Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
"INGEGNERIA ELETTRONICA"

CICLO XXI

COORDINATORE Prof. Bertozzi Davide

# Driving the Network-on-Chip Revolution to Remove the Interconnect Bottleneck in Nanoscale Multi-Processor Systems-on-Chip.

Settore Scientifico Disciplinare ING-INF/01

| Dottorando | Tutore |
|---|---|
| Dott. Medardoni Simone | Prof. Bertozzi Davide |
| *(firma)* | *(firma)* |

Anni 2006/2008

# ABSTRACT

The sustained demand for faster, more powerful chips has been met by the availability of chip manufacturing processes allowing for the integration of increasing numbers of computation units onto a single die. The resulting outcome, especially in the embedded domain, has often been called SYSTEM-ON-CHIP (SoC) or MULTI-PROCESSOR SYSTEM-ON-CHIP (MP-SoC).

MPSoC design brings to the foreground a large number of challenges, one of the most prominent of which is the design of the chip interconnection. With a number of on-chip blocks presently ranging in the tens, and quickly approaching the hundreds, the novel issue of how to best provide on-chip communication resources is clearly felt.

NETWORKS-ON-CHIPS (NoCs) are the most comprehensive and scalable answer to this design concern. By bringing large-scale networking concepts to the on-chip domain, they guarantee a structured answer to present and future communication requirements. The point-to-point connection and packet switching paradigms they involve are also of great help in minimizing wiring overhead and physical routing issues. However, as with any technology of recent inception, NoC design is still an evolving discipline. Several main areas of interest require deep investigation for NoCs to become viable solutions:

- The design of the NoC architecture needs to strike the best tradeoff among performance, features and the tight area and power constraints of the on-chip domain.
- Simulation and verification infrastructure must be put in place to explore, validate and optimize the NoC performance.
- NoCs offer a huge design space, thanks to their extreme customizability in terms of topology and architectural parameters. Design tools are needed to prune this space and pick the best solutions.
- Even more so given their global, distributed nature, it is essential to evaluate the physical implementation of NoCs to evaluate their suitability for next-generation designs and their area and power costs.

This dissertation performs a design space exploration of network-on-chip architectures, in order to point-out the trade-offs associated with the design of each individual network building blocks and with the design of network topology overall. The design space exploration is preceded by a comparative analysis of state-of-the-art interconnect fabrics with themselves and with early network-on-chip prototypes. The ultimate objective is to point out the key advantages that NoC realizations provide with respect to state-of-the-art communication infrastructures and to point out the challenges that lie ahead in order to make this new interconnect technology come true. Among these latter, technology-related challenges are emerging that call for dedicated design techniques at all levels of the design hierarchy. In particular, leakage power dissipation, containment of process variations and of their effects. The achievement of the above objectives was enabled by means of a NoC simulation environment for cycle-accurate modelling and simulation and by means of a back-end facility for the study of NoC physical implementation effects. Overall, all the results provided by this work have been validated on actual silicon layout.

# Contents

# Chapter 1

# Introduction

Silicon vendors are constantly facing pressure to deliver feature-rich, high-performance, low-power, low-cost chips, in as short a time as possible. Luckily, silicon manufacturing techniques have been continuously perfected, following the well-known Moore's Law; this has provided the potential for answering customer demands.

However, along the years, an increasing gap has been observed among the number of available on-chip transistors and the capability of designers to make good use of them. As a consequence, some trends in chip design have become crystal clear:

- An increasing emphasis on modularity, reuse and parallelism is mandatory. Redesign from scratch is too time-consuming. Also, deploying multiple instances of existing computation blocks can be more efficient than developing more powerful blocks. Therefore, libraries of so-called INTELLECTUAL PROPERTY (IP) cores are increasingly becoming the foundation of platform development.

- Also based on the previous item, complexity is nowadays shifting from the development of functional units to the task of system integration. This is exacerbated by the fact that full designs are nowadays almost impossible to characterize in all possible operating conditions, leading to closure, optimization and verification issues.

- Software tools devoted to design automation are key at all levels. This applies to performance characterization, platform assembly and validation, physical implementation, etc.. Without efficient tools, the sheer complexity of billion-transistor designs and deep-submicron lithographic processes is impossible to tackle by designer teams of any size.

A typical outcome of these trends are today's MULTI-PROCESSOR-SYSTEMS-ON-CHIPS (MPSoCs). These are full-featured chips, composed of a variety of functional blocks, to the point of integrating the foundation of a whole system or device into a single die. MPSoCs are used in a variety of environments, including multimedia gadgets, gaming stations, smartphones, automotive equipment, healthcare devices, industrial machinery, aerospace control units, and many more. MPSoCs are built upon assemblies of IP cores, and rely extensively on COMPUTER AIDED DESIGN (CAD) tooling for initial design space exploration, system optimization, system verification, and physical implementation.

An increasingly critical piece of the MPSoC puzzle is the on-chip interconnection infrastructure. Today, even MPSoCs used in mid-range mobile phones can easily contain tens of IP cores, and new chips with more than a hundred such internal units are appearing for various applications. The trend expressing the number of IP cores that can be integrated on a chip is exponential, roughly doubling every 18 months. How to effectively provide communication resources among such a number of building blocks is clearly a challenge. In fact, it is likely a key factor in determining the success or failure of upcoming MPSoCs will be the ability to efficiently provide the communication backbone into which to seamlessly plug a variety of IP cores.

A comprehensive solution to on-chip interconnection issues has been proposed in the form of NETWORKS-ON-CHIPS (NoCs).

NoCs are packet-switching networks, brought to the on-chip level. The rationale is that, since the complexity of on-chip communication is rapidly approaching that of large area systems in terms of actors, it makes sense to reuse some of the solutions devised in the latter space. Therefore, NoCs are based upon topologies of switches (also called routers) distributing packets around, over point-to-point links. Since existing IP cores do not normally communicate by means of packets, NETWORK INTERFACES (NIs ) (also called network adapters) are in charge of protocol conversion; they convert commands appearing on the pinout of IP cores into packets, and vice versa at the receiving end of a transaction. NoCs have the potential to bring a large number of advantages to on-chip communication, such as:

- Virtually unlimited architectural scalability. As known from wide area networks, it is easy to comply with higher bandwidth requirements by larger numbers of cores simply by deploying more switches and links.

- Much better electrical performance. All connections are point-to-point. The length of inter-switch links is a design parameter that can be adjusted. The wire parallelism in links can be controlled at will, since packet transmission can be serialized. All these factors imply faster propagation times and total control over crosstalk issues.

- Also due to the possibility of having narrower links than in buses (e.g. 32 bits instead of 100), routing concerns are greatly alleviated, and wiring overhead is dramatically reduced. This leads to higher wire utilization and efficiency.

- Faster and easier design closure achievement. Physical design improvements make NoCs, in general, more predictable than buses. Therefore, it is more unlikely that costly respins will be required upon physical design and performance qualification.

- Better performance under load. Since the operating frequency can be higher than in buses, the data width is a parameter, and communication flows can be handled in parallel with suitable NoC topology design, virtually any bandwidth load can be tackled.

- More modular, plug&play-oriented approach to system assembly. IP cores are attached in point-to-point fashion to dedicated NIs; NIs can be specialized for any interface that may be needed, either industry standards such as AMBA AHB or any custom protocol. Potentially any core may be seamlessly attached to a NoC given the proper NI. Computation and communication concerns are clearly decoupled at the NI level.

- Potential for the development of streamlined design flows. While hierarchical buses are often assembled by hand and therefore must be tuned and validated with manual intervention, a network can be designed, optimized and verified by automated means, leading to large savings in design times, and getting a solution closer to optimality.

- A much larger design space. NoCs can be tuned in a variety of parameters (topology, buffering, data widths, arbitrations, routing choices, etc.), leading to higher chances of optimally matching design requirements. Being distibuted, modular structures, NoCs can also accommodate differently tuned regions. For example, some portions of a NoC could be tuned statically for lower resource usage and lower performance (e.g. by reducing the data width), or could dynamically adjust their mode of operation (e.g. frequency, voltage scaling).

At the same time, NoCs are facing a completely different set of constraints compared to wide area networks. While in the latter environment a switch is implemented with at least one dedicated chip, in a NoC the switch must occupy a tiny fraction of the chip real estate. This means that some of the principles acquired in wide area networking have to be revisited. Some of the challenges lying ahead of NoCs include:

- The tradeoffs among network features, area and power budgets have to be studied from scratch. Policies which are widely accepted in general networking (e.g. dynamic packet routing) must be reassessed to evaluate their impact on silicon area.

- Performance requirements are very different in the on-chip domain, also due to the completely different properties of on-chip wiring. Bandwidth milestones are much easier to achieve, since information transfer across on-chip wires is much faster than across long cables. Conversely, latency bounds are much stricter; while milliseconds or even hundreds of milliseconds are acceptable for wide area networks, IP cores on a chip normally require response times of few nanoseconds.

- Contrary to wide area networks, where nodes may often be dynamically connected to and disconnected from the network, in NoCs the set of attached IP cores is obviously fixed. In many applications, it is also relatively easy to statically characterize the traffic profiles of such IP cores. This opens up the possibility of thoroughly customizing NoCs for specific workloads. How to achieve this goal is, however, less clear.

- Design tools for NoCs can be developed, but, as above, how exactly is an open question. The customizability of NoCs, while an asset, is also an issue when it comes to devising tools capable of pruning the design space in search of the optimal solutions. The problem is compounded by the need to take into account both architectural and physical properties; by the need to guarantee design closure; and by the need to validate that the outcome is fully functional, e.g. deadlock-free and compliant with performance objectives.

- NoCs are a recent technology, and as such, they are in need of the development of thorough infrastructure. In addition to design tools, this includes simulators, emulation platforms, and back-end flows for the implementation on APPLICATION-SPECIFIC INTEGRATED CIRCUITS (ASICS).

The main objective of this dissertation is to perform a design space exploration of network-on-chip architectures, in order to point-out the trade-offs associated with the design of each individual network building block and with the design of network topology overall. This exploration aims at providing guidelines to system designers as to the most suitable network design configuration to meet predefined performance, area and power constraints. The design space exploration is preceded by a comparative analysis of state-of-the-art interconnect fabrics with themselves and with early network-on-chip prototypes. The ultimate objective is to point out the key advantages that NoC realizations provide with respect to state-of-the-art communication infrastructures and to point out the challenges that lie ahead in order to make this new interconnect technology come true. Among these latter, technology-related challenges are emerging that call for dedicated design techniques at all levels of the design hierarchy. In particular, leakage power dissipation, containment of process variations and of their effects. The achievement of the above objectives was enabled by means of a NoC simulation environment for cycle-accurate modelling and simulation and by means of a back-end facility for the study of NoC physical implementation effects. Overall, all the results provided by this work have been validated on actual silicon layout.

# Chapter 2

# State of the Art SoC Communication Architecture

## 2.1 Introduction

The current high levels of on-chip integration allow for the implementation of increasingly complex Systems-on-Chip (SoCs), consisting of heterogeneous components such as general purpose processors, DSPs, coprocessors, memories, I/O units, and dedicated hardware accelerators.

In this context, Multi-Processor Systems-on-Chip (MPSoCs) are emerging as an effective solution to meet the demand for computational power posed by application domains such as network processors and parallel media processors. MPSoCs combine the advantages of parallel processing with the high integration levels of Systems-on-Chips (SoCs).

It is expected that future MPSoCs will integrate hundreds of processing units and storage elements, and their performance will be increasingly interconnect-dominated [21]. Interconnect technology and architecture will become the limiting factor for achieving operational goals, and the efficient design of low-power, high-performance on-chip communication architectures will pose novel challenges. The main issue regards *scalability* of system interconnects, since the trend for system integration is expected to continue. State-of-the-art on-chip buses rely on shared communication resources and on an arbitration mechanism which is in charge of serializing bus access requests. This widely adopted solution unfortunately suffers from power and performance scalability limitations, therefore a lot of effort is being devoted to the development of advanced bus topologies (e.g., partial or full crossbars, bridged buses) and protocols, some of them already implemented in commercially available products. In the long run, a more aggressive approach will be needed, and a design paradigm shift will most probably lead to a packetized on-chip communication based on micronetworks of interconnects or Networks-on-Chip (NoCs) [17, 15].

This chapter focuses on state-of-the-art SoC communication architectures, providing an overview of the most relevant ones from an industrial and research

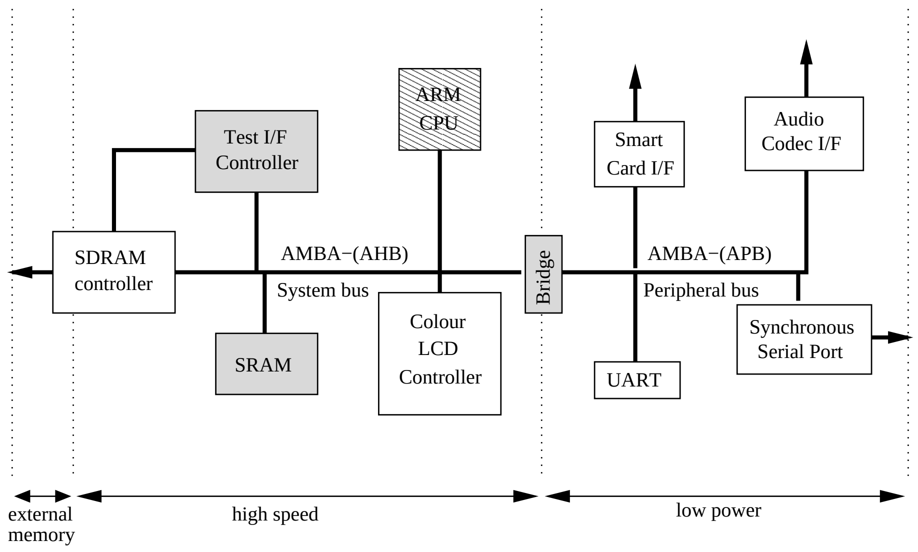


Figure 2.1: Schematic architecture of AMBA bus.

viewpoint. Open bus specifications such as AMBA and CoreConnect will be obviously described more in detail, providing the background which is needed to understand the necessarily more general description of proprietary industrial bus architectures, while at the same time being able to assess their contribution to the advance in the field.

## 2.2 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. However, nowadays AMBA is one of the leading on-chip busing systems because it is licensed and deployed for use with third party IP cores [1]. Designed for custom silicon, the AMBA specification provides standard bus protocols for connecting on-chip components, custom logic and specialized functions. These bus protocols are independent of the ARM processor and generalized for different SoC structures.

AMBA defines a segmented bus architecture, wherein two bus segments are connected with each other via a bridge that buffers data and operations between them. 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 figure 2.1.

### 2.2.1 AMBA System Bus

ASB is the first generation of AMBA system bus, and sits above APB in that it implements the features required for high-performance systems including burst transfers, pipelined transfer operation and multiple bus masters. AHB is a later generation of AMBA bus which is intended to address the requirements of high-performance, high-clock synthesizable designs. ASB is used for simpler, more cost-effective designs whereas more sophisticated designs call for the employment of the AHB. For this reason, a detailed description of AHB follows.

The main features of AMBA AHB can be summarized as follows:

- *Multiple bus masters.* Optimized system performance is obtained by sharing resources among different bus masters. A simple request-grant mechanism is implemented between the arbiter and each bus master. In this way, the 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.

- *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. This overlapping of address and data is fundamental to the pipelined nature of the bus and allows for high performance operation, while still providing adequate time for a slave to provide the response to a transfer. This also implies that ownership of the data bus is delayed with respect to ownership of the address bus. Moreover, support for burst transfers allows for efficient use of memory interfaces by providing transfer information in advance.

- *Split transactions.* They maximize the use of bus bandwidth by enabling high latency slaves to release the system bus during dead time while they complete processing of their access requests.

- *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.

- *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).

A typical AMBA AHB system contains the following components:

**AHB master:** Only one bus master at a time is allowed to initiate and complete read and write transactions. Bus masters drive out the address and control signals and the arbiter determines which master has its signals routed to all of the slaves. A central decoder controls the read data and response signal multiplexor, which selects the appropriate signals from the slave that has been addressed.

**AHB slave:** It signals back to the active master the status of the pending transaction. It can indicate that the transfer completed successfully, or that there was an error or that the master should retry the transfer or indicate the beginning of a split transaction.

**AHB arbiter:** The bus arbiter serializes bus access requests. The arbitration algorithm is not specified by the standard and its selection is left as a design parameter (fixed priority, round-robin, latency-driven, etc.), although the request-grant based arbitration protocol has to be kept fixed.

**AHB decoder:** This is used for address decoding and provides the select signal to the intended slave.

### 2.2.2 AMBA AHB Basic Operation

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. On the contrary, 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.

### 2.2.3 Advanced Peripheral Bus (APB)

The AMBA APB is intended for general-purpose low-speed low-power peripheral devices. It enables the connection to the main system bus via a bridge. All bus devices are slaves, the bridge being the only peripheral bus master.

This is a static bus that provides a simple addressing, with latched addresses and control signals for easy interfacing. ARM recommends a dual Read and

Write bus implementation, but APB can be implemented with a single tristated data bus.

The main features of this bus are the following:

- Unpipelined architecture.
- Low gate count.
- Low power operation.
  - Reduced loading of the main system bus is obtained by isolating the peripherals behind the bridge.
  - Peripheral bus signals are only active during low-bandwidth peripheral transfers.

AMBA APB operation can be abstracted as a state machine with three states. The default state for the peripheral bus is *IDLE*, which switches to *SETUP* state when a transfer is required. SETUP state lasts just one cycle, during which the peripheral select signal is asserted. The bus then moves to *ENABLE* state, which also lasts only one cycle and which requires the address, control and data signals to remain stable. Then, if other transfers are to take place, the bus goes back to SETUP state, otherwise to IDLE. As can be observed, AMBA APB should be used to interface to any peripherals which are low-bandwidth and do not require the high performance of a pipelined bus interface.

### 2.2.4 Advanced AMBA Evolutions

Recently, some advanced specifications of AMBA bus have appeared, featuring increased performance and better link utilization. In particular, the *Multi-Layer AHB* and the *AMBA AXI* interconnect schemes will be briefly addressed in the following sub-sections.

It should be observed that interconnect performance improvement can be achieved by adopting new topologies and by choosing new protocols, at the expense of silicon area. The former strategy leads from shared buses to bridged clusters, partial or full crossbars, and eventually to Networks-on-Chip (NoCs), in an attempt to increase available bandwidth and to reduce local contention. The latter strategy instead tries to maximize link utilization by adopting more sophisticated control schemes and thus permitting a better sharing of existing resources.

Multi-Layer AHB can be seen as an evolution of bus topology while keeping the AHB protocol unchanged. On the contrary, AMBA AXI represents an advanced interconnect fabric protocol.

#### Multi-layer AHB

The Multi-Layer AHB specification emerges with the aim of 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 which enables parallel access paths between multiple masters and slaves in a system [2].

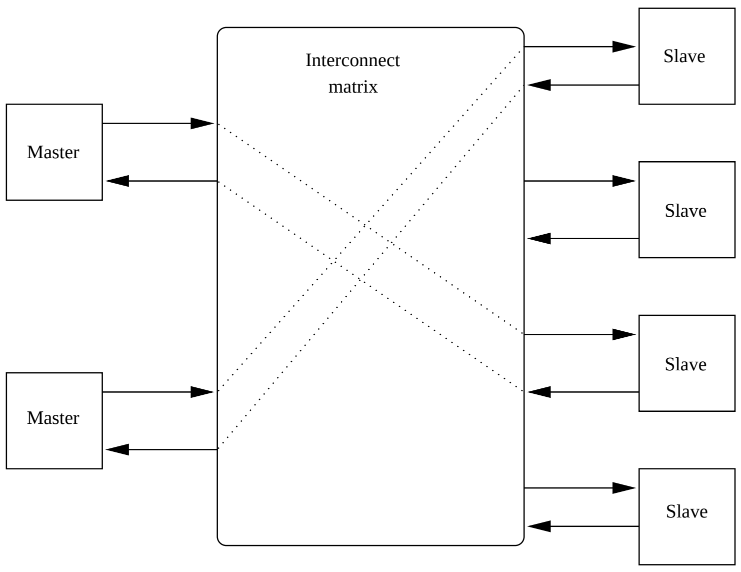


Figure 2.2: Schematic view of the multi-layer AHB interconnect

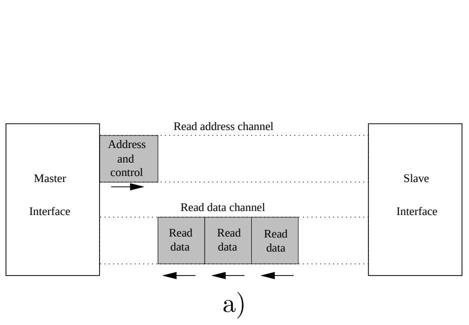
Therefore, the multi-layer bus architecture allows the interconnection of unmodified standard AHB master and slave modules with an increased available bus bandwidth. The resulting architecture becomes very simple and flexible: each AHB layer only has one master and no arbitration and master-to-slave 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 multiplexor to connect the multiple masters to the peripherals and some point arbitration is also required when more than one master wants to access the same slave simultaneously.

Figure 2.2 shows a schematic view of the multi-layer concept. The interconnect matrix contains a decode 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. 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 priority scheme.

The number of input/output ports on the interconnect matrix is completely flexible and can be adapted to suit to system requirements. As the number of masters and slaves implemented in the system increases, the complexity of the interconnection matrix can become significant and some optimization techniques have to be used: defining multiple masters on a single layer, multiple slaves appearing as a single slave to the interconnect matrix, defining local slaves to a particular layer.



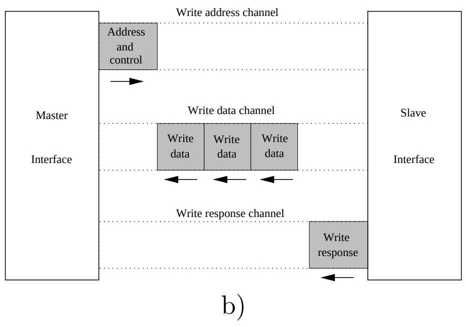


Figure 2.3: Architecture of transfers: a) Read operation; b) Write operation.

Finally, it is interesting to outline the capability of this topology to support multi-port slaves. Some devices, such as SDRAM controllers, work much more efficiently when processing transfers from different layers in parallel.

### AMBA AXI Protocol

AXI is the latest generation AMBA interface. It is designed to be used as a high-speed submicron interconnect, and also includes optional extensions for low-power operation [3]. 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 builds upon the concept of point-to-point connection. AMBA AXI does not provide masters and slaves with visibility of the underlying interconnect, instead featuring the concept of *master interfaces* and symmetric *slave interfaces*. This approach, besides allowing seamless topology scaling, has the advantage of simplifying the handshake logic of attached devices, which only need to manage a point-to-point link.

To provide high scalability and 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 parallelized, allowing multiple outstanding read and write requests.

Figure 2.3a 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 Figure 2.3b.

As can be observed, the 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, in which 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.

However, the AXI protocol is a master/slave-to-interconnect interface definition, and this enables a variety of different interconnect implementations.

Therefore, the mapping of channels, as visible by the interfaces, to actual internal communication lanes 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 signals might be available, up to a full crossbar scheme. 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). Availability of independently scalable resources might, for example, lead to medium complexity designs sharing a single internal address channel while providing multiple data read and write channels.

Finally, some of the key incremental features of the AXI protocol can be listed as follows:

- support for out-of-order completion of transactions
- easy addition of register stages to provide timing closure
- support for multiple address issuing
- separate read and write data channels to enable low-cost Direct Memory Access (DMA)
- support for unaligned data transfers

## 2.3 CoreConnect Bus

CoreConnect is an IBM-developed on-chip bus that eases the integration and reuse of processor, sub-system and peripheral cores within standard product platform designs. It is a complete and versatile architecture clearly targeting high performance systems, and many of its features might be overkill in simple embedded applications [13].

The CoreConnect bus architecture serves as the foundation of IBM Blue Logic$^{\mathrm{TM}}$or other non-IBM devices. The Blue Logic ASIC/SOC design methodology is the approach proposed by IBM [6] to extend conventional ASIC design flows to current design needs: low-power and multiple-voltage products, reconfigurable logic, custom design capability, and analog/mixed-signal designs. Each of these offerings requires a well-balanced coupling of technology capabilities and design methodology. The use of this bus architecture allows the hierarchical design of SoCs.

As can be seen in figure 2.4, the IBM CoreConnect architecture provides three buses for interconnecting cores, library macros, and custom logic:

- Processor Local Bus (PLB).
- On-Chip Peripheral Bus (OPB).
- Device Control Register (DCR) Bus.

