on chip and short TTM,” invited paper in *Real Time and Embedded Computing Conf.*, Sept. 2002.
[29] J. Walrand and P. Varaja, *High Performance Communication Networks,* San Francisco: Morgan Kaufmann, 2000.
[30] E. Rijpkema, 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*, Mar. 2003, pp. 350–355.
[31] I. Saastamoinen, D. Siguenza-Tortosa, and J. Nurmi, “Interconnect IP node for future systems-on-chip designs,” *IEEE Work on Electronic Design, Test and Applications*, Jan. 2002, pp. 116–120.
[32] “XpipesCompiler: A Tool for Instantiating Application Specific Networks-on-Chip,” submitted to DATE 2004.
[33] J. Duato, S. Yalamanchili, and L. Ni, Interconnection networks: An Engineering Approach, IEEE Computer Society Press, 1997.
[34] L.P. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli, “Theory of latency-insensitive design,” *IEEE Trans. On CAD of ICs and Systems*, vol. 20, no. 9, pp. 1059–1076, Sept. 2001.
[35] L. Scheffer, “Methodologies and tools for pipelined on-chip interconnects,” *Int. Conf. On Computer Design*, 2002, pp. 152–157.

Luca Benini received the B.S. degree (summa cum laude) in electrical engineering from the University of Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1994 and 1997, respectively. He is an associate professor in the Department of Electronics and Computer Science in the University of Bologna. He also holds visiting researcher positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, CA.

Dr. Benini’s research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems. On these topics, he has published more than 200 papers in international conference proceedings and journals. He is a co-author of two books: Dynamic Power Management, Design Techniques and CAD tools, Kluwer 1998, and Memory Design Techniques for Low Energy Embedded Systems, Kluwer 2002.

Dr. Benini is a member of the technical program committee for several technical conferences, including the Design Automation Conference, the International Symposium on Low Power Design and the International Symposium on Hardware-Software Codesign; he is a vice programme chair for the 2004 Design, Automation and Test in Europe.

Davide Bertozzi received the PhD degree in Electrical Engineering and Computer Science from University of Bologna (Italy) in 2003.

He currently holds a Post-Doc position at the same University. He has been a visiting researcher at Stanford University, NEC Research America and Philips Research Labs. His research interests include Multi-Processor System-on-Chip Architectures, with particular emphasis on the communication infrastructure and on its high-performance and energy-efficient implementation.

CALL FOR PAPERS

PATMOS 2004

Fourteenth International Workshop on Power and Timing Modeling, Optimization and Simulation

September 15–17, 2004 **Isle of Santorini, Greece**

http://www.vlsi.ee.upatras.gr/patmos04

The PATMOS objective is to provide a forum to discuss and investigate the emerging problems in methodologies and tools for the design of next generations of integrated circuits and systems. The technical program will focus on speed and low-power aspects, with particular emphasis on modeling, characterization, design and architectures.

Contributions are invited for regular presentations and discussion sessions. Electronic submission is required and should follow the style for the final publication. Check the workshop web-page (http://www.vlsi.ee.upatras.gr/patmos04) for complete author and submission instructions. Submitted papers will be reviewed formally and anonymously by several reviewers.

Proposals for panel sessions and special sessions are encouraged and must be received not later than *March 14, 2004*.

Author’s Schedule

* Deadline for paper submission: **March 14, 2004**
* Notification of acceptance: **May 16, 2004**
* Deadline for final paper submission: **June 13, 2004**

General Co-Chairs
Vassilis Paliouras
paliuras@ee.upatras.gr

Odysseas Koufopavlou
odysseas@ee.upatras.gr
University of Patras
Electrical and Computer Engineering Dept.,
26500 Patras, Greece
Phone: +30 2610 997 319
Fax: +30 2610 994 798

Program Chair
Prof. Enrico Macii
enrico.macii@polito.it
Politecnico di Torino
Dip. di Automatica e Informatica
Corso Duca degli Abruzzi 24
10129 Torino, Italy
Phone: +39-011-564 7074
Fax: +39-011-564 7099