The PLB bus connects the processor to high-performance peripherals, such as memories, DMA controllers, and fast devices. Bridged to the PLB, the OPB supports slower-speed peripherals. Finally, the DCR bus is a separate control bus that connects all devices, controllers, and bridges and provides a separate

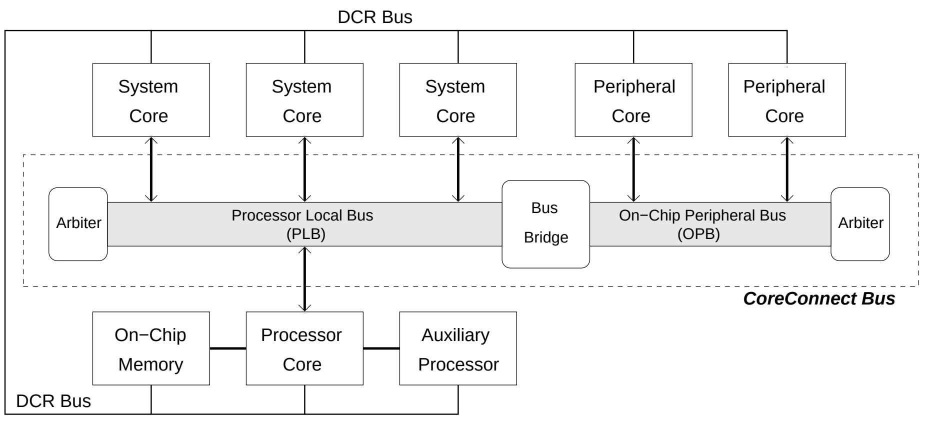


Figure 2.4: Schematic structure of the CoreConnect bus.

path to set and monitor the individual control registers. It is designed to transfer data between the CPU's general purpose registers and the slave logic's device control registers. It removes configuration registers from the memory address map, which reduces loading and improves bandwidth of the PLB.

This architecture shares many high-performance features with the AMBA Bus specification. Both architectures allow split, pipelined and burst transfers, multiple bus masters and 32, 64 or 128-bits architectures. On the other hand, CoreConnect also supports multiple masters in the peripheral bus.

Please note that design toolkits are available for the CoreConnect bus and include functional models, monitors, and a bus functional language to drive the models. These toolkits provide an advanced validation environment for engineers designing macros to attach to the PLB, OPB and DCR buses.

### 2.3.1 Processor Local Bus (PLB)

The PLB is the main system bus targeting high performance and low latency on-chip communication. More specifically, PLB is a synchronous, multi-master, arbitrated bus. It supports *concurrent read and write transfers*, thus yielding a maximum bus utilization of two data transfers per clock cycle. Moreover, PLB implements *address pipelining*, that reduces bus latency by overlapping a new write request with an ongoing write transfer and up to three read requests with an ongoing read transfer [14].

Access to PLB is granted through a central arbitration mechanism that allows masters to compete for bus ownership. This arbitration mechanism is flexible enough to provide for the implementation of various *priority schemes.* In fact, four levels of request priority for each master allow PLB implementation with various arbitration priority schemes. Additionally, an arbitration locking mechanism is provided to support master-driven atomic operations. PLB also exhibits the ability to overlap the bus request/grant protocol with an ongoing transfer.

The PLB specification describes a system architecture along with a detailed description of the signals and transactions. PLB-based custom logic systems require the use of a *PLB macro* to interconnect the various master and slave macros.

The PLB macro is the key component of PLB architecture, and consists of a bus arbitration control unit and the control logic required to manage the address and data flow through the PLB. Each *PLB master* is attached to the PLB through *separate address, read data and write data buses* and a plurality of transfer qualifier signals, while *PLB slaves* are attached through *shared, but decoupled, address, read data and write data buses* (each one with its own transfer control and status signals). The separate address and data buses from the masters allow simultaneous transfer requests. The PLB macro arbitrates among them and sends the address, data and control signals from the granted master to the slave bus. The slave response is then routed back to the appropriate master. Up to 16 masters can be supported by the arbitration unit, while there are no restrictions in the number of slave devices.

### 2.3.2 On-Chip Peripheral Bus (OPB)

Frequently, the OPB architecture connects low-bandwidth devices such as serial and parallel ports, UARTs, timers, etc. and represents a separate, independent level of bus hierarchy. It is implemented as a multi-master, arbitrated bus. It is a fully synchronous interconnect with a common clock, but its devices can run with slower clocks, as long as all of the clocks are synchronized with the rising edge of the main clock.

This bus uses a distributed multiplexer attachment implementation instead of tristate drivers. The OPB supports multiple masters and slaves by implementing the address and data buses as a distributed multiplexer. This type of structure is suitable for the less data intensive OPB bus and allows adding peripherals to a custom core logic design without changing the I/O on either the OPB arbiter or existing peripherals. All of the masters are capable of providing an address to the slaves, whereas both masters and slaves are capable of driving and receiving the distributed data bus.

PLB masters gain access to the peripherals on the OPB bus through the OPB bridge macro. The OPB bridge acts as a slave device on the PLB and a master on the OPB. It supports word (32-bit), half-word (16-bit) and byte read and write transfers on the 32-bit OPB data bus, bursts and has the capability to perform target word first line read accesses. The OPB bridge performs dynamic bus sizing, allowing devices with different data widths to efficiently communicate. When the OPB bridge master performs an operation wider than the selected OPB slave can support, the bridge splits the operation into two or more smaller transfers.

Some of the main features of the OPB specification are:

- Fully synchronous
- Dynamic bus sizing: byte, halfword, fullword and doubleword transfers
- Separate address and data buses
- Support for multiple OPB bus masters
- Single cycle transfer of data between OPB bus master and OPB slaves
- Sequential address (burst) protocol

- 16-cycle fixed bus timeout provided by the OPB arbiter
- Bus arbitration overlapped with last cycle of bus transfers
- Optional OPB DMA transfers

### 2.3.3 Device Control Register Bus (DCR)

The DCR bus provides an alternative path to the system for setting the individual device control registers. These latter are on-chip registers that are implemented outside the processor core, from an architectural viewpoint. Through the DCR bus, the host CPU can set up the device-control-register sets without loading down the main PLB. This bus has a single master, the CPU interface, which can read or write to the individual device control registers. The DCR bus architecture allows data transfers among OPB peripherals to occur independently from, and concurrently with data transfers between processor and memory, or among other PLB devices. The DCR bus architecture is based on a ring topology to connect the CPU interface to all devices. The DCR bus is typicallly implemented as a distributed multiplexer across the chip such that each sub-unit not only has a path to place its own DCRs on the CPU read path, but has also a path which bypasses its DCRs and places another unit's DCRs on the CPU read path. DCR bus consists of a 10-bit address bus and a 32-bit data bus.

This is a synchronous bus, wherein slaves may be clocked either faster or slower than the master, although a synchronization of clock signals with the DCR bus clock is required.

Finally, bursts are not supported by this bus, and 2-cycle minimum read or write transfers are allowed. Optionally, they can be extended by slaves or by the single master.

## 2.4 STBus

STBus is an STMicroelectronics proprietary on-chip bus protocol. STBus is dedicated to SoC designed for high bandwidth applications such as audio/video processing [20]. The STBus interfaces and protocols are closely related to the industry standard VCI (Virtual Component Interface). The components interconnected by an STBus are either *initiators* (which initiate transactions on the bus by sending requests), or *targets* (which respond to requests). The bus architecture is decomposed into *nodes* (sub-buses in which initiators and targets can communicate directly), and the internode communications are performed through FIFO buffers. Figure 2.5 shows a schematic view of the STBus interconnect.

STBus implements three different protocols that can be selected by the designer in order to meet the complexity, cost and performance constraints. From lower to higher, they can be listed as follows:

**Type 1: Peripheral Protocol.** This type is the low cost implementation for low/medium performance. Its simple design allows a synchronous handshake protocol and provides a limited transaction set. The peripheral STBus is targeted at modules which require a low complexity medium

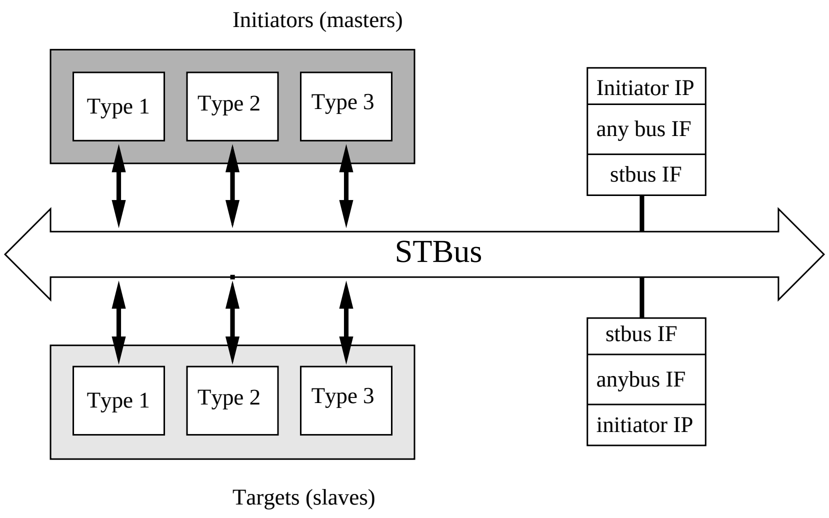


Figure 2.5: Schematic view of the STBus interconnect

data rate communication path with the rest of the system. This typically includes standalone modules such as general-purpose input/output or modules which require independent control interfaces in addition to their main memory interface.

**Type 2: Basic Protocol.** In this case, the limited operation set of the Peripheral Interface is extended to a full operation set, including compound operations, source labeling and some priority and transaction labeling. Moreover, this implementation supports split and pipelined accesses, and is aimed at devices which need high performance but do not require the additional system efficiency associated with shaped request/response packets or the ability to re-order outstanding operations.

**Type 3: Advanced Protocol.** The most advanced implementation upgrades previous interfaces with support for out-of-order execution and shaped packets, and is equivalent to the Advanced VCI protocol. Split and pipelined accesses are supported. It allows performance improvements either by allowing more operations to occur concurrently, or by rescheduling operations more efficiently.

A type 2 protocol preserves the order of requests and responses. One constraint is that, when communicating with a given target, an initiator cannot send a request to a new target until it has received all the responses from the current target. The unresponded requests are called pending, and a pending request controller manages them. A given type 2 target is assumed to send the responses in the same order as the request arrival order. In type 3 protocol, the order of responses may not be guaranteed, and an initiator can communicate with any target, even if it has not received all responses from a previous one.

Associated with these protocols, hardware components have been designed in order to build complete reconfigurable interconnections between Initiators and Targets. A toolkit has been developed around this STBus (graphical interface) to generate automatically top level backbone, cycle accurate high level models,way to implementation, bus analysis (latencies, bandwidth) and bus verification (protocol and behavior).

An STBus system includes three generic architectural components. The *node* arbitrates and routes the requests and optionally, the responses. The *converter* is in charge of converting the requests from one protocol to another (for instance, from basic to advanced). Finally, the *size converter* is used between two buses of the same type but of different widths. It includes buffering capability.

The STBus can implement various strategies of arbitration and allows to change them dynamically. In a simplified single-node system example, a communication between one initiator and a target is performed in several steps.

- a request/grant step between the initiator and the node takes place, corresponding to an atomic rendez-vous operation of the system;
- the request is transferred from the node to the target;
- a response-request/grant step is carried out between the target and the node;
- the response-request is transferred from the node to the initiator.

### 2.4.1 Bus topologies

STBus can instantiate different bus topologies, trading-off communication parallelism with architectural complexity. In particular, system interconnects with different scalability properties can be instantiated such as:

- Single Shared Bus: suitable for simple low-performance implementations. It features minimum wiring area but limited scalability;
- Full Crossbar: targets complex high-performance implementations. Large wiring area overhead.
- Partial Crossbar: intermediate solution, medium performance, implementation complexity and wiring overhead;

It is worth observing that STBus allows for the instantiation of complex bus systems such as heterogeneous multi-node buses (thanks to size or type converters) and facilitates bridging with different bus architectures, provided proper protocol converters are made available (e.g. STBus and AMBA).

## 2.5 WishBone

The WishBone System-on-Chip interconnect [11] defines two types of interfaces, called *master* and *slave*. Master interfaces are cores that are capable of generating bus cycles, while slave interfaces are capable of receiving bus cycles. Some relevant Wishbone features that are worth mentioning are the multi-master capability which enables multi-processing, the arbitration methodology defined by end users attending to their needs, and the scalable data bus widths and operand sizes. Moreover, the hardware implementation of bus interfaces is simple and compact, and the hierarchical view of the WishBone architecture supports structured design methodologies [12].

The hardware implementation supports various IP core interconnection schemes, including: point-to-point connection, shared bus, crossbar switch implementation, data-flow interconnection and off-chip interconnection. The crossbar switch interconnection is usually used when connecting two or more masters together so that every one can access two or more slaves. In this scheme, the master initiates an addressable bus cycle to a target slave. The crossbar switch interconnection allows more than one master to use the bus provided they do not access the same slave. In this way, the master requests a channel on the switch and, once this is established, data is transferred in a point-to-point way.

The overall data transfer rate of the crossbar switch is higher than shared bus mechanisms, and can be expanded to support extremely high data transfer rates. On the other hand, the main disadvantage is a more complex interconnection logic and routing resources.

### 2.5.1 The Wishbone Bus Transactions

The WishBone architecture defines different transaction cycles attending to the action performed (read or write) and the blocking/non-blocking access. For instance, single read/write transfers are carried out as follows. The master requests the operation and places the slave address onto the bus. Then the slave places data onto the data bus and asserts an acknowledge signal. The master monitors this signal and relies the request signals when data have been latched. Two or more back-to-back read/write transfers can also be strung together. In this case, the starting and stopping point of the transfers are identified by the assertion and negation of a specific signal [25].

A read-modify-write (RMW) transfer is also specified, which can be used in multiprocessor and multitasking systems in order to allow multiple software processes to share common resources by using semaphores. This is commonly done on interfaces for disk controllers, serial ports and memory. The RMW transfer reads and writes data to a memory location in a single bus cycle. For the correct implementation of this bus transaction, shared bus interconnects have to be designed in such a way that once the arbiter grants the bus to a master, it will not re-arbitrate the bus until the current master gives it up. Also, it is important to note that a master device must support the RMW transfer in order to be effective, and this is generally done by means of special instructions forcing RMW bus transactions.

## 2.6 SiliconBackplane MicroNetwork

SiliconBackplane MicroNetwork is a family of innovative communication architectures licensed by Sonics for use in SoC design. The Sonics architecture provides CPU independence, true mix-and-match of IP cores, a unified communication medium, and a structure that makes a SOC design simpler to partition, analyze, design, verify, and test [24].

The SiliconBackplane MicroNetwork allows high-speed pipelined transactions (data bandwidth of the interconnect scales from 50Mbytes/s to 4.8Gbyte/s) where the real-time QoS (*Quality of Service*) of multiple simultaneous dataflows is guaranteed. A network utilization of up to 90% can be achieved.

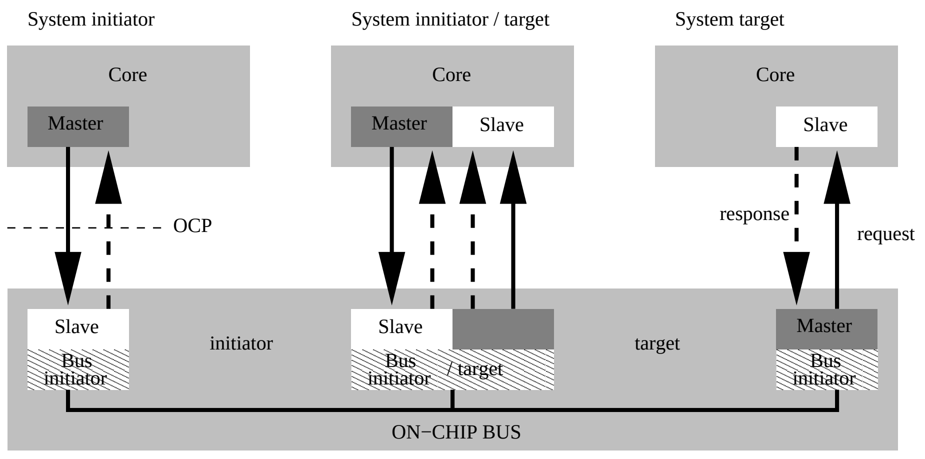


Figure 2.6: Schematic view of the SiliconBackplane system

The SiliconBackplane relies on the SonicsStudio$^{TM}$development environment for architectural exploration, and the availability of pre-characterization results enables reliable performance analysis and reduction of interconnect timing closure uncertainties. The ultimate goal is to avoid over-designing interconnects.

The architecture can be described as a distributed communication infrastructure (thus facilitating place-and-route) which can be extended hierarchically in the form of *Tiles* (collection of functions requiring minimal assistance from the rest of the die) in an easy way. Among other features, the SiliconBackplane MicroNetwork provides advanced error handling in hardware (features for SoC-wide error detection and support mechanisms for software clean-up and recovery of unresponsive cores), runtime-operating reconfiguration to meet changing application demands and data multicast.

The SiliconBackplane system consists of a physical interconnect bus configured with a combination of agents. Each IP core communicates with an attached agent through ports implementing the Open Core Protocol (OCP) standard interface. The agents then communicate with each other using a network of interconnects based on the *SiliconBackplane* protocol. This latter includes patented transfer mechanisms aiming at maximizing interconnect bandwidth utilization and optimized for streaming multimedia applications [23]. Figure 2.6 shows a schematic view of the SiliconBackplane system.

A few specific components can be identified in an agent architecture:

**Initiators:** Who implements the interface between the bus and the master core (CPU, DSP, DMA...). The initiator receives requests from the Open Core Protocol, then transmits the requests according to the SiliconBackplane standard, and finally processes the responses from the target.

**Targets:** Who implements the interface between the physical bus and the slave device (memories, UARTs...). This module serves as the bridge between the system and the Open Core Protocol.

**Service Agent:** Who is an enhanced initiator that provides additional capabilities such as debug and test.

### 2.6.1 System Interconnect Bandwidth

One of the most interesting features of the SiliconBackplane network is the possibility of allocating bandwidth based on a two-level arbitration policy. The system designer can preallocate bandwidth to high priority initiators by means of the concept of *Time-Division Multiple Access (TDMA)*. An initiator agent with a pre-assigned time slot has the rights over that slot. If the owner does not need it, the slot is re-allocated in a round-robin fashion to one of the system devices, and this represents the second level of the arbitration policy.

The TDMA approach provides fast access to variable-latency subsystems and is a simple mechanism to guarantee QoS. The TDMA bandwidth allocation tables are stored in a configuration register at every initiator, and can be dynamically over-written to fit the system needs. On the other hand, the fair round-robin allocation scheme can be used to guarantee bandwidth availability to initiators with less predictable access patterns, since some or many of the TDMA slots may turn out to be left unallocated. Round-robin arbitration policy is particularly suitable for best-effort traffic.

### 2.6.2 Configuration Resources

All the configurable IP cores implemented in the SiliconBackplane system can be configured either at compile time or dynamically by means of specific configuration registers. These configuration devices are accessible by the operating system.

Configuration registers are individually set for each agent, depending upon the services provided to the attached cores. The types of configuration registers are:

- Unbuffered registers: hold configuration values for the agent or its subsystem core.
- Buffered registers: hold configuration values that must be simultaneously updated in all agents.
- Broadcast configuration registers: hold values that must remain identical in multiple agents.

## 2.7 Other On-Chip Interconnects

### 2.7.1 Peripheral Interconnect Bus (PI-Bus)

The PI bus was developed by several european semiconductor companies (Advanced RISC Machines, Philips Semiconductors, SGS-THOMSON Microelectronics, Siemens, TEMIC/MATRA MHS) within the framework of a european project (OMI, Open Microprocessor Initiative framework[1]). After this, an extended backward-compatible PI-Bus protocol standard frequently used in many hardware systems has been developed by Philips [5].

[1]The PI Bus has been incorporated as OMI Standard OMI 324.3D.

The high bandwidth and low overhead of the PI-Bus provide a confortable environment for connecting processor cores, memories, co-processors, I/O controllers and other functional blocks in high performance chips, for time-critical applications.

The PI-Bus functional modules are arranged in macrocells, and a wide range of functions are provided. Macrocells with a PI-Bus interface can be easily integrated into a chip layout even if they are designed by different manufacturers.

The potential bus agents require only a PI-Bus interface of low complexity. Since there is no concrete implementation specified, PI- Bus can be adapted to the individual requirements of the target chip design. For instance, the widths of the address and data bus may be varied. The main features of this bus are:

- Processor independent implementation and design.
- Demultiplexed operation.
- Clock synchronous.
- Peak transfer rate of 200 Mbytes/s (50 MHz bus clock).
- Address and data bus scalable (up to 32 bits).
- 8, 16, 32, bit data access.
- Broad range of transfer types from single to multiple data transfers.
- Multi-master capability.

The PI-Bus does not provide cache coherency support, broadcasts, dynamic bus sizing and unaligned data access. Finally, the University of Sussex has developed a VHDL Toolkit to meet the needs of embedded system designers using the PI-bus. Macrocell testing for PI-bus compliance is also possible using the framework available in the ToolKit [8].

### 2.7.2 AVALON

Avalon is Altera's parameterized interface bus used by the Nios embedded processor. The Avalon switch fabric has a set of pre-defined signal types with which a user can connect one or more IP blocks. It can be only implemented on Altera devices using *SOPCBuilder*, a system development tool that automatically generates the Avalon switch fabric logic.

The Avalon switch fabric enables simultaneous multi-master operation for maximum system performance by using a technique called slave-side arbitration. It determines which master gains access to a certain slave, in the event that multiple masters attempt to access the same slave at the same time. Therefore, simultaneous transactions for all bus masters are supported and arbitration for peripherals or memory interfaces that are shared among masters is automatically included.

The Avalon interconnect includes chip-select signals for all peripherals, even user-defined peripherals, to simplify the design of the embedded system. Separate, dedicated address and data paths provide an easy interface to on-chip user logic. User-defined peripherals are not required to decode data and address bus cycles. Dynamic bus sizing allows developers to use low-cost, narrow memory devices that do not match the native bus size of their CPU. The switch fabric supports each type of transfer supported by the Avalon interface. Each peripheral port into the switch is generated with reduced amount of logic to meet the requirements of the peripheral, including wait-state logic, data width matching, and passing wait signals. Read and write operations with latency can be performed. Latent transfers are useful to masters wanting to issue multiple sequential read or write requests to a slave, which may require multiple cycles for the first transfer but fewer cycles for subsequent sequential transfers. This can be beneficial for instruction-fetch operations and DMA transfers to or from SDRAM. In these cases, the CPU or DMA master may pre-fetch (post) multiple requests prior to completion of the first transfer and thereby reduce overall access latency. Interestingly, the Avalon interface includes signals for streaming data between master/slave pairs. These signals indicate the peripheral's capacity to provide or accept data. A master does not have to access status registers in the slave peripheral to determine whether the slave can send or receive data. Streaming transactions maximize throughput between master-slave pairs, while avoiding data overflow or underflow on the slave peripherals. This is especially useful for DMA transfers.

### 2.7.3 CoreFrame

The CoreFrame architecture has been developed by Palmchip Corp. and relies on point-to-point signals and multiplexing instead of shared tri-state lines. It aims at delivering high performance while simultaneously reducing design and verification time. CoreFrame distinctive features are:

- 400 MB/s bandwidth at 100 MHz (bus speed is scalable to technology and design requirements
- Unidirectional buses only
- Central, shared memory controller
- Single clock cycle data transfers
- Zero wait state register accesses
- Separate peripheral I/O and DMA buses
- Simple protocol for reduced gate count
- Low-capacitive loading for high-frequency operation
- hidden arbitration for DMA bus masters
- Application-specific memory map and peripherals

The most distinctive feature of CoreFrame is the separation of I/O and memory transfers onto different buses. The PalmBus provides for the I/O backplane and allows the processor to configure and control peripheral blocks while the MBus provides a DMA connection from peripherals to main memory, allowing a direct data transfer without processor intervention.

Other on-chip interconnects are not described here for lack of space: IPBus from IDT, IP Interface from Motorola, MARBLE asynchronous bus from University of Manchester, Atlantic from Altera, ClearConnect from ClearSpeed Techn., FISPbus from Mentor Graphics [19].

## 2.8 Conclusions

This chapter addresses the critical issue of on-chip communication for gigascale MPSoCs. An overview of the most widely used on-chip communication architectures is provided, and evolution guidelines aiming at overcoming scalability limitations are sketched. Advances regard both communication protocol and topology, although it is becoming clear that in the long term more aggressive approaches will be required to sustain system performance, namely packet-switched interconnection networks.

# Bibliography

[1] ARM.
*AMBA Specification v2.0*, 1999.

[2] ARM.
*AMBA Multi-layer AHB Overview*, 2001.

[3] ARM.
*AMBA AXI Protocol Specification*, 2003.

[4] Synopsys CoCentric.
http://www.synopsys.com, 2004.

[5] Philip de Nier.
Property checking of PI-Bus modules.
In J.P.Veen, editor, *Proc. ProRISC99 Workshop on Circuits, Systems and Signal Processing*, pages 343–354. STW, Technology Foundation, 1999.

[6] G. W. Doerre and D. E. Lackey.
The IBM ASIC/SoC methodology. a recipe for first-time success.
*IBM Journal Research & Development*, 46(6):649–660, November 2002.

[7] E.Bolotin, I.Cidon, R.Ginosar, and A.Kolodny.
QNoC: QoS architecture and design process for network on chip.
*The Journal of Systems Architecture, Special Issue on Networks on Chip*, December 2003.

[8] ESPRIT.
PI-Bus Systems ToolKit.
http://www.cordis.lu/esprit/src/results/res_area/omi/omi10.htm, 1996.

[9] K.Lee et al.
A 51mw 1.6 ghz on-chip network for low power heterogeneous SoC platform.
*ISSCC Digest of Tech. Papers*, pages 152–154, 2004.

[10] F.Poletti, D.Bertozzi, A.Bogliolo, and L.Benini.
Performance analysis of arbitration policies for SoC communication architectures.
*Journal of Design Automation for Embedded Systems, Kluwer*, (8):189–210, June/September 2003.

[11] Richard Herveille.
*Combining WISHBONE interface signals, Application Note*, April 2001.

[12] Richard Herveille.
*WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Specification*, 2002.

[13] IBM Microelectronics.
*CoreConnect Bus Architecture Overview*, 1999.

[14] IBM Microelectronics.
*The CoreConnect Bus Architecture White Paper*, 1999.

[15] J.Henkel, W.Wolf, and S.Chakradhar.
On-chip networks: A scalable, communication-centric embedded system design paradigm.
*Proc. of Int. Conf. on VLSI Design*, pages 845–851, January 2004.

[16] L.Benini, D.Bertozzi, D.Bruni, N.Drago, F.Fummi, and M.Poncino.
SystemC cosimulation and emulation of multiprocessor SoC designs.
*IEEE Computer*, 36(4):53–59, April 2003.

[17] L.Benini and G.De Micheli.
Networks on chips: a new SoC paradigm.
*IEEE Computer*, 35(1):70–78, January 2002.

[18] M.Loghi, F.Angiolini, D.Bertozzi, L.Benini, and R.Zafalon.
Analyzing on-chip communication in a MPSoC environment.
*Proceed. of IEEE Design Automation and Test in Europe Conference (DATE04)*, pages 752–757, February 2004.

[19] Summary of SoC Interconnection Buses.
http://www.silicore.net/uCbusum.htm, 2004.

[20] P.Wodey, G.Camarroque, F.Barray, R.Hersemeule, and J.P. Cousin.
LOTOS code generation for model checking of STBus based SoC: the STBus interconnection.
*Proc. of ACM and IEEE Int. Conf. on Formal Methods and Models for Co-Design*, pages 204–213, June 2003.

[21] R.Ho, K.W. Mai, and M.A. Horowitz.
The future of wires.
*Proceedings of the IEEE*, 89(4):490–504, April 2001.

[22] E. Rijpkema, K. Goossens, and A. Radulescu.
Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip.
*Proc. of Design Automation and Test in Europe*, pages 350–355, March 2003.

[23] Sonics Inc.
*SiliconBackplane III MicroNetwork IP. Product Brief*, 2002.

[24] Sonics Inc.
*Sonics $\mu$Networks. Technical Overview*, 2002.

[25] Rudolf Usselmann.
*OpenCores SoC Bus Review*, 2001.

# Chapter 3

# Analysing the communication in a memory-centric industrial MPSoC platforms

## 3.1 Introduction

Networked digital products such as set-top-boxes, high-density DVD players or digital video recorders are increasingly relying on Multi-processor Systems-on-Chip (MPSoCs). These platforms feature increasing levels of system integration and high-speed interfaces, thus posing stringent requirements on the performance of the communication architecture.

Traditionally, shared communication resources have been deployed for SoC design, and several arbitration strategies have been investigated for their impact on bus performance. However, this solution scales poorly, since even keeping the bus busy throughout the entire application lifetime does not suffice to meet communication requirements. Therefore, bandwidth across the system has been increased by means of multi-layer communication infrastructures, where clusters of masters and slaves are connected via bridges. This way, a wide variety of bus structures and multi-master systems can be constructed, thus ensuring flexibility to system designers and relieving the scalability limitations of state-of-the-art system interconnects.

At the same time, the complexity of interconnect design increases a lot. In fact, the specific layers building up the overall system interconnect exhibit advanced features such as pipelining, support for split transactions and for multiple outstanding transactions. Moreover, since layers featuring different communication protocols might need to be connected in the same system, the bridge might have to perform a semantic translation of the transactions of one protocol into the transactions of the other protocol. Besides protocol matching, bridges are in charge of additional tasks in heterogeneous MPSoC platforms, such as frequency adaptation and datawidth conversion.

As a consequence, bridges are becoming complex Intellectual Property blocks, highly optimized for a specific communication protocol.

Even deploying highly optimized bridges, performance of the communication architecture of an MPSoC is not guaranteed, but is tightly related to the interaction with the memory and I/O subsystems. The memory architecture determines the prevailing traffic pattern which has to be accommodated by the interconnect fabric. For instance, in presence of many on-chip memory cores, system performance depends on the bus ability to enable concurrent memory accesses without blocking behaviour. However, many state-of-the-art MPSoC platforms for consumer electronics are memory-centric, in that most of the memory is off-chip and the on-chip memory controller becomes the performance bottleneck of the platform. In this case, the interaction between specific features of the communication protocols and the many-to-one traffic pattern has not been analysed yet, and is certainly a function of architectural parameters such as memory access latency or amount of buffering implemented in the system. Moreover, the optimizations performed by advanced memory interfaces (such as those implemented in SDRAM memory controllers) might make this interaction unpredictable.

Given the fine-grain nature of the mechanisms that determine macroscopic MPSoC performance, high-level modelling and simulation tools are likely to fail to capture the needed level of detail which makes global interconnect performance analysis trustworthy. This chapter is centered on cycle-accurate analysis of the communication, memory and I/O architectures of a state-of-the-art MPSoC platform for consumer electronics. We will analyze a multi-abstraction and accurate virtual platform allowing an in-depth investigation of the behaviour of system components, captured in isolation and when inter-operating with each other in a complete MPSoC platform of industrial relevance.

Moreover, modelling extensions to the virtual platform allowed to test architecture variants concerning both the communication infrastructure (use of different communication protocols and topologies), the memory and I/O architectures (on-chip shared memory versus off-chip SDRAM memory, memory controllers with increasing complexity). As a consequence, we were able to shed light on interaction effects between the three main MPSoC subsystems and come up with design guidelines.

The chapter is structured as follows. Section 3.2 illustrates the framework. Our modelling is described in Section 3.3. Experimental results are reported in Sections 3.4 and 3.5. Discussion of results and conclusion are in Section 3.6 and 3.7

## 3.2 MPSIM, a SystemC Platform Simulation

The MPSIM [5, 4] environment is designed to investigate the system-level architecture of MPSoC platforms. To be able to fully assess system performance, a cycle-accurate, signal-accurate modeling infrastructure is put into place.

MPSIM is a plug-and-play platform based upon the SystemC [2] simulation engine, where multiple IP cores and interconnects can be freely mixed and composed. At its core, MPARM is a collection of component models, comprising processors, interconnects, memories and dedicated devices like DMA engines. The user can deploy different system configuration parameters by means of command line switches, which allows for easy scripting of sets of simulation runs. A thorough set of statistics, traces and waveforms can be collected to analyze performance bottlenecks and to debug functional issues. To take into account other crucial design variables, power models for many of the MPARM components are supplied. Frequency and voltage scaling can be realized at runtime thanks to dedicated programmable registers.

MPARM features a choice of several IP cores to be used as system masters. Some of these are taken from the open source or academic domain, and while spanning over a range of architectures, they typically model pre-existing industrial general purpose processors with little to no possibility of modifying the supported instruction set and architecture.

MPARM provides extensive facilities to study the performance of alternative memory hierarchies. Three layers of memory devices are defined: (1) on-tile, strongly coupled to the processor, e.g. caches and SCRATCH - PAD MEMORIES (SPMs ); (2) on-chip, attached to the system interconnect; (3) off-chip, driven by a DRAM memory controller. In addition, to analyze inter-processor communication behaviour, memories can be defined as private or shared, and a cache snooping mechanism is provided. The latency of each memory can be freely defined.

In terms of interconnect, MPARM provides a wide choice, spanning across multiple topologies (shared buses, bridged configurations, partial or full crossbars, NoCs) and both industry-level fabrics (AMBA AHB and AXI [14], STBus [26]) and academic research architectures such as the $\chi$pipes NoC described in this dissertation.

On top of the hardware platform, MPARM provides a port of the uClinux [1] and RTEMS [6] operating systems. The choice of RTEMS is motivated by the fact that RTEMS is a lightweight OS for embedded systems, but it offers at the same time good support for multiprocessing, and provides native calls for communication and synchronization in such multiprocessor environments.

Application code, either OS-based or not, can be easily compiled with standard GNU cross-compilers. Scripts and makefiles fully automate the process of building for a multiprocessor platform. Simple function calls, provided by support libraries of the simulator, allow flexible performance profiling: statistics can be collected during OS boot, application execution,or critical sections of algorithms.

MPARM also features support libraries to help fast development and debugging of new applications and benchmarks. This is key for establishing a solid and flexible simulation environment. MPARM includes several benchmarks from domains such as telecommunications and multimedia, and libraries for synchronization and message passing.

Debug functions include a built-in debugger, which allows to set breakpoints, execute code step-by-step and inspect memory content; it is additionally capable of dumping the full internal status of the execution cores. When testing applications written without underlying OS support (i.e., no native I/O calls are available), messages and status information can still be easily provided to the user by means of pseudo-instructions.

Multiple communication and synchronization paradigms are possible in MPARM, including plain data sharing on a shared memory bank, message passing among SPM resources of each processor, interrupts and semaphore polling.

Simulation accuracy and flexibility have to be traded off with simula tion speed. However, MPARM, despite being signal-accurate and cycle-accurate, is

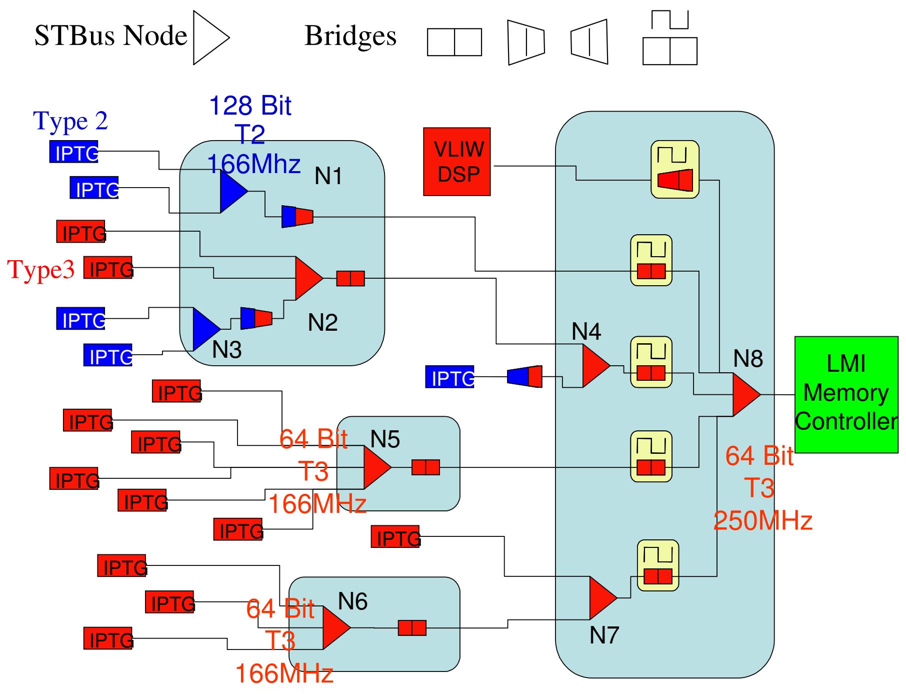


Figure 3.1: STBus reference MPSoC platform.

fast and usable. Simulation performance is in the range of 200 kCPUcycles/s, which is enough to simulate applications of reasonable complexity in few minutes.

In the context of this chapter, the MPARM features are key to interconnect performance evaluation. MPARM stimulates the communication subsystem with functional traffic generated by real applications running on top of real processors.

This opens up the possibility for communication infrastructure exploration under real workloads and for the investigation of its impact on system performance at the highest level of accuracy.

## 3.3 MPSoC platform modelling

State-of-the-art communication architectures are rapidly evolving along two main directions. On one hand, new communication protocols aim at a full exploitation of the physical bandwidth made available by the interconnect (e.g., masking slave access latencies). On the other hand, parallel topologies based on partial or full crossbars are progressively replacing simple shared busses when parallel communication flows need to be accommodated.

Industrial MPSoC platforms often implement multi-layer communication architectures, built up by composing basic interconnect units through bridges. Our modelling effort targets such complex multi-layer systems. In particular, we focus on an internally-developed MPSoC platform for consumer applications. Its mission-critical subset is illustrated in Fig.3.1.

The platform is made up of a number of IP Cores, grouped in several functional clusters, each one implementing functionalities like video stream decrypting and decoding, image resizing or more generic DMA tasks, and therefore features different combinations of data width, clock frequency and STBus protocol type. Proprietary STBus converters and adapters (named *GenConv*) are in charge of bridging the heterogeneous clusters, and make use of buffering resources to store bus requests, responses and outstanding transactions.

Communication requirements are also different across clusters. Some of them impose tight requirements on both system latency and bandwith. Some are more tolerant to latency but need garanteed bandwidth over a longer time window. Finally, the CPU is known to be particularly sensitive to latency in some conditions.

The system uses the traditional unified memory architecture with a single off-chip DDR SDRAM, which then becomes the target for the bulk of the bus transactions. This architectural template is frequently used to cut down on the cost for technology integration of embedded memories and when data footprint forces to store processing data in a large off-chip memory. In such an architecture, protocol intrinsic features can help releaving the bus utilization and so reduce congestions and contention points.

Message-based arbitration was set in STBus nodes, which poses a trade-off in our analysis: LMI-friendly traffic can be generated at the cost of a rigid fixed-priority arbitration scheme supported in message-enabled nodes. Messages allow several packets to be routed through a node without re-arbitration, thus avoiding the interleaving of SDRAM accesses addressing different memory banks. Message-based arbitration and store-and-forward management of write transactions are set in the GenConv components. Preliminary system simulations also allowed to size request, response and outstanding transaction FIFOs in the GenConv.

The ST220 VLIW DSP core was used in place of the general purpose processor. It is a 32 bit core running at 400 MHz, with 32kB instruction and 32kB data caches, which fetches data and instructions from the off-chip DDR SDRAM. For this purpose, the core is connected to an upsize (from 32 to 64 bit) and frequency (from 400 to 250 MHz) GenConv converter. In our simulation runs, the ST220 processor core will run a synthetic benchmark, namely matrix addition with variable size matrices. This benchmark was tuned to generate a significant amount of cache misses, whose performance was monitored throughout the entire simulation lifetime.

Finally, functional traffic generated by the most critical audio and video IP cores is reproduced by means of configurable traffic generators (*IPTGs*), which were programmed by means of STMicroelectronics-provided configuration files.

The whole platform was modelled and simulated with clock-cycle accuracy and a SystemC-based virtual platform described in Section 3.2 was used as the backbone environment [24]. The modelling effort resulted in a platform model with multiple abstraction levels. Finally, the modelling effort was completed by the integration of each sub-component into the complete platform and by setting up a statistics collection system.

A description of the main system component models follows.

### 3.3.1 STBus platform component models

The **STBus Interconnect** is the proprietary communication on-chip system developed at STMicroelectronics[26]. The components interconnected by an STBus *node* are either *initiators* (which initiate transactions on the bus by sending requests), or *targets* (which respond to requests).

STBus leverages two physical channels, one for initiator requests and one for target responses, and supports split transactions. While a system initiator is receiving data from an STBus target, another one can issue a second request to a different target. As soon as the response channel frees up, the second request can be immediately serviced, thus hiding target wait states behind those of the first transfer. The amount of saved wait states depends on the depth of the prefetch FIFO buffers at the target side.

Additionally, the split channel feature allows for multiple oustanding requests from the masters, with support for out-of-order delivery.

Single cycle arbitration in STBus paves the way for low latency transactions. First, a request/grant step between the initiator and the node takes place, corresponding to an atomic rendez-vous operation of the system. Then, the request is transferred from the node to the target, a response-request/grant step is carried out between the target and the node. Finally, the response is transferred from the node to the initiator.

STBus implements three different protocols featuring increasing complexity. Type 1 is the low cost implementation for low/medium performance. Type 2 introduces compound operations, source labeling, priority labelling and posted writes. Split and pipelined transactions are fully supported. Its simple design allows a synchronous handshake protocol and provides a limited transaction set. Type 2 (the basic protocol) extends the limited operation set of the peripheral interface to compound operations, source labeling, priority labelling and posted writes. Moreover, this implementation supports split and pipelined accesses. Finally, Type 3 provides the additional system efficiency associated with shaped request/response packets and the ability to support out-of-order transactions.

The **Generic Converter** is an STBus configurable block performing clock domain crossing, data with and STBus protocol type conversion. These functions can be performed standalone or in any combination within the same instance, according to the given configuration. Combining conversions has the advantage of minimizing the latency and the area with respect to having them managed separately.

The Generic converter also supports out-of-order response traffic and is optionally able to sort responses according to the initial request order (re-ordering). This capability is for example by default applied whenever T2/T3 STbus protocol conversion is required, as STBus T2 does not allow out-of-order responses.

**IPTG** is a SystemC block developed at STMicroelectronics aimed at reproducing the communication behaviour of a generic IP. In its simplest configuration, IPTG can generate bus traffic which obeys some statistical properties, i.e in terms of burst length, transaction types, addressing schemes, or it can also issue a transaction according to a specified sequence. Transactions can also be grouped to form a so-called "behaviour" (see Fig.3.2) and behaviours can then be executed multiple times or interleaved with one another in specific fashions. However, IPTG is best used to emulate the behaviour of complex real-life IPs: such IPs can be often seen as having a number of internal sub-process

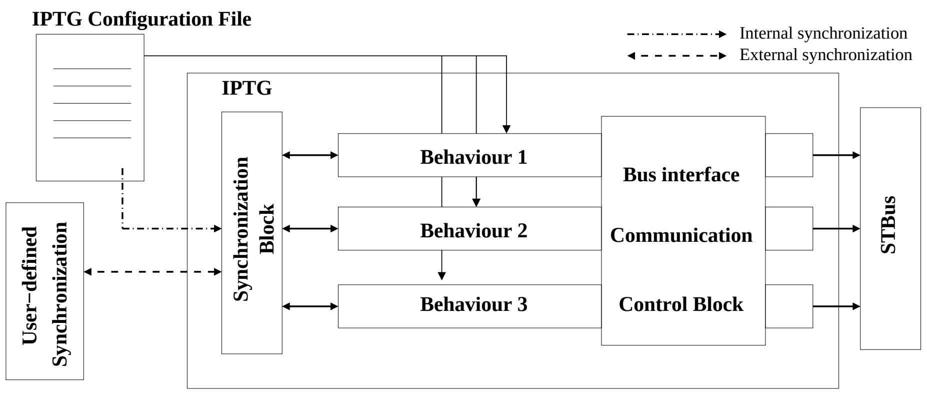


Figure 3.2: Traffic generator architecture.

(or agents), each one with its own characteristics (buffering space, transaction pipelining capability) but in some way dependent on each other (e.g., when operating in pipeline). With IPTG, each agent traffic is handled automatically according to its characteristics, and inter-agent synchronization points can be set to emulate dependencies between them. Once instantiated in a platform, IPTGs will generate bus transactions at different abstraction levels (transaction-level, bus cycle-accurate) according to what is specified in a per-IP configuration file, where all the required options and parameters are set. IPTG turns out to be a quite powerful and handy tool to the system integrator, as it allows to try out the SoC communication infrastructure in real-life conditions such as heavy-loaded transients which are not likely to be reproduced using random packet injection.

We developed a **memory controller** SystemC model and validated its functional correctness and timings with RTL signal waveforms on a cycle-by-cycle basis. The controller follows the internal LMI specification of SDRAM memory interface. The model includes a bus dependent and a bus independent part, thus easily allowing porting to several bus infrastructures. In this analysis, an STBus target interface is implemented. Input and output FIFOs allow storage of incoming packets or injection of outgoing packets into the bus. FIFO size and bus data width are tunable parameters.

In the bus independent part, bus-specific data (e.g., transaction identifier) are stripped off, while bus-agnostic data carrying memory access information (e.g., address, size, read/write flag, byte enable, write data) are kept and fed to an optimization engine. This latter performs memory access optimizations such as opcode merging and variable-depth lookahead, and generates the corresponding sequence of SDRAM commands (e.g., precharge, autorefresh, active, read, write) while meeting SDRAM timing specifications (e.g, TRAS, TCAS), which are model parameters. Data size and burst size are tunable as well. The controller can drive both SDR SDRAM and DDR SDRAM memory devices.

Finally, latencies incurred for memory controller operation were back-annotated in our model from real-life ST memory controllers following LMI specification. Similarly, our model was validated by comparing SystemC signal traces with those provided by ST, and we found an *exact matching* for all tested memory access patterns.

### 3.3.2 Architectural variants

We extended the modelling capability of the virtual platform described so far in order to explore architectural variants and assess the interaction between the communication architecture and the other MPSoC subsystems.

On one hand, we ported new communication protocols (AMBA AHB and AMBA AXI) and made them inter-operate with the traffic generators and the off-chip memory controller, so to keep the same platform template. Since in the real system both audio and video IP cores and the memory controller natively come with STBus interface, we developed bridges for protocol conversion purposes, and accounted for the corresponding latencies. On the other hand, we replaced the off-chip SDRAM memory with an on-chip shared memory, so to test the system with a memory core featuring a cheaper access cost. Finally, we developed several platform instances (which we call the *collapsed* variants) where the most heavily congested cluster (node N5 in Fig.3.1) is removed and its communication actors attached to the central N8 cluster. This way, we intend to compare a centralized versus a distributed solution for the communication architecture, thus spanning the bus access versus multi-hop latency trade-off.

The design of these architectural variants required the development of new component models, which are hereafter briefly described.

The **AMBA AHB** [14] system backbone consists of a shared communication channel connecting multiple system cores. The channel is composed of two split and unidirectional data links (one for reads, one for writes), but only one of them can be active at any time, thus preventing the multiplexing of requests and responses on the interconnect signals. Transaction pipelining (*i.e.* split ownership of data and address lines) is supported to provide for higher throughput but not as a means of allowing multiple oustanding transactions. In practice, each transaction consists of an address and a data phase. While the data phase of the $i-$th transfer is in progress, the address for the $i+1-$th transfer can be anticipated on the address lines. However, address sampling is only allowed when the data phase completes successfully, and if this latter takes a few cycles to complete, the configuration of the address lines is frozen and their sampling postponed as well. As a consequence, the AHB-compliant transaction pipelining cannot be considered as a means of allowing multiple oustanding transactions. Bursts are supported by AHB masters and arbiter as a way to amortize arbitration time, and the non-posted paradigm for write transactions is implicitely assumed. The SystemC model of the AHB interconnect we developed does not implement split transactions. The AHB protocol can also be used on top of parallel topologies. The Multi-Layer AHB is an interconnect specification that enables parallel access paths between multiple masters and slaves by means of an interconnection matrix, and gives the benefit of increased overall bandwidth and flexible system architecture. Single-master AHB layers at the input of the matrix do not even need to support arbitration nor split/retry transactions (AHB-Lite protocol).

**AMBA AXI** [14] builds upon the concept of point-to-point connection. Therefore, it can be used to connect (i) a communication initiator to a bus, (ii) a communication target to a bus, or (iii) directly an initiator to a target. In practice, the connection between any two devices translates into the instantiation of a master interface and of the symmetrical slave interface. Five different logical monodirectional channels are provided in AXI interfaces, and activity on

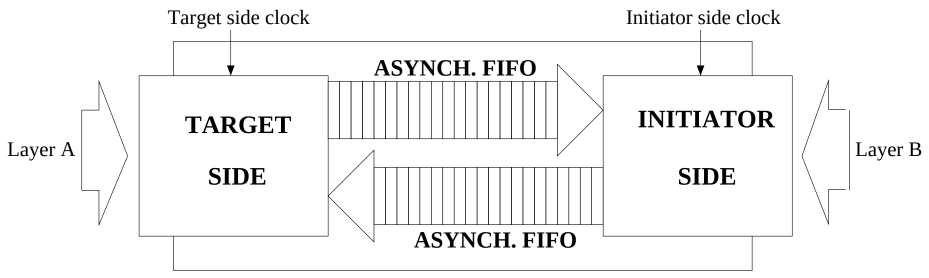


Figure 3.3: Generic scheme for hybrid bridges.

them is largely asynchronous and independent (2 address channels, a read data and a write data channel, and a channel for write responses). This allows to support multiple outstanding transactions (with out-of-order or in-order delivery selectable by means of transaction IDs). Other advanced features include burst transactions with only the first address issued and the support for register insertion for timing closure transparent to the protocol. The interface definition enables a variety of different interconnect implementations, which are not addressed by the AXI specification. In particular, single shared address and data lanes could be used, or alternatively multiple data lanes. This stems from the consideration that in most systems, the address channel bandwidth requirement is significantly less tha the data channel bandwidth requirement. AMBA AXI modelling was based upon the SystemC libraries provided within the Synopsys CoCentric/DesignWare suites. Such systems can achieve a good balance between system performance and interconnect complexity by using a shared address bus with multiple data busses to enable parallel data transfers. To the limit, designers could opt for a multi-layer approach, with multiple address and data busses [31].

We modelled in SystemC a number of **bridges**: AHB-AHB, AXI-AXI, AHB-STBus, AXI-STBus, AHB-AXI, STBus-AHB, STBus-AXI. Their generic architecture is reported in Fig.3.3, where we identify a target side, an initiator side and asynchronous FIFOs providing support for different clock domains. The developed bridges have some common features: (i) they handle write transactions in a store-and-forward fashion, (ii) they have a blocking target side in presence of read transactions and (iii) they have tunable latency. Frequency, datawidth and protocol conversion latencies of the above bridges were set to be equal to those of the GenConv for STBus. Although introducing the same latencies, our bridges were not designed to be competitive with the highly optimized STBus-STBus ones. They rather implement basic bridging functionality and do not exploit some advanced features of the communication protocols. For instance, they are always blocking on read transactions. This does not affect our results, since our goal is not a crossbenchmarking of communication protocols, but rather an analysis of how several communication protocol features, interconnect fabric topologies and memory architecture configurations combine together to determine global performance metrics. From this viewpoint, deploying lightweight bridges even for advanced communication protocols saves power and area (a typical GenConv bridge performing frequency conversion between T3 nodes at 64 bits can be as large as an STBus node with 5x3 crossbar topology at 64 bits), but penalizes across-layer communications. Therefore, it will be our interest to find out how sensitive system performance is with respect to bridge functionality.

## 3.4 Interaction between communication and memory sub-systems

Since system performance in a multi-layer architecture results from the interaction of traffic patterns at many congestion points, we first analyze the single layer scenario, and get indications that pave the way for a better understanding of the multi-layer scenario.

### 3.4.1 Single-layer architecture

#### Many-to-many traffic pattern

The work in [27] analyzed single-layer shared busses with many slave devices attached, and proved the effectiveness of STBus and AXI in handling parallel communication flows. While memory wait states translate into idle cycles for AMBA AHB, STBus and AXI are able to mask them. STBus leverages two physical channels, one for request packets and one for response packets. While one response phase is in progress, one or more new requests can be propagated across the request path, so that another slave device can be accessed in parallel. Its access latency is then masked by the response phase of the previously addressed slave. Instead, AXI can arbitrate its internal data lanes on a word-by-word basis, therefore the access latency of one slave device is masked by the data transfer of another slave device.

AHB is an effective solution for bus utilizations under 40%, where its low cost implementation still guarantees competitive performance. More advanced busses, such as STBus, are needed to deal with bus utilizations ranging from 40% to 80%. STBus is able to initiate new transactions by addressing one/more slave devices while the previous one is performing its response phase, thus hiding the latency of this latter. Finally, AXI can interleave the beats of different burst transactions on its internal data lanes (fine-granularity arbitration mechanism) and can leverage 5 independent and almost asynchronous physical channels. This makes AXI too complex to deal with lightweight traffic conditions, but very effective in presence of bus congestion above 80%.

These results adding new awareness of the fact that the multiple physical resources employed by STBus and AXI, together with their advanced arbitration mechanisms, are at the core of their ability to handle parallel communication flows.

#### Many-to-one traffic pattern

Here we still focus on single-layer shared busses, but with a single slave device, hence accommodating a many-to-one traffic pattern. This reflects the architecture of the clusters in Fig.3.1. In this experiment, all IPTGs generate bursty read accesses to the shared memory (an on-chip core with 1 wait state), however we proved the independence of the results from the mix of bus transactions.

In this memory-centric scenario, the maximum bus efficiency is posed by the memory controller, since there are no opportunities to parallelize access patterns to different memory devices. With our simple controller, the response data channel in each communication architecture (carrying read data) is forced to work with 50% efficiency: 1 data transfer followed by 1 idle cycle (corresponding to 1 memory wait state). What can degrade such efficiency is the handover overhead incurred by the communication protocols. Let us consider the handover between two consecutive burst read transactions. While the last read data word of the first burst is injected into the bus by the slave device, the new address associated with the next burst should be concurrently driven by the next master actor on the bus. This way, the new memory access can be readily initiated on the next clock cycle.

AMBA AHB can hide bus handover overhead by changing the HGRANTx signals when the penultimate address in a burst has been sampled. The new master can then readily drive the bus address lines without any handover overhead. Interestingly, the many-to-one traffic pattern is the best operating condition for AMBA AHB. STBus achieves the same performance but with a different mechanism. Transactions are split in a request phase and a response phase, during which a request (response) packet is sent through the bus. Each packet is then mapped onto the physical interface as a series of smaller units called cells. Each cell is transferred with a request/grant handshake, and the final cell in the packet is marked. The target asserts a grant signal to the next initiator in the same cycle it provides the last response data unit to the previous initiator. Since this grant signal is propagated asynchronously from the target to the waiting initiator through the STBus node in the same clock cycle, the next transfer can start immediately without handover overhead and the response channel can be driven with 50% efficiency. Finally, AXI is able to sustain the required efficiency by means of the burst overlapping mechanism. A master can drive another burst address after the slave accepts the address of the previous burst transaction. This enables the slave to begin processing data for the second burst in parallel with the completion of the first burst or, at least, to have the control information for the next transfer readily available for sampling. Therefore, even the internal read data lane of an AXI bus can be operated with 50% efficiency.

Given the above considerations, our simulations did not show significant differences between performance of the communication architectures in the single-layer, single-slave scenario, hence results are not reported here. In this context, busses should be able to sustain the transfer efficiency posed by the memory controller. AMBA AHB is surprisingly as efficient as the other protocols when a 50% maximum efficiency is required. More complex memory controllers may implement buffering and perform optimizations on queued transactions, hence at least split transaction support would be required to the communication protocols. These findings apply well to set-top-box platforms, where shared memory/external devices represent bottleneck vertices to which most of the communication takes place.

### 3.4.2 Multi-layer architecture

Let us now extend our analysis to the complete MPSoC platform described in section 3.3. IPTGs reproduce traffic patterns of real-life IP cores.

As regards the memory architecture, we still consider a simple memory con-

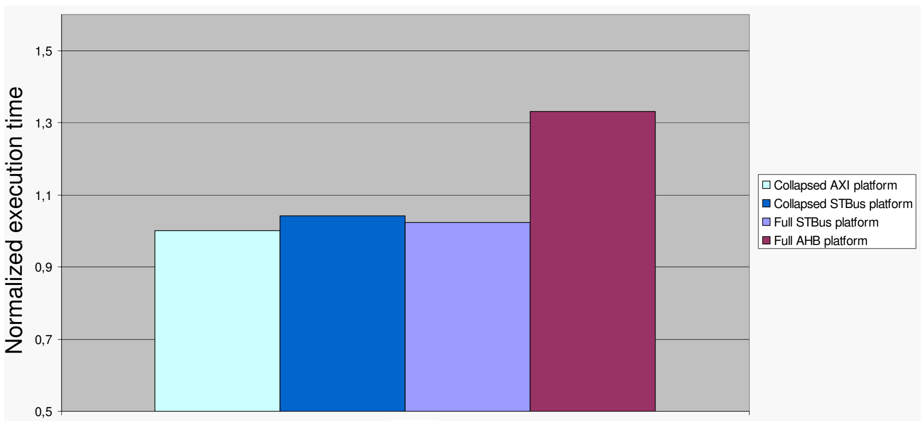


Figure 3.4: Performance of MPSoC platform instances.

troller driving an on-chip shared memory with 1 wait state and postpone the analysis with LMI memory controller.

The first two bars in Fig.3.4 compare the normalized execution time of the collapsed platform instances, since this makes the role of the bridges and initiator interface complexity negligible. AXI and STBus collapsed variants exhibit almost the same performance, thus confirming the findings of subsection 3.4.1. In fact, the collapsed communication architectures resemble the single-layer single-slave scenario, where all interconnects were showed to perform almost the same.

The third bar in Fig.3.4 allows to compare the single-layer STBus with the full (multi-layer) STBus platform. Again, the two solutions show negligible differences in their performance metrics. The multi-layer approach introduces distributed buffering and relies on multiple outstanding transaction support of its master bus interfaces. The longer transaction latency associated with crossing the master-to-slave path in the multi-layer architecture is compensated by the initiators' capability to initiate new transactions before the previous ones complete. The resulting performance is equivalent to that of a single-layer architecture, where the multiple oustanding transaction capability does not help much since the target interface has a single-slot buffering here. Therefore, each transaction is blocking, which compensates the benefits of a shorter master-to-slave path.

The performance ratio between collapsed and distributed interconnect solutions however changes if the memory device gets progressively slower in responding to access requests. Fig.3.5 clearly shows the increasing advantage of distributed solutions as the memory latency increases. Please note that the use of AXI and STBus is interchangeable here, what really matters is the architecture topology. A fast memory penalizes communication architectures with large crossing latencies. In contrast, a slow memory makes distributed solutions preferrable, since the distributed buffering allows multiple outstanding transactions capable bus interfaces to keep pushing transactions into the bus. The master-to-slave multi-hop path gets therefore filled, and the system is able to deliver high throughput.

However, collapsed solutions are not always feasible. First, physical limita-

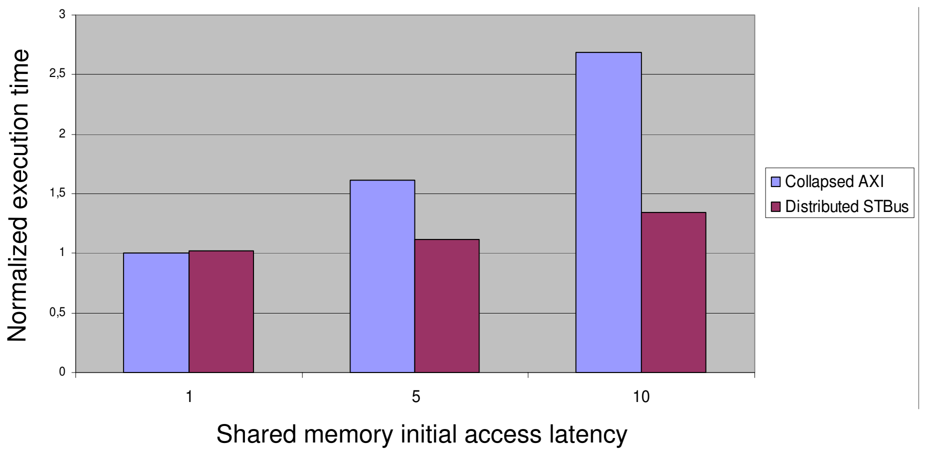


Figure 3.5: Performance of distributed vs centralized communication architectures as a function of memory speed.

tions as well as layout and routability constraints prevent from connecting tens or hundreds of actors on the same interconnection layer. Moreover, communication actors with similar interfaces (e.g., datawidth, protocol type) may have to be grouped in the same cluster, thus cutting down on the number of required adapters. Finally, clusters of homogeneous cores favour performance closure. If we restrict our analysis only to distributed architectures, we can easily realize that bridges become a key component for system performance. In fact, from Fig.3.4 we can compare performance of multi-layer STBus and AHB architectures. Please note that this is the best operating condition for AHB, since the memory device responds with just 1 clock cycle delay. We can observe that AHB solution is ineffective, due to the fact that AHB-AHB bridges are blocking on each transaction. The source node of the bridges are non-split AHB layers, and are therefore blocked until the transaction in progress is completed.

The blocking behaviour of the bridge may depend either on intrinsic limitations of the protocol semantics or on low-cost implementations of the bridges. For instance, we realized lightweight AXI-AXI bridges as illustrated in Fig.3.3 with blocking target side on read transactions. In fact, while write transactions can be easily managed by means of a store-and-forward policy, implementing non-blocking read transactions has a heavier impact on bridge complexity (control information must be stored and reassociated with response data, read requests might be serviced in a different order than they were received, etc.). Because of this lightweight bridge design, we found a performance for the distributed AXI platform almost equivalent to the full AHB platform. Therefore, advanced features of AXI, which could potentially reach the same performance level of STBus, are vanished by poor bridge functionality. For this reason, bridge engineering is becoming increasingly complex as advanced features are introduced in communication protocols, and they can be viewed as true IP blocks. In our case, the availability of proprietary STBus bridges made performance of STBus distributed platforms hardly achievable by the other distributed schemes.

The final scenario which is left to investigate is the multi-layer architecture with LMI memory controller and off-chip DDR SDRAM memory in place of the

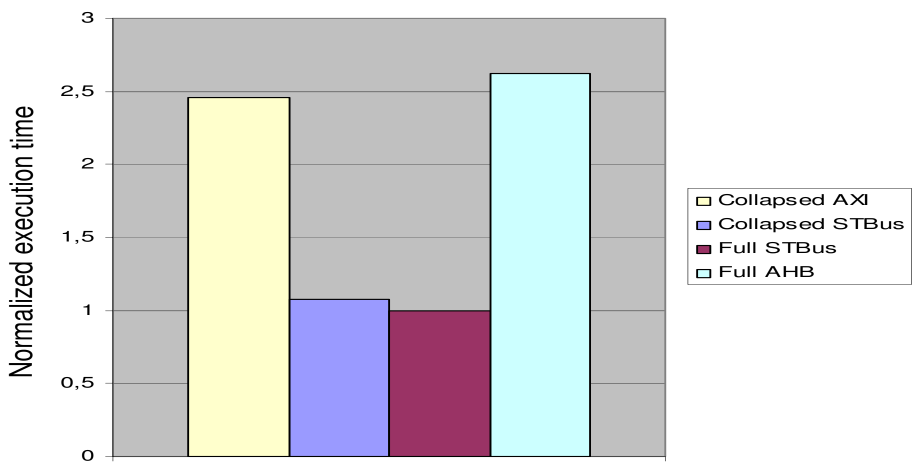


Figure 3.6: Performance of platform instances with LMI memory controller.

on-chip shared memory. The main differences lie in the higher response latency of the memory sub-system (11 cycles to get the first read data word since the request was sampled), in the multi-slot FIFO implemented in the memory bus interface and in the optimizations performed on queued transactions (opcode merging, lookahead) by the memory controller. Since the LMI controller natively exhibits an STBus bus interface, a bridge is required to connect it to other interconnect fabrics. This bridge must be able to implement split transactions, i.e. not to block the bus when a new transaction is sampled and until the associated memory access is completed. Otherwise the input FIFO of the memory controller would never contain more than one pending transaction, and no optimizations could be performed. Fig.3.6 illustrates execution time of various platform instances. Since the memory response latency is high, we expect distributed platforms to outperform collapsed ones. However, collapsed AXI is much worst than collapsed STBus, since (i) this latter does not require a bridge, (ii) the multiple oustanding transaction capability of STBus initiators can be fully exploited to fill in the FIFO of the memory bus interface and (iii) thus memory controller optimizations can be exploited. In contrast, collapsed AXI was using a simple protocol converter unable to perform split transactions. For these reasons, collapsed STBus can approach the performance of distributed STBus.

If we restrict our analysis to distributed solutions, we notice that the performance gap between STBus and AHB has increased a lot with respect to Fig.3.4, due to the higher latency of the memory sub-system, which makes the penalty associated with non-split blocking bridges even more severe.

## 3.5 Fine-grain platform performance analysis

The memory controller drains the bulk of all bus transactions in the platform of Fig.3.1. Therefore, properly monitoring the behaviour of the bus-memory controller interface can help system designers identify where bottlenecks are. For instance, should low bandwidth communication be monitored at the I/O

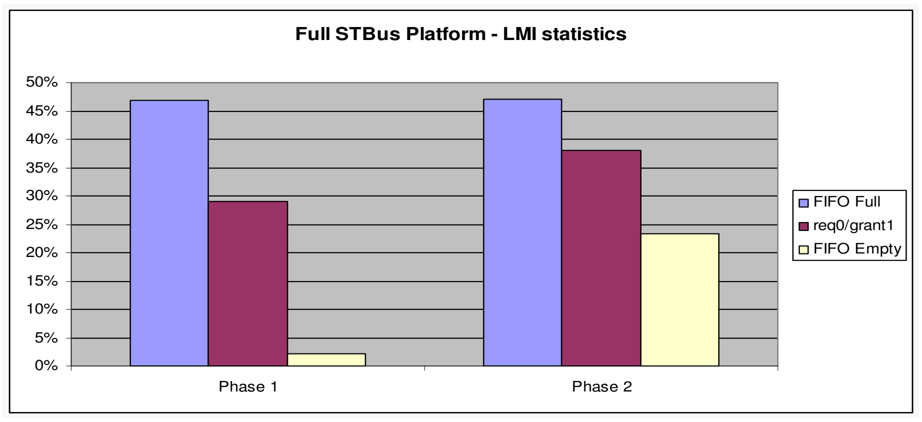


Figure 3.7: LMI statistics for the full STBus platform.

interface, this might be due to the actual inefficiency of the memory controller or to the poor performance of the system interconnect. Our modelling environment enables this kind of analysis at a fine granularity.

As an example, we report in Fig.3.7 the statistics taken at the bus interface of the LMI memory controller in our full STBus-based MPSoC platform. Two working regimes are illustrated out of the MPSoC application lifetime. During the first execution phase, the FIFO of the bus interface is full for 47% of the time, while it is available to store new transactions for the remaining 53%, partitioned as follows: for 29% of the time there are no incoming requests (request signal is 0 while grant signal is 1), and for remaining 24% the bus interface is storing new memory access requests. The FIFO is empty only for a marginal time fraction. We conclude that this execution phase exhibits intensive memory traffic which the interconnect is able to handle pretty well.

During the second phase, the time percentage during which the FIFO is full remains unaltered, while the FIFO is empty for a longer time. Therefore, the new traffic pattern has a lower intensity than the previous one on average, but is it more bursty. We repeated the same test for a full AHB platform, and found that the FIFO is never full (since our AHB implementation does not support split transactions) and that for 98% of the time there are no incoming requests. This clearly indicates that the system interconnect is the performance bottleneck, and not the memory controller.

## 3.6 Summary

We now turn our analysis findings into guidelines for designers of state-of-the-art industrial MPSoCs:

1. For single-layer systems, a significant performance differentiation between different communication protocols can be observed only when they have to deal with a many-to-many traffic pattern. In this context, advanced interconnets are able to hide slave response latency by processing parallel communication flows. This is achieved by means of a fine granularity of arbiter decisions (e.g., rearbitration of data links on a cycle-by-cycle ba-

sis) and of multiple physical resources (multiple communication channels, buffering in the bus and/or in the bus interfaces).

2. In single-layer systems with a centralized slave, the performance of this latter and of its control logic bounds the maximum performance that communication protocols can achieve. This upper bound depends on the slave response latency, on the amount of buffering implemented at its bus interface and on the optimizations which might be performed by its controller. When the required bus efficiency is low (e.g., 50%), simple interconnect fabrics may provide the same performance of advanced and more complex communication infrastructures.

3. In the absence of technology constraints, a distributed multi-layer system interconnect results in significant performance speed-ups with respect to centralized ones only if **(i)** initiator bus interfaces support multiple outstanding transactions **(ii)** the target side of bridges is able to handle split/non-blocking transactions **(iii)** the response latency of target devices is long enough with respect to the data transport latency across a multi-hop interconnect.

4. As long as the features of the previous point are supported by a distributed system interconnect, performance differentiation of competing communication protocols is only marginal in presence of a centralized target bottleneck. On one hand, this puts emphasis on the generation of memory controller-friendly traffic at the initiators rather than on the optimization of the system interconnect. On the other hand, this calls for optimizations of the I/O architecture to remove the system bottleneck.

5. Bridges are becoming true IP blocks. The introduction of new features in communication protocols might be vanished by the deployment of lightweight bridges with basic functionality. More research is needed to understand whether it is really worth increasing bridge complexity, instead of keeping lightweight bridges for path segmentation and traffic routing and pushing complexity at the system interconnect boundaries, which is known as the network-on-chip solution.

6. The availability of complete modelling and simulation frameworks like the one developed for this work makes it easier to accurately identify system bottlenecks and to fine-grain tune the architecture for the application domain of interest. For instance, we have showed how to identify working conditions during application lifetime and how to discriminate between poor performance of the memory controller and of the communication architecture.

## 3.7 Conclusion

Traditionally, MPSoC interconnects have been based on the shared bus concept [14]. In other words, a bunch of wires would be laid among all the IP cores in the design. Only one pair of devices would be allowed to communicate over this bus at any point in time; access to the shared medium would be regulated by

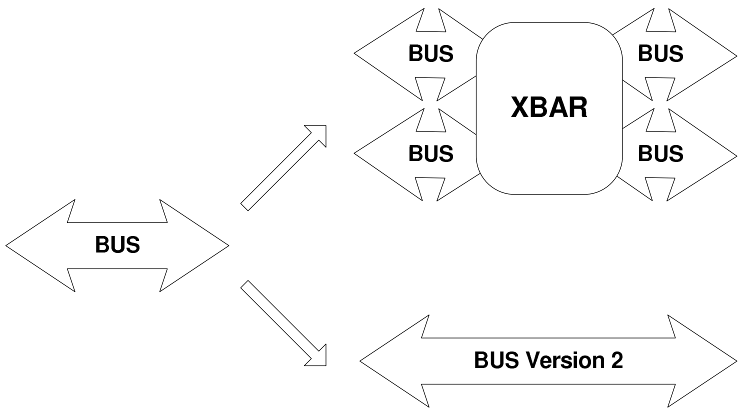


Figure 3.8: Evolution of shared buses towards hierarchical buses and more advanced protocols.

arbitration, either based on fixed priorities, on time slots, randomly, or on other criteria.

Shared buses have as a main advantage their extreme simplicity, both in conceptual terms and circuit design terms. However, they are completely unsuitable for next-generation MPSoCs, due to two fundamental limitations:

- Their maximum available bandwidth is capped by their shared nature, and this limit can easily be trespassed when the number of attached cores becomes more than a few.
- Their electrical performance degrades dramatically with new lithographic nodes. Since a shared bus is necessarily a structure composed of global wires, as per the INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS (ITRS) [3], its propagation delay actually increases with miniaturization. Therefore, with each new chip generation, a shared bus becomes slower in operating frequency, and even slower when compared to the progress in speed achieved by logic blocks. Long wires are also more vulnerable to crosstalk, variability and electrical noise, all of which represent increasingly serious problems in current technologies.

In response to these issues, buses have undergone evolutions [14, 26, 7] in two respects: protocols and topologies (Figure 3.8).

Protocol evolution allows for more sophisticated handshakes occurring on the bus, such as multiple outstanding transactions, out-of-order retirement of responses, burst requests, smarter arbitration, etc.. These evolutions help in making the best possible use of the limited available bandwidth. While useful in temporarily reducing the extent of bandwidth issues, they still do not provide a long-term solution to the fundamental limitations of buses.

Topology evolutions are a more radical departure from the original shared bus paradigm. The main principle is to deploy multiple buses, attached to each other by bridges or elements called crossbars - i.e., devices providing full simultaneous connectivity among all their inputs and all their outputs. The outcome is often called hierarchical bus or multilayer bus. Hierarchical buses are a much better response to MPSoC design concerns, and in fact most MPSoCs today leverage hierarchical buses.

Even despite these improvements, buses are still a sub-optimal solution for next-generation MPSoCs, due to several factors:

- Hierarchical buses are mostly a manual workaround, by means of which designers try to fix the issues they are presently facing. The development and verification steps have to be performed mostly manually, and it is hard to guarantee that the design will scale upon the addition of more IP cores in the next revision of the design. Issues such as deadlock prevention, address mapping, and compliance with performance objectives are among the challenges left to designers.

- From the physical design point of view, hierarchical buses are not much better than shared buses. While allowing for some wire segmentation (wires only have to span regions of the whole system), wires are still normally laid with wide parallelism (typically more than 100 wires for a 32-bit bus and possibly close to 200 for a 64-bit bus), and they still connect multiple entities (a large fanout). Together, these issues mean that buses are still electrically inefficient, and difficult to route during physical design.

- Buses normally involve interaction among three agents, usually called master, slave and arbiter, instead of providing a point-to-point, one-to-one handshake. This is unnecessarily making system integration harder.

# Bibliography

[1] The uclinux embedded linux/microcontroller project
http://www.uclinux.org/

[2] Systemc community, 2003.
http://www.systemc.org

[3] SIA Semiconductor Industry Association.
The international technology roadmap for semiconductors.
SIA Semiconductor Industry Association, 2002.
Tech. Rep.

[4] D. Bertozzi L. Benini M. Loghi, F. Angiolini and R. Zafalon.
Analyzing on-chip communication in a mpsoc environment.
Proceedings of the conference on Design, automation and test in Europe, February 2004.
IEEE Computer Society.

[5] F. Poletti D. Bertozzi L. Benini M. Ruggiero, F. Angiolini and R. Zafalon.
Scalability analysis of evolving soc interconnect protocols.
roceedings of the 2004 International Symposium on System-on-Chip, 2004.
pp. 169-172.

[6] On-Line Application Research (OAR).
Rtems, open-source real-time operating system for multiprocessor systems, 2002.

[7] D. Wingard.
Micronetwork-based integration for socs.
Proceedings of Design Automation Conference (DAC), 2001.
pp. 673-677.

[8] K.Lahiri, A.Raghunathan, G.Lakshminarayana,
"The LOTTERYBUS on-chip communication architecture",
Trans. on VLSI Systems, Vol.14, no.6, pp.596-608, June 2006.

[9] S. Pasricha, Y. Park, F. Kurdahi, N. Dutt,
"System-Level Power-Performance Trade-Offs in Bus Matrix Communication Architecture Synthesis",
CODES+ISSS 2006.

[10] S. Pasricha, N. Dutt,
"COSMECA: Application Specific Co-Synthesis of Memory and Communication Architectures for MPSoC",
Design Automation and Test in Europe, pp.700-705, 2006.

[11] S.Pasricha, N.Dutt, M.B.Romdhane,
"Constraint-driven bus matrix synthesis for MPSoC",
Asia South Pacific Design Automation Conf., pp.30-35, 2006.

[12] S. Murali and G. De Micheli,
"An Application-Specific Design Methodology for STbus Crossbar Generation",
Design Automation and Test in Europe, pp. 1176-1181, 2005.

[13] S. Pasricha, N. Dutt, E.Bozorgzadeh, M. Ben-Romdhane,
"FABSYN: Floorplan-aware Bus Architecture Synthesis"
IEEE Trans. on VLSI Systems, Vol.14, no.3, pp.241-253, March 2006.

[14] ARM Ltd., Sheffield, U.K., AMBA 2.0/3.0 Specifications.
Available: http://www.arm.com/armtech/AMBA

[15] Siemens AG, Open Microprocessor Initiative, OMI 324 PI Bus,
Rev 0.3d 1994, OMI Standards Draft.

[16] Sonics Inc., Sonics Integration Architecture.
Available: http://www.sonicsinc.com

[17] J. Turner and N. Yamanaka,
Architectural choices in large scale ATM switches,
IEICE Trans. Commun., vol. E-81B, no. 2, pp.120-137, Feb. 1998.

[18] Sekar, K.; Lahiri, K.; Raghunathan, A.; Dey, S.;
"Integrated Data Relocation and Bus Reconfiguration for Adaptive System-on-Chip Platforms",
Design Automation and Test in Europe, Vol.1, pp.1-6, march 2006.

[19] F. Polloni et al.,
"Fast System-Level Design Space Exploration for Low Power Configurable Multimedia System-on-Chip",
15th IEEE Int. ASIC/SoC Conference, pp 150-154, Sep. 2002.

[20] F.Poletti, D.Bertozzi, A.Bogliolo, L.Benini,
"Performance analysis of arbitration policies for SoC communication architectures",
Journal of Design Automation for Embedded Systems, pp.189-210, 2003.

[21] A. Wieferink et al.,
"System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms",
IEE Proc. on Computers and Digital Techniques, Vol. 152, Issue 1, pp.3-11, 2005.

[22] Chulho Shin et al.,
"Fast exploration of parameterized bus architecture for communication-centric SoC design",
Design Automation and Test in Europe, pp.352-357, Vol.1, 2004.

[23] Xinping Zhu; Wei Qin; Malik, S.;
"Modeling operation and microarchitecture concurrency for communication architectures with application to retargetable simulation",
IEEE Trans. on VLSI Systems, Volume 14, Issue 7, pp.707 - 716, 2006.

[24] Loghi, M.; Angiolini, F.; Bertozzi, D.; Benini, L.; Zafalon, R.;
"Analyzing on-chip communication in a MPSoC environment",

Design Automation and Test in Europe, pp. 752-757, Vol.2, 2004.

[25] IBM Corporation, Armonk, NY, CoreConnect Bus Architecture.
Available: http://www.chips.ibm.com/products/coreconnect/.

[26] ST Microelectronics, STBus Interconnect.
Available: http://www.st.com/stonline/prodpres/dedicate/soc/cores/stbus.htm

[27] M. Ruggiero, F. Angiolini, F. Poletti, D. Bertozzi, L. Benini, R. Zafalon,
"Scalability Analysis of Evolving SoC Interconnect Protocols",
Int. Symposium on System-on-Chip, Nov 16-18, 2004, pp. 169-172.

[28] N.Wang, M.A. Bayoumi,
"Dynamic fraction control bus: new SOC on-chip communication architecture design",
IEEE Int.SoC Conference, pp.199- 202, 2005.

[29] Open Core Protocol International Partnership (OCP-IP).
Available: http://www.ocpip.org

[30] Crossbow Technologies.
Available: http://www.crossbowip.com

[31] Synopsys CoCentric
"http://www.synopsys.com",
2004.

[32] S.Murali, M.Coenen, A.Radulescu, K.Goossens,
"A methodology for mapping multiple use-cases onto networks-on-chips",
Design Automation and Test in Europe, pp.118-123, 2006.

# Chapter 4

# Network-on-Chip: An Interconnect Fabric For MPSoCs

## 4.1 Introduction

The increasing integration densities made available by shrinking of device geometries will have to be exploited to meet the computational requirements of applications from different domains, such as multimedia processing, high-end gaming, biomedical signal processing, advanced networking services, automotive or ambient intelligence. Interestingly, the request for scalable performance is being posed not only to high-performance microprocessors, which have been tackling this challenge for a long time, but also to embedded computing systems. As an example, systems designed for ambient intelligence are increasingly 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 has to be addressed at manageable power levels and affordable costs [16].

Such a performance cannot be provided by a single processor core, but requires a heterogeneous on-chip multi-processor system containing a mix of general-purpose programmable cores, application specific processors and dedicated hardware accelerators. In order for the computation scalability provided by multi-core architectures to be effective, the communication bottleneck will have to be removed.

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

A scalable communication infrastructure that better supports the trend of SoC integration consists of an on-chip micro-network of interconnects, generally known as *Network-on-Chip (NoC)* architecture [23, 30, 36]. The basic idea is borrowed from the wide-area networks domain, and envisions on-chip networks on which packet-switched communication takes place, as depicted in Fig.4.1 Cores access the network by means of proper interfaces, and have their packets forwarded to destination through a certain number of intermediate hops (corresponding to switching elements).

The modular nature of NoC architectures and the enormous communication bandwidth they can provide leads to network-centric multi-processor systems (MPSoCs) featuring high structural complexity and functional diversity. While in principle attractive, these features imply an extension of the design space that needs to be properly mastered by means of new design methodologies and tool flows [31].

A NoC design choice of utmost importance for global system performance concerns topology selection. Topology describes the connectivity pattern of networked cores, and heavily impacts the final throughput and latency figures for on-chip communication. In particular, these figures stem from the combination of the theoretical properties of a NoC topology (e.g., average latency, bisection bandwidth) with the quality of its physical synthesis (e.g., latency on the express links of multi-dimension topologies, maximum operating frequency). Several researchers [36, 29, 31, 13] envision NoCs for the general purpose computing domain as regular tile-based architectures (structured as mesh networks or fat trees). In this case, the system is homogeneous, in that it can be obtained by the replication of the same basic set of components: a processing tile with its local switching element.

However, in other application domains, designers need to optimise performance at a lower power cost and rely on SoC component specialization for this purpose. This leads to the on-chip integration of heterogeneous cores having varied functionality, size and communication requirements. In this scenario, if a regular interconnect is designed to match the requirements of a few communication-hungry components, it is bound to be largely over-designed for the needs of the remaining components. This is the main reason why most of current heterogeneous SoCs make use of irregular topologies [11, 22].

This chapter introduces basic principles and guidelines for the NoC design. At first, the motivation for the design paradigm shift of SoC communication architectures from shared busses to NoCs is examined. Then, the chapter goes into the details of NoC building blocks (switch, network interface and switch-to-switch links), presenting their design principles and the trade-offs spanned by different implementation variants. Readers will be given the opportunity to become familiar with the theoretic notions by analysing a few case studies from real-life NoC prototypes. For each of them, the design objectives leading to the specific architecture choices will be illustrated.

Then, the key issue of topology selection will be discussed with reference to both the general purpose and the application specific computing domains. This is a high-impact decision in the NoC (and system) design process, and will be devoted ample room in this chapter. Finally, the main challenges that research

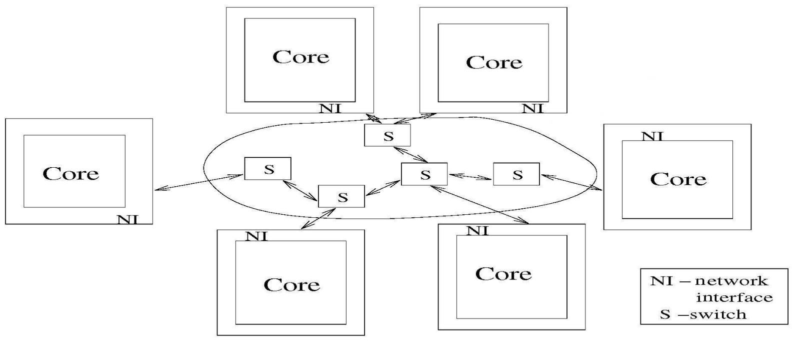


Figure 4.1: A generic network-on-chip architecture.

has to tackle in order for NoCs to become mainstream will be briefly discussed.

## 4.2 Design Challenges For On-Chip Communication Architectures

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

### 4.2.1 Technology challenges

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 [23]. 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. In this direction, recent communication protocols such as AMBA AXI pose no requirements for a fixed relationship between the various channels the protocol is structured into. This way, the insertion of a register stage in any channel is made possible, at the cost of an additional cycle of latency [2]. By breaking long timing paths, the maximum operating frequency can be preserved. NoCs are multi-hop architectures pushing this trend to the limit: an aggressive path segmentation enables an operating frequency which is much higher than that of state-of-the-art interconnect fabrics and even of attached communicating cores. The potentially higher speed of NoCs can also be an indirect means of masking their inherently higher communication latency.

Another technology related issue concerns performance predictability. This can be defined as the deviation of post-layout performance figures from post-synthesis ones. The traditional flow for standard cell design features logic synthesis and placement as two clearly decoupled stages. Wire load models make this splitting possible. They consist of pre-characterized equations, supplied within technology libraries, that attempt to predict the capacitive load that a gate will have to drive based on its fan-out alone. A gate driving a fan-out of two other gates is very likely to be part of a local circuit. Thus, its capacitive load is little more than the input capacitance of the two downstream gates. A gate with a fan-out of one thousand is likely to be the driver of a global network. Therefore, some extra capacitance is expected due to the long wires needed to carry the signal around. This assumption works very well as long as wire loads do not become too large. Otherwise, the characterization of wire load models becomes very complex, and the prediction inaccuracies become critical. It was showed in [15] that a state-of-the-art AMBA AHB Multi-Layer interconnect is ineffectively handled by this synthesis flow already at the 130nm technology node: after placement, the actual achievable frequency decreases by a noticeable 17%. This drop means that some capacitive loads, which were not expected after the logical synthesis, arose due to routing constraints. An explanation can be found in the purely combinational nature of the Multi-Layer fabric, which implies long wire propagation times and compounds the delay of the crossbar block. In contrast, the same work in [15] proves that an on-chip network incurs a negligible timing penalty of less than 3% after taking actual capacitive loads into account. This confirms that NoC wire segmentation is highly effective and results in wire load predictability. However, even for NoCs the above synthesis flow proves substantially inadequate as technology scales further down to the 65 nm node. The origin of the problem lies in the unacceptable inaccuracy in wire load estimation. Even when synthesizing single NoC modules (i.e., even without considering long links), after the logic synthesis step, tools were expecting some target frequency to be reachable. However, after the placement phase, the results were up to 30% worse [6]. Unfortunately, traditional placement tools are not able to deeply modify the netlists they are given as an input. In general, they can only insert additional buffering to account for unexpected loads on few selected wires. Therefore, if the input netlist is fundamentally off the mark due to erroneous wire load expectations, not only a performance loss is certain, but the placement runtime skyrockets. To address this issue, placement-aware logic synthesis tools are replacing the old flow. In this case, after a very quick initial logic synthesis based on wire load models, the tool internally attempts a coarse placement of the current netlist, and also keeps optimizing the netlist based on the expected placement and the wire loads it implies. The final resulting netlist already considers placement-related effects. Therefore, after this netlist is fed to the actual placement tool, performance results will not incur major penalties. Overall, NoCs promise better scalability to future technology nodes, but their efficient design will require silicon-aware decision making at each level of the design process, in particular placement-aware logic synthesis.

Another challenge associated with the effects of nanoscale technologies is posed by global synchronization of large multi-core chips. The difficulty to control clock skew and the power associated with the clock distribution tree will probably cause a design paradigm shift toward globally asynchronous and locally synchronous (GALS) systems [10]. The basic GALS paradigm is based on a system composed of a number of synchronous blocks designed in a traditional way (and hence exploiting standard synthesis methodologies and tools). However, it is assumed that clocks of such synchronous systems are not necessarily correlated and consequently that those synchronous systems communicate asynchronously using handshake channels. Locally synchronous modules are usually surrounded by asynchronous wrappers providing such inter-block data transfer. Practical GALS implementations may form much more complex structures, such as bus [14] or NoC structures [7] for inter-block communications, and use different data synchronization mechanisms.

The GALS paradigm does not rely on absolute timing information and therefore favours composability: local islands of synchronicity can be arbitrarily combined to build up larger systems [28]. NoC architectures are generally viewed as an ideal target for application of the GALS paradigm [7]. While in the short term a network-centric system will be most likely composed of multiple clock domains with tight or loose correlation, in the long run fully asynchronous interconnects might be an effective solution. The uptake of these latter for commercial applications will largely depend on the availability of a suitable design tool flow. More in general, GALS NoCs are a promising means of tackling a number of interconnect issues, from power and EMI reduction to clock skew management, while preserving design modularity and improving the back-end time spent achieving timing convergence.

Finally, signal integrity issues (cross-talk, power supply noise, soft errors, etc.) will lead to more transient and permanent failures of signals, logic values, devices and interconnects, thus raising the reliability concern for on-chip communication [8]. 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 [18] and the exploitation of performance optimisation techniques such as wavefront pipelining [21]. It is worth observing that permanent communication failures can also occur as an effect of the deviation of technology parameters from nominal design values (such as effective gate length, threshold voltage or metal width). In fact, precise control of chip manufacturing becomes increasingly difficult and expensive to maintain in the nanometer regime. The ultimate effect is a circuit performance and power variability that results in increasing yield degradation in successive technology nodes. The support for process variation tolerance poses unique design challenges to on-chip networks. From a physical viewpoint, network circuits should be able to adapt to in-situ actual technology conditions, for instance by means of the self-calibrating techniques proposed in [17, 32]. From an architecture viewpoint, one or more sections of the network might be unusable, thus calling for routing and topology solutions able to preserve interconnection of operating nodes [20].

### 4.2.2 Scalability challenges

Present-day SoC design has broken with 1-processor system design that has dominated since 1971. Dual-processor system design is very common in the design of voice-only mobile telephone handsets. A general purpose processor handles the handset's operating system and user interface. A DSP handles the phone handset's baseband processing tasks (e.g., DSP functions such as FFTs and inverse FFTs, symbol coding and decoding, filtering). Processing bandwidth is finely tuned to be just enough for voice processing to cut product cost and power dissipation, which improves key selling points such as battery life and talk time.

Incorporation of multimedia features (music, still image, video) has placed additional processing demands on handset SoC designs and the finely tuned, cost-minimized 2-processor system designs for voice-only phones simply lack of processing bandwidth for these additional functions. Consequently, the most recent handset designs with new multimedia features are adding either hardware acceleration blocks or application processors to handle the processing requirements of the additional features. The design of multi-processor SoC systems is today a common practice in the embedded computing domain [LEI06]. Many SoCs today incorporate dozens or even hundreds of interconnected processor cores. The ITRS (International Technology Roadmap for Semiconductors) predicts that this trend will continue, and more than 800 integrated processor cores are expected in 2020 [3].

A similar design paradigm shift is taking place in the high performance computing domain [12]. In the past, performance scaling in conventional single-core processors has been accomplished largely through increases in clock frequency (accounting for roughly 80% of the performance gains at each technology generation). But frequency scaling has run into fundamental physical barriers. First, as chip geometries shrink and clock frequencies rise, the transistor leakage current increases, leading to excessive power consumption and heat. Second, the advantages of higher clock speeds are in part negated by memory latency, since memory access times have not been able to keep pace with increasing clock frequencies. Third, for certain applications, traditional serial architectures are becoming less efficient as processors get faster (due to the so-called von Neumann bottleneck), further undercutting any gains that frequency increases might otherwise achieve. In addition, RC delays in signal transmission are growing as feature sizes shrink, imposing an additional bottleneck that frequency increases don't address.

Therefore, performance will have to come by other means than boosting the clock speed of large monolithic cores. Another means of maintaining microprocessor performance growth has traditionally come from microarchitectural innovations. They include multiple instruction issue, dynamic scheduling, speculative execution and non-blocking caches. However, early predictions for the superscalar execution model projected diminishing returns in performance for increasing issue width, and they finally came true.

In light of the above issues, the most promising approach to deliver massively scalable computation horsepower while effectively managing power and heat is to break up functions into many concurrent operations and to distribute them across many parallel processing units. Rather than carrying out a few operations serially at an extremely high frequency, chip multiprocessors (CMPs) achieve high performance at more practical clock rates, by executing many operations in parallel. This approach is also more power-aware: rather than making use of a big, power-hungry and heat-producing core, CMPs need to activate only those cores needed for a given function, while idle cores are powered down. This fine-grained control over processing resources enables the chip to use only as much power as it is needed at any time.

Evidence of this CMP trend is today unmistakable as practically every commercial manufacturer of high-performance processors is currently introducing products based on multi-core architectures: AMD's Opteron, Intel's Montecito, Sun's Niagara, IBM's Cell and Power5. These systems aim to optimize performance per watt by operating multiple parallel processors at lower clock frequencies. Clearly, within the next few years, performance gains will come from increases in the number of processor cores per chip, leading to the emergence of a key bottleneck: the global intra-chip communication infrastructure. Perhaps the most daunting challenge to future systems is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processing cores in a power efficient fashion.

Thirty years of experience with microprocessors taught the industry that processors must communicate over buses, so the most efficient way of interconnecting processors has been through the use of common bus structures. When SoCs only used one or two processors, this approach was practical. With dozens or hundreds of processors on a silicon die, it no longer is. Bus-centric design restricts bandwidth and wastes the enormous connectivity potential of nanometer SoC designs. Moreover, chip-wide combinational structures (as many buses are) have been proven to map inefficiently to nanoscale technologies. Low latency, high data-rate, on-chip interconnection networks have therefore become a key to relieving one of the main bottlenecks for MPSoC and CMP system performance. NoCs represent an interconnect fabric which is highly scalable and can provide enough bandwidth to replace many traditional bus-based and/or point-to-point links.

### 4.2.3 Design productivity challenges

It is well known that synthesis and compiler technology development does not keep up with IC manufacturing technology development [3]. Moreover, times-to-market need to be kept as low as possible. Reuse of complex pre-verified design blocks is an effective means of increasing productivity, and regards both computation resources and the communication infrastructure [4]. 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 fabric compared with shared bus architectures. However, the reuse of processing elements is facilitated by the definition of standard interface sockets (corresponding to the front-end of network interfaces), which make the modularity property of NoC architectures effective. The Open Core Protocol (OCP) [1] was devised as an effective means of simplifying the integration task through the standardization of the core interface protocol. It also paves the way for more cost-effective system implementations, since it can be custom-tailored based on the complexity and the connectivity requirements of the attached cores. AMBA AXI [2] is another example of standard interface socket for processing and/or memory cores. The common feature of these point-to-point communication protocols is that they are core-centric transaction-based protocols which abstract away the implementation details of the system interconnect. Transactions are specified as though communication initiator and target were directly connected, taking the well-known layered approach to peer-to-peer communication that proved extremely successful in the domain of local- and wide-area networks. Referring to the reference ISO/OSI model, the network interface front-end can be viewed as the standardized core interface at session layer or transport layer, where network-agnostic high level communication services are provided. This way, the development of new processor cores and of new NoC architectures become two largely orthogonal activities.

Finally, let us observe that NoC components (e.g., switches or interfaces) can be instantiated multiple times in the same design (as opposed to the arbiter of traditional shared buses, which is instance-specific) and reused in a large number of products targeting a specific application domain.

## 4.3 Network-On-Chip Architecture

Most of the terminology for on-chip packet switched communication is adapted from the computer cluster domain. Messages that have to be transmitted across the network are usually partitioned into fixed-length *packets*. Packets in turn are often broken into message flow control units called *flits*. In 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:

1. Network interface
2. Switch
3. Network link

and will be described hereafter.

## 4.4 Network Interface

The network interface (NI) is the basic block allowing connectivity of communicating cores to the global on-chip network. It is is charge of providing several services, spanning from session and transport layers (e.g., decoupling computation from communication, definition of QoS requirements of network transactions, packetization) up to the layers closer to the physical implementation (e.g., flow control, clock domain crossing).

The NI provides basic core wrapping services. Their role is to adapt the communication protocol of the attached core to the communication protocol of the network. Layering naturally decouples system processing elements from the system they reside in, and therefore enables design teams to partition a design into numerous activities that can proceed concurrently since they are minimally inter-dependent. Layering also naturally enables core reuse in different systems. By selecting an industry standard interface, there is no added time for this reuse approach since all cores require such an interface.

Packetization is the very basic service the network interface should offer: taking the incoming signals of the processor core transaction and building packets compliant with the NoC communication protocol. This service should carefully optimize packet and flit size in order to achieve high-performance network operation and reduce implementation complexity of network building blocks.

At a lower level of abstraction, clock domain crossing may need to be performed by the NI. Even if the clock frequency is the same over the entire chip, phase adaptation will be needed for communications. In a more general case, the system might consist of multiple clock domains which act as islands of synchronicity. Clearly, proper synchronizers are needed at the clock domain boundary. A simplified case of this scenario occurs when each communicating core/tile

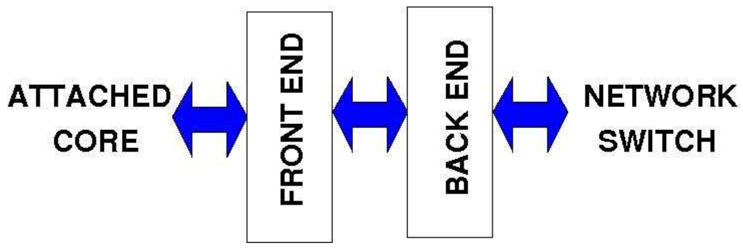


Figure 4.2: Architecture of a network interface.

coincides with one clock domain, while the network gives rise to an additional clock domain with an operating frequency which is typically much higher that that of the attached cores. This way, clock domain crossing has to be performed at the network interfaces. To the limit, a true GALS paradigm might be applied, which envisions a fully asynchronous interconnect fabric. In this case, the conversion from synchronous to asynchronous (and viceversa) takes place in the NI.

The NI is also the right place to specify and enforce latency and/or bandwidth guarantees. This can be achieved for instance by reserving virtual circuits throughout the network or by allocating multiple buffering resources and by designing complex packet schedulers in the NI in order handle traffic with different QoS requirements.

In addition to adaptation services, the NI is expected to provide the traditional transport layer services. Transaction ordering is one of them: whenever dynamic routing schemes are adopted in the network, packets can potentially arrive unordered, thus raising the memory consistency issue. In this case, NIs should reorder the transaction before forwarding data or control information to the attached component. Another transport layer service concerns the reliable delivery of packets from source to destination nodes. The on-chip communication medium has been historically considered as a reliable medium. However, recent studies expect this to be no longer the case in the context of nanoscale technologies. The NI could be involved in providing reliable network transactions by inserting parity check bits in packet tails at the packetization stage or by implementing end-to-end error control. Finally, NIs should be in charge of flow control. When a given buffering resource in the network is full, there needs to be a mechanism to stall the packet propagation and to propagate the stalling condition upstream. The flow control mechanism is in charge of regulating the flow of packets through the network, and of dealing with localized congestion. The NI is involved with the generation of flow control signals exchanged with the attached switch. Moreover, flow control has to be carried out also with the connected cores. In fact, when the network cannot accept new packets any more because of congestion, new transactions can be accepted in the NI from the connected core just at the same, provided the necessary amount of decoupling buffers is available in the NI. However, when also these buffers are full, the core behaviour is impacted by congestion in the network and it has to be stalled if further communication services are required.

A generic template for the NI architecture is illustrated in Fig. 4.2. The structural view of this hardware module includes a front-end and a back-end sub-module. The front-end implements a standardized point-to-point protocol allowing core reuse across several platforms. The interface then assumes the attributes of a socket, that is, an industry-wide well-understood attachment interface which should capture all signalling between the core and the system (such as dataflow signalling, errors, interrupts, flags, software flow control, testing). A distinctive requirement for this standard socket is to enable the configuration of specific interface instantiations along a number of dimensions (bus width, data handshaking, etc.). It is common practice to implement the front-end interface protocol so to keep backward compatibility with existing protocols such as AMBA AXI, OCP or DTL. This objective is achieved by using a transaction-based communication model, which assumes communicating cores of two different types: masters and slaves. Masters initiate transactions by issuing requests, which can be further split in commands (e.g., read or write) and write data. One or more slaves receive and execute each transaction. Optionally, a transaction can also involve a response issued by the slave to the master to return data or an acknowledgement of transaction completion. This request-response transaction-based model directly matches the bus-oriented communication abstractions typically implemented at core interfaces, thus reducing the complexity of core wrapping logic. As a consequence, the NI front-end can be viewed as an hardware implementation of the session layer in the ISO/OSI reference model. Traditionally, the session layer represents the user's interface to the network. High level communication services made available by the session layer must be implemented by the transport layer, which is still unaware of the implementation details of the system interconnect (and hence still belongs to the NI front-end in Fig. 4.2). The transport layer relieves the upper layers from any concern with providing reliable, sequenced and QoS oriented data transfers. It is the first true and basic end-to-end layer. For instance, the transport layer might be in charge of establishing connection-oriented end-to-end communications.

The transport layer provides transparent transfer of data between end nodes using the services of the network layer. These services, together with those offered by the link layer and the physical layer, are implemented in the NI back-end. Data packetization and routing functions can be viewed as essential tasks performed by the network layer, and are tightly interrelated. The NI back-end also provides data link layer services. Primarily, communication reliability has to be ensured by means of proper error control strategies and effective error recovery techniques. Moreover, flow control is handled at this layer, by means of upstream (downstream) signalling regulating data arrival from the processor core (data propagation to the first switch in the route), but also of piggybacking mechanisms and buffer/credit flushing techniques. Finally, the physical channel interface to the network has to be properly designed. NoCs have distinctive challenges in this domain, consisting of clock domain crossing, high frequency link operation, low-swing signalling and noise-tolerant communication services.

An insight in the $\chi$pipes Lite [33] network interface implementation will provide an example of these concepts. Its architecture is showed in Fig. 4.3. This chapter views this case study as an example of a lightweight NI architecture for best effort NoCs.

The network interface (NI) is designed as a bridge between an OCP interface and the NoC switching fabric. Its purposes are protocol conversion (from OCP to network protocol), packetization, the computation of routing information (stored in a Look-Up Table, LUT), flit buffering to improve performance and

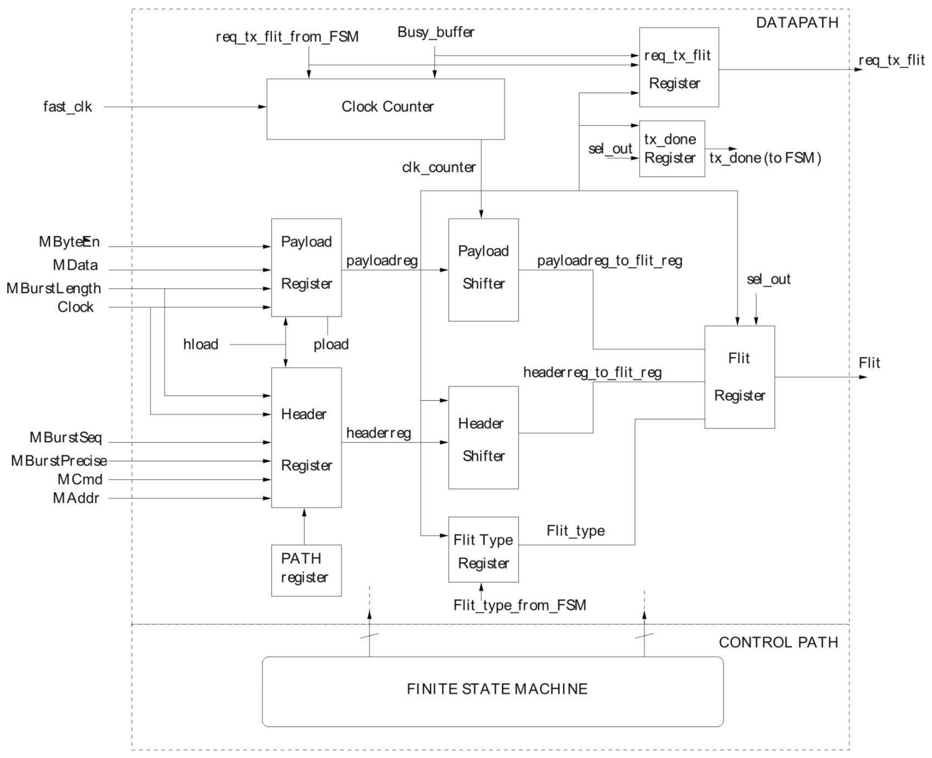


Figure 4.3: *χpipes Lite* network interface initiator architecture.

flow control. For any given OCP transaction, some fields have to be transmitted once, while other fields need to be transmitted repeatedly. Initiator and target NIs are attached to communication initiators and targets respectively. Each NI is split in two sub-modules: one for the request and one for the response channel. These sub-modules are loosely coupled: whenever a transaction requiring a response is processed by the request channel, the response channel is notified; whenever the response is received, the request channel is unblocked. The NI is built around two registers: one holds the transaction header (1 refresh per OCP transaction), while the second one holds the payload (refreshed at each OCP burst beat). A set of flits encodes the header register, followed by multiple sets of flits encoding a snapshot of the payload register subsequent to a new burst beat. Header and payload content is never allowed to mix. Routing information is attached to the header flit of a packet by checking the transaction address against an LUT. The length in bit of the routing path depends on the maximum switch radix and on the maximum number of hops in the specific network instance at hand.

The header and payload registers represent the boundary of the NI front-end and also act as clock domain decoupling buffers. These registers can be read from the NI back-end at a much higher speed than the writing speed of the OCP side. In practice, network and attached cores can operate at different frequencies. The only constraint posed by this architecture is that the OCP frequency is obtained by applying an integer divider to the network frequency. A finer grain control of the frequencies would make the NI architecture more

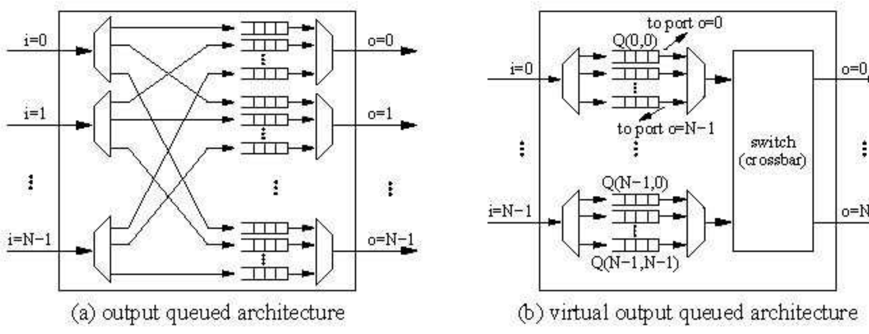


Figure 4.4: Common buffering strategies for switch architectures.

complex and costly from an area and power viewpoint.

## 4.5 Switch Architecture

Switches carry packets injected into the network to their destination, following a statically (deterministic) or dynamically (adaptive) defined routing path. A switch forwards packets from one of its input ports to one or more of its output ports using arbitration logic to solve conflicts. Switch design is usually driven by a power-performance trade-off: high-performance switches for on-chip communication require power-hungry *buffering resources*. The buffering strategy (Fig.4.4) determines the location of buffers inside the switch. A switch may allocate buffers at the input ports, at the output ports or both. In input queuing the queues are at the input of the switch. A scheduler determines at which times queues are connected to which output ports such that no contention occurs. The scheduler derives contention-free connections, a switch matrix (crossbar switch) can be used to implement the connections. In traditional input queuing, there is a single queue per input, resulting in a buffer cost of N queues per switch. However, due to the so-called head-of-line blocking, for large N, switch utilization saturates at 59% [27]. Therefore, input queuing results in weak utilization of the links. Another variant of input queuing is virtual output queuing (VOQ), which combines the advantages of input and output queuing. It has a switch like in input queuing and has the link utilization close to that of output queuing: 100% link utilization can still be achieved, when N is large [26]. As for output queuing, there are $N^2$ queues. For every input I there are N queues Q(I,o), one for each output o. Typically the set of N queues at each input port of a VOQ switch is mapped to a single RAM.

Switch performance can be affected by the kind of *routing method* implemented. In *source routing*, the path of the packet across the network is fixed before injecting it into the network and is embedded in the packet header. Switches only have to read the routing directives from the packet header and to apply them. This is the approach taken by the network interface of Fig. 4.3. In *distributed routing*, the path is dynamically computed at every switch when the header flit is routed.

This latter carries information on the destination node, which is used by the local switch to perform routing path computation. This can be achieved in two ways: by either accessing a local look-up table or by means of a combinational logic implementing predefined routing algorithms. Source routing results in faster switches, but features limited scalability properties. Moreover, adaptive routing is not feasible with this approach. With distributed routing, switches are more complex and therefore slower (due to the routing logic on the critical path), but they are able to take dynamic routing decisions.

*Switching techniques* determine the way network resources (buffer capacity, link bandwidth) are allocated to packets traversing the network. *Bufferless* switching would be the most power-effective approach for resource allocation, however it comes with heavy side-effects. In fact, whenever a packet cannot proceed on its way to destination due to a conflict with another packet, it should be either discarded (which is not acceptable for on-chip networks) or misrouted. This latter solution is likely to incur severe livelock problems. A minimal amount of buffering is instead required by circuit switching. In this case, long lasting connections are established throughout the network between source and destination nodes. Once reserved, these connections can be used by the corresponding communication flows in a contention-free regime. Buffers in this architecture serve two main purposes: retiming (e.g., input and/or output sampling of the signals in the switch) or buffering of packets devoted to circuit set-up. In *circuit switching*, the latency for circuit set-up adversely impacts overall system performance. Moreover, long lasting circuits established throughout the network suffer from low link utilization. For the above reasons, *buffered* switching is the most common switching technique used in the NoC domain up to date. In practice, data buffering is implemented at each switch in order to decouple the allocation of adjacent channels in time. The buffer allocation granularity occurs on a packet- or flit-basis, depending on the specific swiching technique. Three policies are feasible in this context [35].

In *store-and-forward switching*, an entire packet is received and stored in a switch before forwarding it to the next switch. This is the most demanding approach in terms of memory requirements and switch latency. Also *virtual cut-through switching* requires buffer space for an entire packet, but allows for a lower latency communication, since a packet starts to be forwarded as soon as the next switch in the path guarantees that the complete packet can be stored. If this is not the case, the current router must be able to store the entire packet. Finally, a *wormhole switching* scheme can be employed to reduce switch memory requirements and to provide low latency communication. The head flit enables 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 forwarded to the next switch as soon as enough space is available to store it, even though there is no enough space to store the entire packet. If a head 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 deadlock prone and may result in low link utilization due to link blocking.

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.

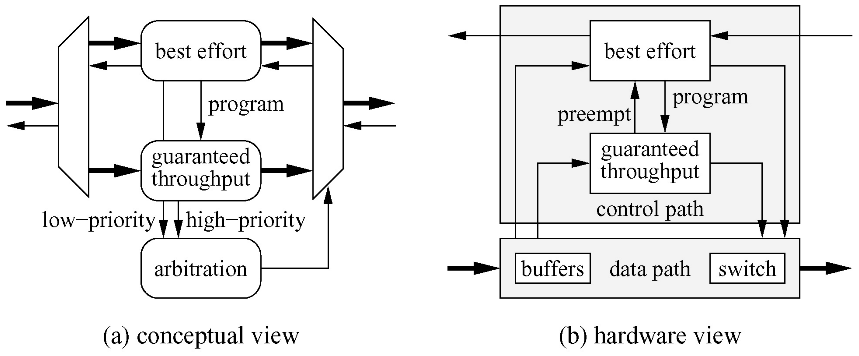


Figure 4.5: CA combined GT-BE router.

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. As already mentioned, with *circuit switching* a connection is set up over which all subsequent data is transported. Therefore, contention resolution takes place only at set-up at the granularity of connections, and time-related guarantees during data transport can be provided. In *time division circuit switching* [9] , 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 mainly rely on network over-sizing to upper-bound fluctuations of performance metrics. Two case studies taken from the literature will help us understand how the above design issues can be addressed in real-life switch implementations.

### 4.5.1 Æthereal

The *Æthereal* NoC architecture makes use of a router that tries to combine guaranteed throughput (GT) and best effort (BE) services [9]. The GT router subsystem is based on a time-division multiplexed circuit switching approach. A router uses a *slot table* to (i) avoid contention on a link, (ii) divide up bandwidth per link between connections, and (iii) switch data to the correct output. Every slot table T has S time slots (rows), and N router outputs (columns). There is a logical notion of synchronicity: all routers in the network are in the same fixed-duration slot. In a slot S at most one *block* of data can be read/written per input/output port. In the next slot, the read blocks are written to their appropriate output ports. Blocks thus propagate in a store-and-forward 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 showed that both input queuing with wormhole switching or virtual cut-through switching and virtual output queuing with wormhole switching are feasible in terms of buffering cost. The BE and GT router sub-systems are combined in the *Æthereal* router architecture of

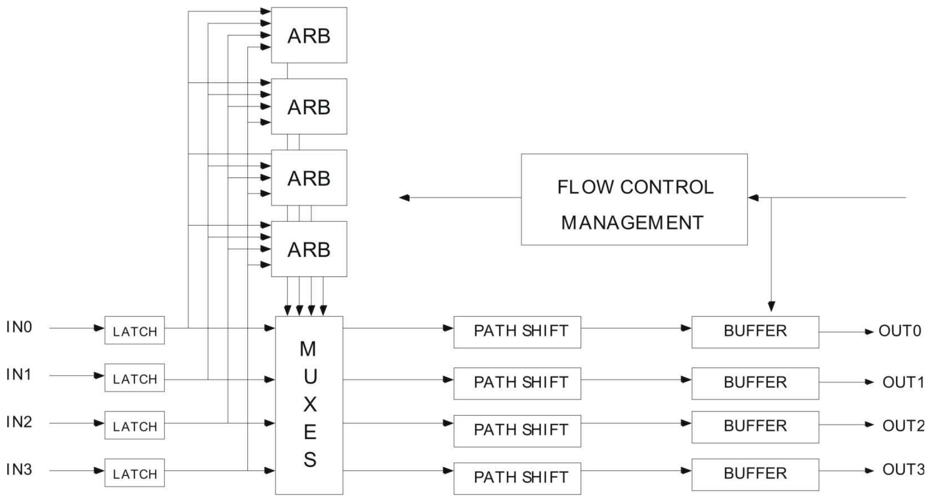


Figure 4.6: *χpipes Lite* switch architecture.

Fig.4.5. The GT router offers a fixed end-to-end latency for its traffic, which is given the highest priority by the arbiter. The BE router uses all the bandwidth (slots) that has not been reserved or used by GT traffic. GT router slot tables are programmed by means of BE packets (see the arrow "program" in Fig. 4.5). Negotiations, resulting in slot allocation, can be done at compilation time, and be configured deterministically at run time. Alternatively, negotiations can be done at run time.

### 4.5.2 Xpipes Lite

A different perspective has been taken in the design of the switch for the best effort *χpipes Lite* NoC [33]. The switch is represented in Fig.4.6. It features one cycle latency for switch operation and one cycle for traversing the ouput link, thus 2 cycles are required to traverse the switch fabric overall. The switch is output-queued and wormhole switching is used as a convenient buffer allocation policy. Please notice that while in traditional output queuing each input channel has its own buffer in each output port, in this case the need to reduce total switch power (no full custom design techniques were supposed to be used) has led to the implementation of a unique buffer for all input channels at each output port. The choice of different performance-power trade-off points may significantly differentiate on-chip networks from traditional off-chip realizations due to the different constraints they have to meet.

The input ports are latched to break the timing path. As static source routing is used, the header of the packet contains all the information required to route the packet. This keeps the switch routing logic minimal.

Arbitration is handled by an allocator module for each output port according to a round-robin priority algorithm and performed upon receipt of a header flit. After a packet wins the arbitration, routing information pertaining the current switch is rotated away in the header flit. This allows to keep the next hop at a fixed offset within the header flit, thus simplifying switch implementation. Access to output ports is granted until a tail flit arrives. The *χpipes Lite* switch

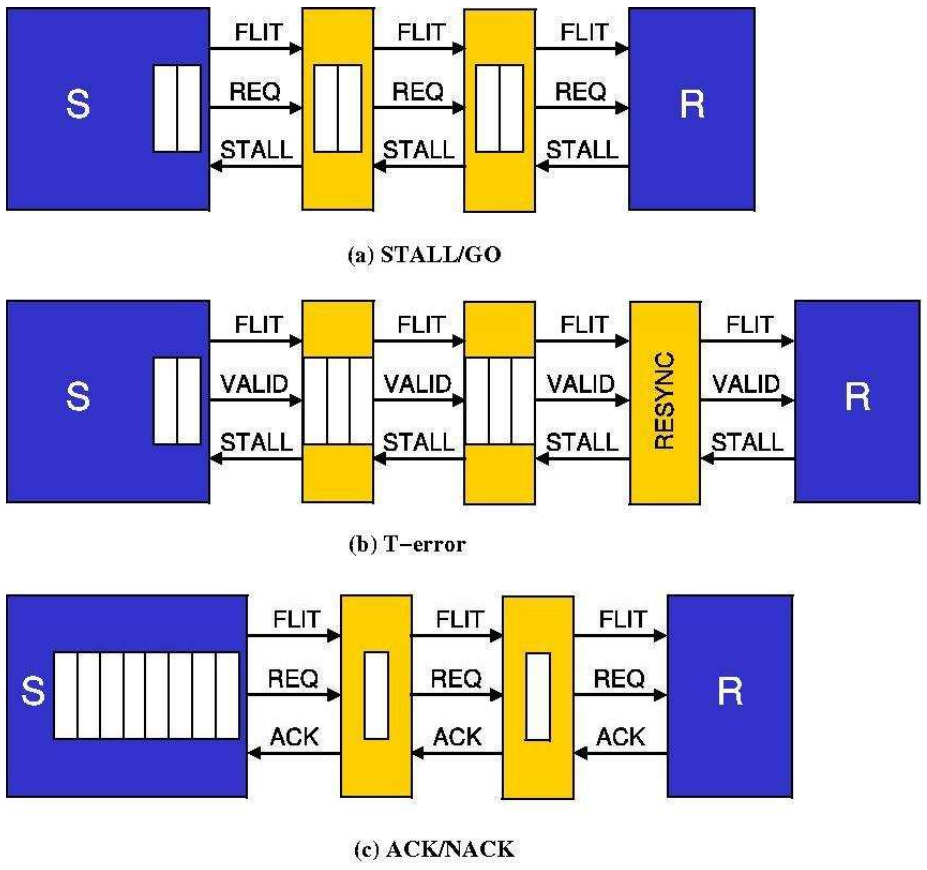


Figure 4.7: Impact of flow control on link pipelining implementation.

is defined as a soft macro, i.e., it is parameterizable in the number of input and output ports, in the link width, in the output buffer size but also in the flow control technique.

## 4.6 Link Design

In the context of nanoscale technologies, long wires increasingly determine the maximum clock rate, and hence performance, of the entire design. The problem becomes particularly serious for domain-specific heterogeneous SoCs, where the wire structure is highly irregular and may include both short and extremely long switch-to-switch links. Moreover, it has been estimated that only a fraction of the chip area (between 0.4 and 1.4%) will be reachable in one clock cycle [34].

A solution to overcome the interconnect-delay problem consists of pipelining interconnects [24, 25]. Wires can be partitioned into segments (by means of relay stations, which have a function similar to that of latches on a pipelined datapath) whose length satisfies pre-defined timing requirements (e.g., desired clock speed of the design). This way, link delay is changed into latency, but the data introduction rate is not bounded by the link delay any more. Now, the latency of a channel connecting two modules may turn out to be more than one clock cycle. Therefore, if the functionality of the design is based on the sequencing of the signals and not on their exact timing, then link pipelining does not change the functional correctness of the design. This requires the system to consist 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 nanoscale 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 [24]. The effectiveness of the latency-insensitive design methodology is strongly related to the ability of maintaining a sufficient communication throughput in presence of increased channel latencies.

The architecture of a relay station depends on the flow control scheme used across the network, since switch-to-switch links not only carry packets but also control information determining the way the downstream node communicates buffer availability to the upstream node (i.e., flow control signals). In order to make this point clear, let us focus on three flow control protocols with very different characteristics: *stall/go*, *T-error* and *Ack/nack* [5].

### 4.6.1 Stall/Go

*stall/go* is a low-overhead scheme which assumes reliable flit delivery. *T-error* is much more complex, and provides logic to detect timing errors in data transmission. This support is however only partial, and usually exploited to improve performance rather than to add reliability. Finally, *Ack/nack* is designed to support thorough fault detection and handling by means of retransmissions.

*Stall/go* is a very simple realization of an ON/OFF flow control protocol. It requires just two control wires (Fig.4.7a): one going forward and flagging data availability, and one going backward and signaling either a condition of buffers filled ("STALL") or of buffers free ("GO"). Stall/go can be implemented with distributed buffering along the link; namely, every repeater can be designed as a very simple two-stage FIFO. The sender only needs two buffers to cope with stalls in the very first link repeater, thus resulting in an overall buffer requirement of $2N + 2$ registers, with minimal control logic. Power is minimized since any congestion issue simply results in no unneeded transitions over the data wires. Performance is also good, since the maximum sustained throughput in the absence of congestion is of one flit per cycle by design, and recovery from congestion is instantaneous (stalled flits get queued along the link towards the receiver, ready for flow resumption). In the NoC domain with pipelined links, *stall/go* indirectly reflects the performance of credit-based policies, since they exhibit equivalent behaviour. The main drawback of *stall/go* is that no provision whatsoever is available for fault handling. Should any flit get corrupted, some complex higher-level protocol must be triggered.

### 4.6.2 T-Error

The *T-error* protocol (Fig.4.7b) aggressively deals with communication over physical links, either stretching the distance among repeaters or increasing the operating frequency with respect to a conventional design. As a result, timing errors become likely on the link. Faults are handled by a repeater architecture leveraging upon a second delayed clock to resample input data, to detect any inconsistency and to emit a VALID control signal. If the surrounding logic is to be kept unchanged, as we assume in this chapter, a resynchronization stage must be added between the end of the link and the receiving switch. This logic handles the offset among the original and the delayed clocks, thus realigning the timing of DATA and VALID wires; this incurs a one-cycle latency penalty. The timing budget provided by the T-Error architecture can also be exploited to achieve greater system reliability, by configuring the links with spacing and frequency as conservative as in traditional protocols. However, *T-error*lacks a really thorough fault handling: for example, errors with large time constants would not be detected. Mission-critical systems, or systems in noisy environments, may need to rely on higher-level fault correction protocols. The area requirements of *T-error* include three buffers in each repeater and two at the sender, plus the receiver device and quite a bit of overhead in control logic. A conservative estimate of the resulting area is $3M + 2$, with M being up to 50% lower than N if *T-error* features are used to stretch the link spacing. Unnecessary flit retransmissions upon congestion are avoided, but a power overhead is still present due to the control logic. Performance is of course dependent on the amount of self-induced errors.

### 4.6.3 Ack/Nack

The main idea behind the *Ack/Nack* flow control protocol (Fig. 4.7c) is that transmission errors may happen during a transaction. For this reason, while flits are sent on a link, a copy is kept locally in a buffer at the sender. When flits are received, either an ACKnowledge (ACK) or NotACKnowledge (NACK) is sent back. Upon receipt of an ACK, the sender deletes the local copy of the flit; upon receipt of a NACK, the sender rewinds its output queue and starts resending flits starting from the corrupted one, with a GO-BACK-N policy. This means that any other flit possibly in flight in the time window among the sending of the corrupted flit and its resending will be discarded and resent. Other retransmission policies are feasible, but they exhibit higher logic complexity. Fault tolerance is built in by design, provided encoders and decoders for error control codes are implemented at the source and destination respectively. In an *Ack/Nack* flow control, a sustained throughput of one flit per cycle can be achieved, provided enough buffering is available. Repeaters on the link can be simple registers, while, with N repeaters, $2N + k$ buffers are required at the source to guarantee maximum throughput, since *Ack/Nack* feedback at the sender is only sampled after a round-trip delay since the original flit injection. The value of k depends on the latency of the logic at the sending and receiving ends. Overall, the minimum buffer requirement to avoid incurring bandwidth penalties in a NACK-free environment is therefore $3N + k$. *Ack/Nack* provides ideal throughput and latency until no NACKs are issued. If NACKs were only due to sporadic errors, the impact on performance would be negligible. However, if NACKs have to be issued also upon congestion events, the round-trip delay in the notification causes a performance hit which is very pronounced especially with long pipelined links. Moreover, flit bouncing between sender and receiver causes a waste of power.

| | Stall/Go | T-Error | Ack/Nack |
|---|---|---|---|
| BUFFER AREA | $2N+2$ | $> 3M+2$ | $3N+K$ |
| LOGIC AREA | Low | High | Medium |
| PERFORMANCE | Good | Good | Depends |
| $POWER_{ESTIMATION}$ | Low | Medium/High | High |
| FAULT TOLERANCE | Unavailable | Partial | Supported |

Table 4.1: Flow control protocols at a glance.

The distinctive features of each flow control scheme are summarized in Table 4.1, which also points out the implications of the flow control technique on the implementation of link pipelining. Clearly, the best choice depends on the ultimate design objective and on the level of abstraction at which designers intend to enforce fault tolerance.

## 4.7 NoC design challenges

Although the benefits of NoCs architectures are substantial, reaching their full potential presents numerous research challenges. There are three main issues that must be addressed for future NoCs: power consumption, latency and CAD compatibility. As exposed in [19], research challenges for NoCs architectures can be classified into four broad areas:

**Technology and circuits.** The most important technology constraint for NoCs is power. To close the power gap, research should develop optimized circuits for NoC components. Other constraints include design productivity and cost, reflecting the issues derived from the use of new or exotic technologies that require an additional effort in developing CAD compatibility.

**NoC microarchitecture and system architecture.** Architecture research must address the primary issues of power and latency, as well as critical issues such as congestion control. This should be address at network-level (topology, routing, and flow control) as well as in the router microarchitecture. The delay of routers and the number of hops required by a typical message should be reduced. Circuit research to reduce channel latency can also help to close the latency gap. All of these architectural improvements must be developed taking into account technology constraints and the limited power budget. In particular, power consumption is a critical issue and future NoC architectures must address it effectively, for example by avoiding unnecessary work or by dynamically modulating voltage and frequency.

**NoC design tools.** NoC design tools should be able to better interface with system-level constraints and design, for instance by means of an accurate characterization and modelling of system traffic. It is very important to update CAD tools with specialized libraries for new NoC microarchitectures and interconnects, validation capabilities for new NoC designs and feedback from end-user to simplify the design process. In order to preserve design tools usability with CMOS scaling, it is of utmost importance to develop new area, power, timing, thermal and reliability models. Finally, new methods to estimate power-performance of NoCs other than simulations are needed, due to the increasing complexity of future NoC designs.

**NoC comparison.** Also, it is very important to develop common evaluation metrics (such as latency and bandwidth under area, power, energy, and heat dissipation constraints) and standard benchmarks to allow direct unambiguous comparison between different NoC architectures.

## 4.8 Conclusion

Parallel architectures are becoming mainstream both in the high performance and in the embedded computing domains. The most daunting challenge to future multi-processor systems is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processor cores in a power efficient fashion. Even though in the short term the evolution of topologies and protocols of state-of-the-art interconnects will suffice to sustain MPSoC scalability, on-chip networks are widely recognized as the long term and disruptive solution to the problem of on-chip communication. This chapter has provided basic principles and network building blocks have been analysed. Finally, the chapter has highlighted the challenges that lie ahead to make NoC architectures mainstream. They can be broadly categorized into low power and low latency circuits and architectures, silicon-aware design tools with system-level exploration capabilities and the definition of standard quality metrics and benchmarks.

# Bibliography

[1] Ocp international partnership. open core protocol specification, 2001.
http://www.ocpip.org.

[2] AMBA AXI Protocol Specification, 2003.

[3] ITRS, 2007.
Http://www.itrs.net/Links/2007ITRS/Home2007.htm.

[4] H.Tenhunen A.Jantsch.
Will networks on chip close the productivity gap?
Networks on Chip, A.Jantsch and Hannu Tenhunen (eds), Kluwer, 2003.
pp.3-18.

[5] D.Bertozzi L.Benini A.Pullini, F.Angiolini.
Fault tolerance overhead in network-on-chip flow control schemes.
SBCCI, 2005.
pp.224-229.

[6] P.Meloni D.Atienza S.Murali L.Raffo G. De Micheli L.Benini A.Pullini, F.Angiolini.
Noc design and implementation in 65nm technology.
1st IEEE/ACM Int. Symp. On Networks-on-Chip, 2007.
pp.273-282.

[7] et al. D. Lattard.
A telecom baseband circuit-based on an asynchronous network-on-chip.
Proc. of the International Solid State Circuits Conference, ISSCC, Feb 2007.
San Francisco, USA.

[8] G.De Micheli D.Bertozzi, L.Benini.
Energy-reliability trade-off for nocs.
Networks on Chip, A.Jantsch and Hannu Tenhunen (eds.), Kluwer, 2003.
pp.107-129.

[9] A.Radulescu J.van Meerbergen P.Wielage E.Waterlander E.Rijpkema, K.Goossens.
Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip.
Design Automation and Test in Europe DATE03, March 2003.
pp.350-355.

[10] F.K.Gurkaynak et al.
Gals at eth zurich: Success or failure?

Proc. of the 12th IEEE Int. Symp. On Asynchronous Circuits and Systems, 2006.
pag.150-159.

[11] 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, Feb 2002.
Vol.1, pp. 473-474.

[12] S.Borkar et al.
Platform 2015: Intel processor and platform evolution for the next decade.
Technology@Intel Magazine, Intel Corporation, March 2005.

[13] S.J.Lee et al.
An 800 mhz star-connected on-chip network for application to systems on a chip.
ISSCC03, February 2003.

[14] T. Villiger et al.
elf-timed ring for globally-asynchronous locally-synchronous systems.
Proc. of the Ninth IEEE Int. Symp. on Asynchronous Circuits and Systems, Vancouver, 2003.
pp. 141-150.

[15] S.Carta L.Benini L.Raffo F.Angiolini, P.Meloni.
Contrasting a noc and a traditional interconnect fabric with layout awareness.
DATE, 2006.
pp.1-6.

[16] F.Boekhorst.
Ambient intelligence, the next paradigm for consumer electronics: How will it affect silicon?
ISSCC, February 2002.
Vol.1 ,pp.28-31.

[17] P.Thiran G. De Micheli F.Worm, P.Ienne.
A robust self-calibrating transmission scheme for on-chip networks.
in IEEE Transactions on Very Large Scale Integration (VLSI) System, Jan. 2005.
Vol.13, no.1.

[18] J.M.Rabaey H.Zhang, V.George.
Low-swing on-chip signaling techniques: Effectiveness and robustness.
IEEE Trans. on VLSI Systems, June 2000.
Vol.8 nr.3, pp.264-272.

[19] R.Ho D.N. Jayasimha S.W. Keckler L.Peh J.D.Owens, W.J. Dally.
Research challenges for on-chip interconnection networks.
IEEE Micro, 2007.
pp.96-108.

[20] J.Duato J.Flich, S.Rodrigo.
An efficient implementation of distributed routing algorithms for nocs.
2nd IEEE Int. Symp. On Networks-on-Chip, 2008.

[21] W.Wolf J.Xu.
Wave pipelining for application-specific networks-on-chips.
CASES02, October 2002.
pp.198-201.

[22] L.Benini.
Application-specific noc design.
Design Automation and Test in Europe Conf., 2006.
pp.1-5.

[23] G.De Micheli L.Benini.
Networks on chips: a new soc paradigm.
IEEE Computer, January 2002.
Vol.35, Issue 1, pp.70-78.

[24] A.L.Sangiovanni-Vincentelli L.P.Carloni, K.L.McMillan.
Theory of latency-insensitive design.
IEEE Trans. On CAD of ICs and Systems, September 2001.
Vol.20, nr.9, pp.1059-1076.

[25] L.Scheffer.
Methodologies and tools for pipelined on-chip interconnects.
Int. Conf. On Computer Design, 2002.
pp.152-157.

[26] N. McKeown.
Scheduling algorithms for input-queued cell switches, 1995.
PhD thesis, Univ. of California Berkeley.

[27] M.J.Karol.
Input versus output queuing on a space division packet switch.
IEEE Trans. on Communications, 1987.
COM-35(12), pp.1347-1356.

[28] C.Stahl-M.Piz M.Krstić, E.Grass.
System integration by request-driven gals design.
IEE Proc. Computers & Digital Techniques, September 2006.
Vol. 153, Issue 5, pp 362-372.

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

[30] K.Goossens P.Wielage.
Networks on silicon: Blessing or nightmare?
Proc. Of the Euromicro Symposium on Digital System Design DSD02, September 2002.
pp.196-200.

[31] J.P.Soininen-M.Forsell M.Millberg J.Oeberg K.Tiensyrja A.Hemani S.Kumar, A.Jantsch.
A network on chip architecture and design methodology.
IEEE Symp. on VLSI ISVLSI02, April 2002.
pp.105-112.

[32] M.Lajolo S.Medardoni, D.Bertozzi.

Variation tolerant noc design by means of self-calibrating links.
Proc. of DATE 2008., March 2008.

[33] et al. S.Stergiou.
Xpipes lite: A synthesis oriented design flow for networks on chips.
Proc. of the Design, Automation and Test in Europe Conference, 2005.
pp. 1188-1193.

[34] S. W. Keckler V. Agarwal, M. S. Hrishikesh and D. Burger.
Clock rate versus ipc: The end of the road for conventional microarchitectures.
in Proc. 27th Annu. Int. Symp. Computer Architecture, June 2000.
pp. 248-250.

[35] W.J.Dally and B.Towels.
Principles and practices of interconnection networks.
Morgan Kaufmann, 2004.

[36] B.Towles W.J.Dally.
Route packets, not wires: On-chip interconnection networks.
Design and Automation Conference DAC01, June 2001.
pp.684-689.

Chapter 5

# Switches: Architecture, Analysys and Optimization

## 5.1 Introduction

Networks-on-chip (NoCs) bring a packet-based communication paradigm to the on-chip domain, and consist of a modular and scalable architecture extensively relying on path segmentation. Their ability to tackle the scalability challenge for on-chip communication and to relieve the interconnect delay bottleneck at the architecture-level is generally accepted [12, 14].

At the same time, a few emerging issues still prevent NoC technology from becoming mainstream. First, NoCs break down the long timing paths of state-of-the-art interconnects in a number of hops that packets traverse on their way to destination. This causes communication latency to become a serious concern for system performance. Thus, design techniques for low-latency NoCs are being devoted a lot of attention by the NoC community [1, 2, 3].

Second, power dissipation issues have grown to such importance that they now constrain attainable performance. A multi-layer AMBA crossbar has been showed in [12] to be still more area- and power-efficient than its NoC counterpart (one order of magnitude), with most of the power drained by buffering resources and by the clock tree[15]. Interestingly, in many test cases switches have been showed to consume as much power as the sum of master and slave network interfaces, due to the lower switching activity of these latter and to the lower operating frequency of most NI front-ends[15].

Third, even if NoC capacitive loads and propagation delays can be controlled much better than in shared busses, issues such as wiring congestion, link power consumption and the need for placement-aware logic synthesis still have to be explored to assess the feasibility of NoCs in forthcoming technology nodes.

The intricacies of nanoscale design are urging a paradigm shift in NoC characterization frameworks. High-level performance and power models have been long advocated for assessing NoC architectures and for key predictions on the scalability properties of NoCs to future technologies[16, 18, 20]. The questionable accuracy of these models has led some researchers to take an implementation-oriented approach, presenting area, performance and power figures as a result of their effort investment on the backend synthesis flow[13, 21, 17]. These works generally trade-off the high level of accuracy of their results with the restricted scope of their analysis. The comparative study in [11] is an exception, although it compares the energy efficiency of very specific switch architectures, which limits the generality of results. Moreover, these latter are provided for a 90nm technology node.

This chapter takes the same implementation-oriented approach and sheds light on the main architectural trade-offs that are at the basis of NoC switch design in 65nm low-power technology. Our work leverages placement-aware logic synthesis tools and goes through a two step analysis to provide backend synthesis-proven insights in switch design. First, we accurately assess the delay, area and power breakdown of traditional switches for packet-switched networks. Then, we identify an architectural level tuning knob that allows us to specialize a generic switch architecture for specific design objectives such as low-latency or low-power.

We identify control and datapath decoupling as such tuning knob, and account for the buffering, switching and flow-control implications of changing the propagation delay of control signals through a mux-based crossbar into an initial latency for path establishment inside the switch. Since configuration signals can be kept stable throughout the entire connection lifetime, a significant slack in data-path operation is now exposed, which can be exploited in two ways: applying gate-level netlist transformations to cut down on power consumption or increasing data-path operating frequency with respect to the coupled architecture. The connection life-time depends on the path allocation granularity, and can be as short as the time for single packet traversal, or as long as the time to propagate multi-packet messages. In order to overcome the low resource utilization and routing congestion of long-lasting connections used in traditional circuit-switching solutions, our analysis targets packet-level circuit switching, wherein circuits are established on a per-(equivalent) packet basis.

This chapter is structured as follows. Section 5.2 illustrates our modelling and simulation environment and the deployed synthesis flow. Section 5.3 presents the basic switching fabric to which control and datapath decoupling is applied (Section 5.4). Section 5.5 describes and analyzes switch implementations spanning the area-latency-power trade-off. Conclusions are drawn in Section 5.6.

## 5.2 Modelling and synthesis flow

Our modelling, characterization and back-end synthesis flow of NoC switches is illustrated in Fig.5.1. The entire system is modelled in SystemC at the cycle accurate level and functional traffic is injected by means of programmable generators of transactions compliant with the OCP2.0 protocol specification.

The slave side of the OCP connection is represented by a network interface front-end, which converts the OCP protocol into the network protocol and performs clock domain crossing. Packets are injected into the switches by means of a NI back-end, which was made available in two variants: one supports packet switching, while the other one supports packet-level circuit-switching.

Switches were modelled at the RTL-equivalent level of accuracy. See section 5.5.1 for the detailed description of the switch architectural variants. In order to better focus our analysis, only switches were synthesized. We used a 65nm

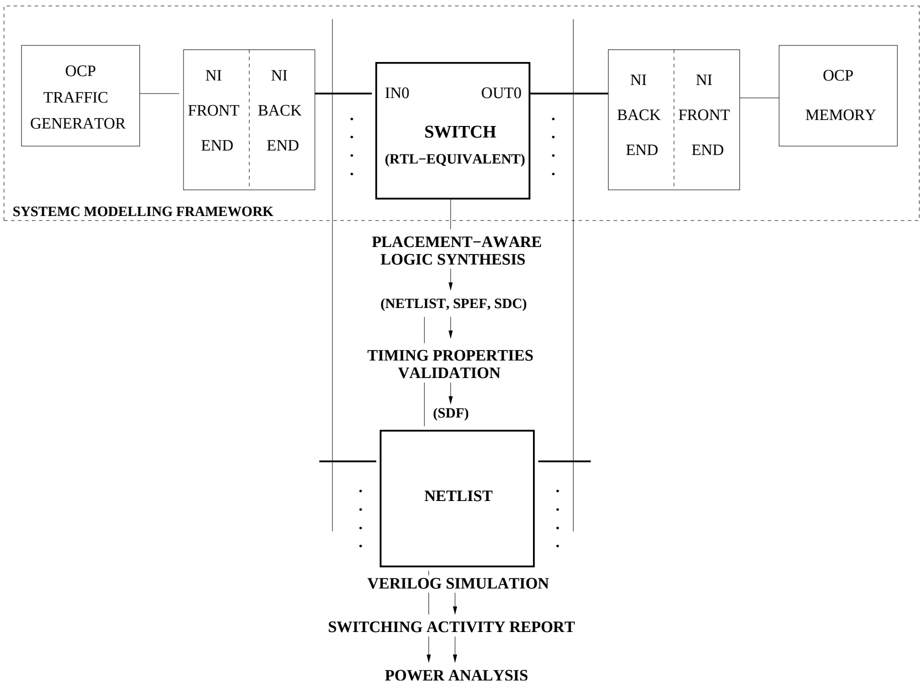


Figure 5.1: Modelling and synthesis framework of NoC switches.

standard cell technology library provided by a partner foundry, featuring low threshold voltages and optimized for low-power. Although the traditional flow for standard cell design features logic synthesis and placement as two clearly decoupled stages, previous work such as [13] indicates that this assumption is fundamentally inadequate for the 65nm node, in particular due to wireload model inconsistencies. We therefore reverted to placement-aware logic synthesis tools (Synopsys Physical Compiler), which minimize the risk of significant performance losses in post place-and-route netlists and which normally guarantee reasonable placement runtimes. The final resulting netlist already considers placement-related effects. Such netlist was then simulated in Modelsim. Input patterns were provided by the same SystemC test infrastructure of the RTL models by deploying the Modelsim interface for mixed Verilog-SystemC simulation. Therefore, the same traffic patterns were used both for performance and for power characterizations. HDL simulation includes accurate delay models generated by Synopsys PrimeTime, which also validates timing properties of the synthesized netlists. It also accounts for estimated parasitics (SPEF file) and design constraints (SDC file). Finally, a switching activity report was generated, and fed to Synopsys PrimePower, which combines it with SPEF and SDC information to get power/energy profiles.

## 5.3 Basic switching fabric

We started our analysis by modelling a basic switching fabric for packet-switched networks. It implements a 2-cycle-latency (one for switch operation and one for traversing the ouput link), output-queued wormhole-switched router supporting fixed priority arbitration on the output lines. Allocation of inputs towards specific output ports is handled by an allocator module for each output port. Access to output ports is granted until a tail flit arrives. Arbitration is subsequently performed upon receipt of a header flit which also carries desired output port information (source-based static routing). Link (and flit) width is 32 bits.

Finally, a flow control scheme with stall/go semantics was implemented[22]. It is a simple realization of an on/off flow control protocol, requiring just two control wires: one going forward and flagging data availability, and one going backward and signaling either a condition of buffer filled (stall) or of buffer free (go). For this protocol, recovery from congestion is instantaneous.

Logic synthesis of the switch proved that its critical path results from the interaction of the control path with the datapath. Most of the clock cycle time is spent in driving the crossbar selection signals. The signal delay through the allocator and the crossbar in fact takes 55% of the critical path in a 4x4 switch. Another 15% is spent in buffering and flow control logic, while the remaining delay is due to the clock propagation time of the input sampling stage (21%) and to the library setup time for the output flip-flop (about 9%). The maximum operating frequency for the 4x4 switch was found to be around 1GHz, to cut down on it. The basic idea consists of decoupling the control-path and the data-path of a switch, therefore analyzing their impact on switch performance separately. We will therefore prove that the switch critical path is a result of the control-path and of its interaction with the data-path. At this stage, multiple conflicting requests from the switch input ports must be arbitrated, and most of the clock cycle time is spent in driving the crossbar selection signals.

## 5.4 Control and datapath decoupling

Since the data-path configuration in a switch stays the same during the entire packet traversal time but the arbitration and crossbar multiplexer selection delays impact the switch critical path at each clock cycle, we decided to decouple the datapath and the control path. For each input packet, the switch arbiter is invoked once, thus resulting in an initial arbitration latency for path setup through the data-path. The concept architecture is illustrated in Fig.5.2. Please note that the control signals can be driven by the upstream arbiter or directly by the initiator network interface depending on the network protocol. Since the focus of this work is not on the efficiency of the path setup protocol and of its implementation (which we view as a further optimization step), but on the core switch architecture, we assumed a setup mechanism based on control signals. An initiator device drives a request line of the arbiter, at the same time indicating the desired output port and the data input port. Such information are easily available in statically-routed networks. In modelling the decoupled switch architecture, we reused as many components as possible from the basic switching fabric and just changed small details (e.g., tail recognition logic, arbiter module interface). Blocks implementing core functions stay the same in both implementations (e.g., flow control, arbitration algorithm, crossbar).

By first synthesizing the data-path of the decoupled switch architecture, we found the critical path breakdown illustrated in Fig.5.3 as a function of switch arity. The decoupling from the control path led to largely reduced critical paths

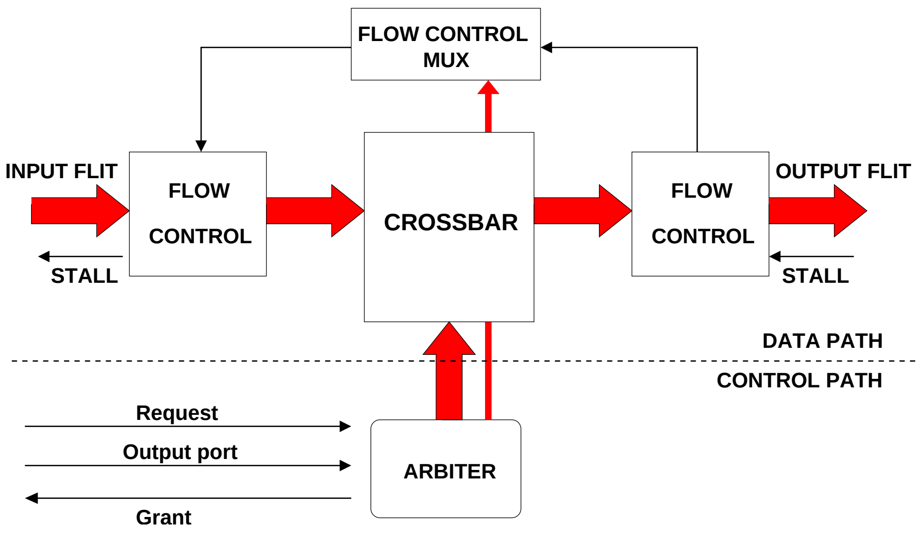


Figure 5.2: Switch architecture with decoupled datapath and control path.

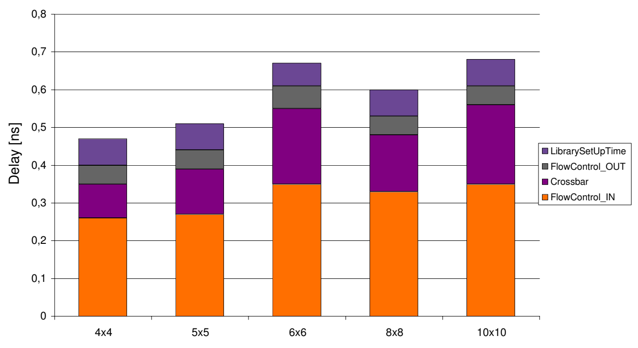


Figure 5.3: Critical path in the data plane

(52% improvement for a 4x4 switch) in the data plane. However, we observed that a 5x5 switch, generally deployed to implement mesh topologies, hardly achieves 2 GHz operating frequency while a 10x10 switch hardly works at 1.5 GHz. Although such speeds are well beyond those achievable by NoC prototypes in state-of-the-art technologies, we would have expected a higher acceleration of the data-path. The reason for this lies in the significant fraction of the data-path delay drained by flow control logic (66%). In a 4x4 switch, crossbar delay accounts only for 19%, while the remaining 15% can be ascribed to the library setup time. The relative contribution of the crossbar delay grows to 31% for a 10x10 switch, wherein flow control still accounts for 59%.

With stall/go, two-slot buffer stages are needed at switch inputs and outputs for the purpose of flow control: one slot is used for normal flit propagation and the other one is a back-up slot needed to store incoming flits before the backpressure mechanism stops the upstream switch propagation of trailing flits. More buffer slots can be added for performance gains, but are not required for functionality. Although simplified, this buffering architecture requires some control logic whose delay adds up to the data-path delay and determines the performance limitations we have observed. Decoupling such delay from the data-path is not as easy as for the arbitration logic, since flow control comes into play on a flit-by-flit basis and not on a per-packet basis.

By synthesizing the new switch architecture, we found that the arbitration delay almost reaches 1.2 ns for 10x10 switches (round-robin arbitration), and is below 1 ns for switches smaller than 5x5. This means that if the data-path can be clocked at 1,5 GHz, 2 equivalent data-path clock cycles are needed for path establishment across the switch. We found this number to be pretty independent of the switch arity. If the data-path can be operated at more than 2 GHz, the arbitration latency then grows to 3 cycles. We tested this up to 2,4 GHz. These values have been derived for a round-robin arbitration stage, and more relaxed timing constraints have been derived when fixed priority policies are implemented.

Finally, our synthesis results indicated that an arbitration round can always be carried out within 2 data-path equivalent clock cycles, and this number proved almost independent of the switch arity. This means that the initial latency for path setup through the switch can be reduced to just 2 data-path clock cycles.

## 5.5 Spanning the area-latency-power trade-off

With respect to the operating frequency of the basic switching fabric, the data-path of the decoupled architecture now exposes a significant slack. This latter can be exploited in a number of implementation variants spanning the latency-power trade-off. At one side of the design space, we can come up with a latency-optimized solution, which translates the slack into a higher operating frequency, much higher than the speed of the networked communication actors, so that the penalty cycles for across-network communication can be masked by shorter network clock periods. System interconnect overclocking with respect to computation and storage elements speed is a typical design option of NoC-based systems, in that NoC path segmentation allows to achieve very high operating frequencies. On the contrary, state-of-the-art interconnects either suffer from the delay scalability limitations of crossbars or introduce protocols allowing the insertion of retiming stages in a transparent way. This system interconnect overcloking solution however comes at a significant power cost.

At the opposite extreme of the design space stands a solution wherein the slack is exploited to infer low-power switch implementations. In this latter case, an incremental run of the synthesis tool with relaxed timing constraints is able to perform power-driven gate-level netlist transformations resulting in more power-efficient designs for the same target performance, as already proved for simpler designs in [19].

### 5.5.1 Selection of switch architectures

In order to prove and explore in detail the design flexibility made available by decoupled switch architectures, we selected a few representative implementation variants in the power-latency design space. Without lack of generality, a 4x4 switch will be hereafter assumed.

First, we consider the initial switching fabric described in Section 5.3, i.e., a *coupled architecture* suitable for packet-switched networks deploying wormhole switching. This solution heavily relies on buffering to sustain performance. Our in-house experience suggested to implement 6 slot buffers for each output port of the output-buffered switch. Based on our initial critical path analysis, we chose a 1GHz operating frequency for this switch, which will be hereafter referred to as *packet-switching (PS)*. OCP cores (processing cores and memories) were assumed to be working at 500MHz, with clock domain crossing implemented inside network interfaces (NIs).

Second, we consider the low-latency corner of the design space, i.e. a *decoupled architecture* where the datapath is operated at a speed of 2 GHz. The control path (i.e., the arbitration logic) takes 2 datapath equivalent clock cycles to execute, and therefore is operated at 1GHz. For a fair analysis, we opted for the architectural solution illustrated in Fig.5.2, implementing stall/go flow control stages at switch inputs and outputs, since these stages are there also in the PS variant. In this case, however, we just implemented the minimum buffering requirements for correct functionality (i.e., 2 slots), since circuit-switching is deployed. In particular, we opted for packet-level circuit switching (PLCS), i.e., we setup and tear-down a connection inside the switch for the transmission of a PS-equivalent packet. Circuit management is handled as illustrated in Fig.5.2.

Third, we consider the low-power design corner, i.e. a *decoupled architecture* where the datapath is operated at the relaxed frequency of 1 GHz, and so is the control path. The incremental synthesis run triggering power-driven netlist transformations was performed. Also this solution has the same flow control architecture and connection management protocol of Fig.5.2. An overview of the selected implementation variants is reported in Table 5.1.

| Switch variant | OCP freq. | Datapath freq. | Control Path freq. |
|---|---|---|---|
| PS | 500MHz | 1GHz | 1GHz |
| LowLatPLCS | 500MHz | 2GHz | 1GHz |
| LowPowPLCS | 500MHz | 1GHz | 1GHz |

Table 5.1: Switch implementation variants.

### 5.5.2 Area comparative analysis

As can be observed from Fig.5.4, PS consumes more area than other solutions for at least two reasons: it deploys 6-slot buffers for each output port and it consequently features buffer control logic impacting area. The contribution of the sequential logic of course decreases for LowPowPLCS and LowLatPLCS, which just feature 2-slot buffers at the input and output ports. LowLatPLCS takes in general more area than LowPowPLCS, since the datapath has been synthesized with tight timing requirements, which have been met at the cost of area.

### 5.5.3 Performance comparative analysis

We simulated the system in Fig.5.1 with the different switch variants. Interestingly, we took NI-related effects into account. At first, we monitored *transaction*

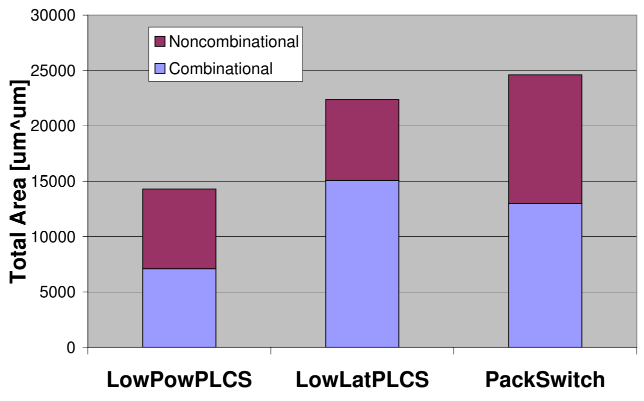


Figure 5.4: Area breakdown.

*latency*. We measured it for burst write transactions, from the time the first write data word is sampled at the master NI front-end, to the time the same word is driven to the OCP memory from the slave NI front-end. This experiment is independent of packet length. For PS, such latency amounts to 10 *network clock cycles*: 2 in the master NI, 1 in the switch, 2 in switch input and output links, and 5 in the slave NI (depacketing and synchronization with the OCP clock). For LowPowPLCS, transaction latency grows to 14 *network clock cycles*, due to the circuit setup overhead through the switch. Even LowLatPLCS exhibits 14 *network cycles*: it features one more cycle in the arbiter than LowPowPLCS (since the data-path now operates at a higher speed) but 1 cycle less in the slave NI for the easier alignment of its fast clock with the OCP one.

The above *network cycles* have different clock periods, and a useful insight consists of expressing them in terms of the OCP clock period. In doing this conversion, we also sum up the handshaking time spent in the TGEN-master NI and slave NI-Memory OCP links. Final results indicate an overall transaction latency of *6 OCP clock cycles for LowLatPLCS, 8 OCP cycles for PS and 10 OCP cycles for LowPowPLCS*. This meets our expectations: the low latency solution masks its setup time overhead with the higher frequency of its datapath (which explains the difference with LowPowPLCS latency), and brings additional performance improvements for the same reason. Scalability analysis of these values with circuit length and setup time is left for future work.

A summary of latency figures is reported in Table 5.2.

| Implementation variant | Network Cycles | Network cycle period | OCP cycles |
|---|---|---|---|
| PS | 10 | 1ns | 8 |
| LowLatPLCS | 14 | 0.5ns | 6 |
| LowPowPLCS | 14 | 1ns | 10 |

Table 5.2: Latency figures.

As an additional performance metric, we monitored the *execution time* spent by OCP traffic generators (TGENs) to complete a set of predefined OCP transactions. This metric accounts for the fact that the master OCP interface only

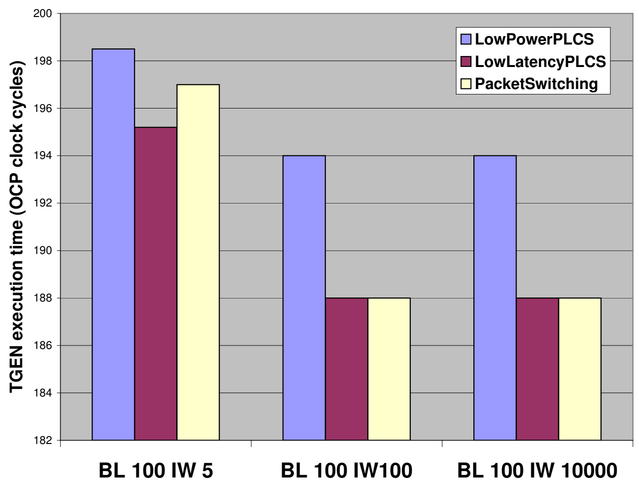


Figure 5.5: Execution time results.

needs to wait untill one OCP transaction has been accepted by the network. This is relevant in particular for posted writes, which do not force the OCP master interface to wait till the write is actually carried out by the memory and its success is notified back. The more buffering is implemented through the network, the lower the blocking time of the master OCP interface, and the lower the execution time of the TGEN. Fig.5.5 shows the execution time for one TGEN to carry out 10 burst write transactions (each consisting of 100 burst beats) with increasing inter-burst idle waiting time (IW). Many other traffic patterns were simulated, but did not provide different results with respect to those simulated for Fig.5.5.

When the IW is large, there are no tail effects between the end of one transaction and the beginning of the following one. Moreover, since the memory throughput is rather slow compared to the network one (it works at 500MHz and features 1 wait state), at regime the behavior is the same for all solutions: 1 data word is accepted by the master NI front-end every 2 OCP clock cycles, since the internal network pipeline from source to destination is full and the data transfer beat is determined by memory speed. The only difference lies in the ability of each solution to perform the initial transient for each burst efficiently. In particular, LowLatPLCS and PS show the same behavior, in that in spite of their different speeds and network protocols they can consume the same number of data words from TGEN at max speed (1 word per OCP clock cycle) before achieving the regime operating condition. On the contrary, LowPowPLCS saturates much faster and therefore its TGEN takes longer to execute.

When the IW reduces significantly, then tail effects come into play, which are better handled by LowLatPLCS. In particular, the time that the LowLat-PLCS solution spends in setting up the circuit leaves enough time for flits of the previous packet to move further down the network pipeline, and this allows to consume more write data words of the next burst transaction at max speed. Then, the usual saturation regime takes place. On the contrary, the unmasked

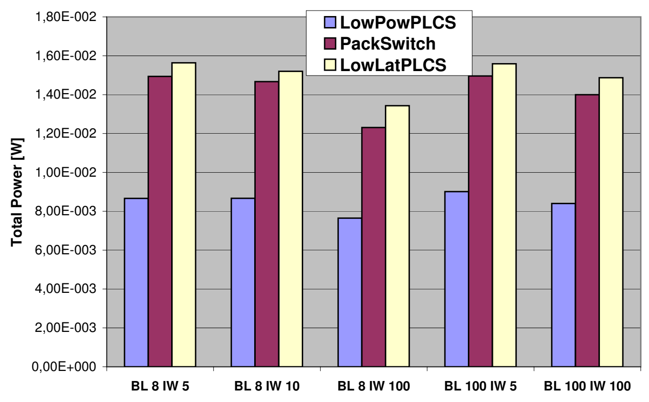


Figure 5.6: Power consumption analysis.

setup time in LowPowPLCS slows down its execution and leaves bubbles in the memory input buffer. This set of experiments shows that (i) high transmission rates are the best operating regime for LowLatPLCS and that (ii) further performance differentiation would need a removal of the slave bottleneck. Experiments characterizing switch behavior under congestion are omitted for lack of space, but did not introduce significant novelties to the picture. The same holds for the power characterization that follows.

### 5.5.4 Power/energy comparative analysis

Following the power characterization flow illustrated in Fig.5.1, we profiled both stand-by power (no network transactions) as well as operational power. As regards stand-by power, we found that LowPowPLCS consumes 37% less total power than PS (7.43mW against 11.8mW). This is due to the lower number of buffering resources while working at the same operating frequency, and (to a smaller extent) to the power optimizations performed by the synthesis tool on the data-path. In contrast, LowLatPLCS consumes only 10.6% more power than PS. Again, this is due to the lower number of buffering resources and to the fact that only the data-path is operated at a higher frequency. Finally, we measured that leakage power accounts in all switch variants at most for 0.5% of total power. This is due to the use of an effective low-power technology library.

Fig.5.6 reports the operational power consumed by the switch for different traffic patterns injected by the TGEN. We observe that LowPowPLCS is always the most power-efficient implementation, followed by PS and by LowLatPLCS. We observe that as the IW increases, power decreases and tends asymptotically to the stand-by power. Interestingly, we observe that the power gap between LowPowPLCS and PS (on average 40.11% power savings) is much larger than the one between PS and LowLatPLCS. In spite of the worst performance incurred by LowPowPLCS, its total energy (Fig.5.7) is still always below that of other solutions, saving 38% energy on average with respect to packet switching. With short IW, the performance benefits of LowLatPLCS counterbalance its power overhead and determine 6.23% energy savings. When the IW increases, we observe performance balancing between LowLatPLCS and PS, therefore the higher power incurred by LowLatPLCS determines also an energy overhead.

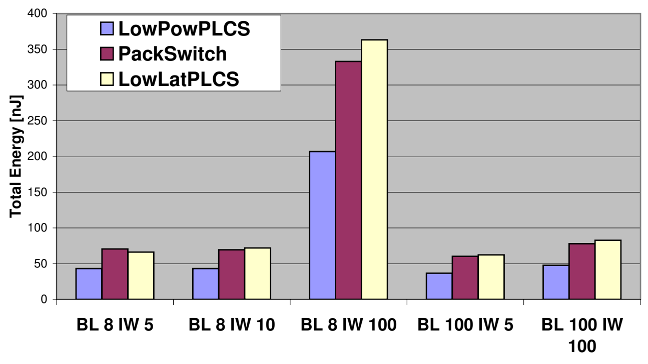


Figure 5.7: Energy dissipation analysis.

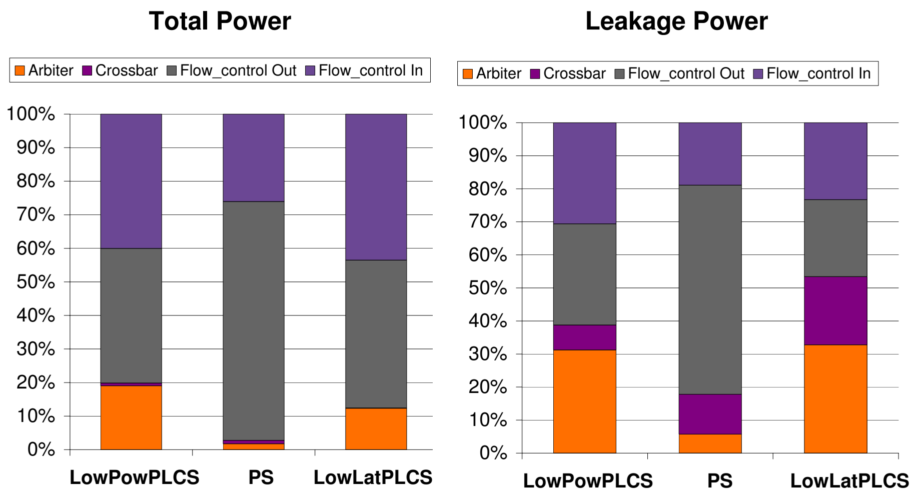


Figure 5.8: Power breakdown.

These results highlight the potentials of the LowPowPLCS solution when considering a trade-off between performance, area and power quality metrics, while raise some skepticism on LowLatPLCS, at least for these memory-dominated test conditions. Experiments on congestion showed power and energy figures in agreement with those just illustrated, and were not reported.

We illustrate in Fig.5.8 the total and leakage power breakdown for all the switch variants. Since dynamic power dominates total power, its breakdown is similar to that of total power and hence is not reported. In PS we observe that buffering resources at the input (for flow control) and mostly at the output stage (for flow control and performance) dominate overall power consumption. In LowPowPLCS flow control stages are again the most power-hungry components (80%),while the remaining 20% is consumed by the arbiter module. Almost the same behaviour was measured for LowLatPLCS, with the only difference that the arbiter's relative contribution to power decreases since the datapath is now clocked two times faster. Please note that the crossbar contribution to total power is always almost negligible.

As regards leakage power, we found a precise correlation between the area breakdown and the leakage power breakdown, which denotes the need for low-area footprints in leakage-sensitive technologies. It is worth observing that leakage of one sub-module may heavily depend on the specific synthesis constraints for that component, as the leakage contributions of the crossbar in LowPowPLCS and LowLatPLCS clearly indicate. At the same time, while it is true that the arbitration control stage incurs a small fraction of the switch dynamic power, care must be devoted to arbitrarily increasing the complexity of the control logic due to its impact on leakage power.

## 5.6 Conclusions

This chapter explores the performance,area and power implications of decoupling control path from datapath in NoC switches. In particular, it was showed how effectively decoupled switch architectures can be used to span the power-latency trade-off. We intended to point out quality metrics of different switch variants strictly associated with the internal switch architecture. The dependence of such metrics on other external factors (e.g., circuit setup time and protocol, network size, end-to-end flow control efficiency) was intentionally left out of this work.

# Bibliography

[1] J.Kim et al.,
"A low latency router supporting adaptivity for on-chip interconnects",
DAC, pp.559-564, 2005.

[2] S.Stergiou et al.,
"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips",
DAC, pp.559-564, 2005.

[3] J.Kim et al.,
"A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks",
Int.Symp.on Computer Architecture, pp.4-15, 2006.

[4] Rijpkema, E. et al.,
Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip,
in Computers and Digital Techniques, IEE Proceedings,
Volume 150, Issue 5, 22 Sept. 2003 Page(s): 294-302.

[5] S.Sathe, D.Wiklund, D.Liu,
Design of a switching node (router) for on-chip networks.
Int. Conf. on ASIC, pp.75-78, 2003.

[6] D.Wiklund, D.Liu,
SoCBUS: Switched network on chip for hard real time embedded systems,
in Proc. of IPDPS, 2003, pp.78a.

[7] P.T.Wolkotte et al.
An energy-efficient reconfigurable circuit-switched network-on-chip,
in Proc. of IPDPS, 2005, pp.155a.

[8] A.Ahmadinia et al.,
A practical approach for circuit routing on dynamic reconfigurable devices,
in Proc. of RSP, 2005, pp.84-90.

[9] P.H.Pham, Y.Kumar, C.Kim,
High performance and area-efficient circuit-switched network on chip design,
in Proc. of CIT06, 2006.

[10] Q.Ye, J.Liu, L.R.Zheng,
Switch Design and Implementation for Network-on-Chip,
Conference on High Density Microsystem Design and Packaging and Component Failure Analysis,
June 2005, pp.1-7.

[11] A.Banerjee, R.Mullins, S.Moore,
A power and energy exploration of network-on-chip architectures,
in International Symposium on Networks-on-Chip, 2007.
May 2007, pp.163-172.

[12] F.Angiolini, P.Meloni, S.Carta, L.Benini, L.Raffo,
Contrasting a NoC and a traditional interconnect fabric with layout awareness,
in Design Automation and Test in Europe Conference,
2006, pp.124-129.

[13] A.Pullini et al.,
NoC design and implementation in 65nm technology,
Int. Symposium on Networks-on-Chip,
2007, pp.273-282.

[14] F.Steenhof et al.,
Networks on Chips for high-end consumer-electronics TV system architectures,
DATE, 2006, pp.148-153.

[15] D.Bertozzi,
Network interface architecture and design issues,
book chapter from *Networks on Chips: Technology and Tools*,
Morgan Kaufmann, 2006, San Francisco (USA).

[16] H.Wang, X.Zhu, L.S.Peh, S.Malik,
Orion: a power-performance simulator for interconnection networks,
in ACM/IEEE MICRO, Nov. 2002, pp.294-305.

[17] K.Lee, S.J.Lee, H.J.Yoo,
Low-power network-on-chip for high-performance SoC design,
IEEE Transactions on VLSI Systems, vol.14, no. 2, pp.148-160, 2006.

[18] X.Chen, L.S.Peh,
Leakage power modelling and optimization in interconnection networks,
ISLPED 2003, pp.90-95.

[19] S.Medardoni, D.Bertozzi, E.Macii,
Power-optimal RTL arithmetic unit soft-macro selection strategy for leakage-sensitive technologies,
to be presented at ISLPED 2007.

[20] J.Xi, P.Zhong,
A transaction-level NoC simulation platform with architecture-level dynamic and leakage energy models,
GLSVLSI 2006, pp. 341-344.

[21] N.Banerjee, P.Vellanki, K.S.Chatha,
A power and performance model for network-on-chip architectures,
DATE 2004, pag.21250.

[22] A.Pullini et al.,
”Fault tolerance overhead in network-on-chip flow control schemes”,
SBCCI, pp.224-229, 2005.

# Chapter 6

# Network Interface: Architecture, Analysys and Optimization

With the advent of multi-processor system-on-chip (MPSoC) technology, computation efficiency can be achieved by combining multiple programmable processors within a multicore system, hence only marginally impacting programmability and configurability. Relevant examples thereof come from commercial products both in the high-performance microprocessor domain [1] and in the embedded computing domain with tighter optimization constraints [2, 3].

Perhaps the most daunting challenge to make MPSoC technology mainstream is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processing cores. This task is on burden of the global intrachip communication infrastructure. Networks-on-Chip are a promising solution for designing scalable communication architectures for MPSoCs, featuring better modularity and design predictability when compared to bus based systems.

Although preliminary analysis frameworks have pointed out the performance enhancements achievable by on-chip networks [4], the area concern for NoC implementation was raised by many works in the open literature. For instance, it was showed in [4] that a NoC architecture for a 30 core system takes one order of magnitude more cell area than that of a state-of-the-art multi-layer interconnect. In terms of floorplan area, the increase can be as large as a few tens of squared millimiters.

It should be observed that in realistic NoC architectures, network interfaces (NI) play a significant role in determining overall NoC area. In [5], a TV companion chip was redesigned with a NoC as the interconnect fabric, and 78% of increase in chip area was proved to come from the NIs. In the $\chi$*pipes* based system in [4], more than half of the NoC area is due to NIs. Finally, these network components should come with low area footprint, since the size of IP modules attached to them is relatively small.

Network interface sharing could be a viable option to reduce the area overhead of these components. The most common approach consists of attaching multiple cores to the same NI, thus sharing the same input port to the network (see for instance [6]). This approach involves replication of the buffering resources in the NI, thus leading to an increase of area which hardly justifies this design choice. In fact, trusting published reports, buffers can account for more than 30% of the NI area. Moreover, the replication of buffering resources increases the complexity of buffer control logic (e.g., scheduling), thus resulting in a reduction of the maximum operating frequency or in an increase of NI latency. The problem can be relieved as in [6] by infering custom-made hardware FIFOs at the cost of flexibility. Other approaches delve into the intricacies of NI design trying to reduce its complexity [7]. However, this solution can be successful only up to a certain extent, since the support for processing cores with advanced communication capabilities (e.g., multiple oustanding transactions, out-of-order completion, quality of service guarantees) requires complex NI architectures anyway.

This chapter advocates for a more radical solution preserving the flexibility of fully synthesized architectures. We propose a network interface sharing architecture wherein the entire NI resources (including buffering) are shared among a cluster of processing cores or communication targets. We design traffic merging and splitting modules for this purpose, which interleave the different traffic flows on a unique network interface. While potentially resulting in more area efficient implementations, this solution needs to preserve compliance with end-to-end communication protocols linking network interfaces with communicating cores. The work focuses on the Open Core Protocol (OCP) [25], a standard socket interface for industry-relevant MPSoCs, and proves the feasibility of traffic merging and splitting in this context.

Moreover we shows that by devising proper optimization techniques at the merging and splitting modules, NI sharing outperforms a pure serialization of conflicting network access patterns. Moreover, system-level effects associated with the presence of global slave bottlenecks or physical-level effects associated with a higher achievable operating frequency (due to a lower switch radix) may mask the further level of arbitration introduced by traffic mergers and splitters. Finally, this work assesses the non-trivial area-performance trade-off spanned by our architecture based on NI sharing in an application-specific as well as in a general purpose scenario.

Design guidelines for a traffic merger and splitter are analysed in Section 6.1. Then, architecture details and related optimizations follow in Sections 6.2 and 6.3, respectively. Synthesis results are illustrated in Section 6.4. Basic performance tests follow in Section 6.5 while a system-level analysis is illustrated in Section 6.6.

## 6.1 Design requirements

Fig.6.1 illustrates the concept architecture of a *traffic merger*, which merges traffic flows coming from different initiator cores. A similar scheme can be applied to group a number of target cores, thus allowing them to share a unique network interface. The architecture block required for this purpose is denoted as *traffic splitter* from here on. The main requirements when designing a traffic merger (splitter) are as follows:

- **Preserving compliance to the point-to-point protocol**. Communicating cores are typically wrapped in such a way that they are connected to the

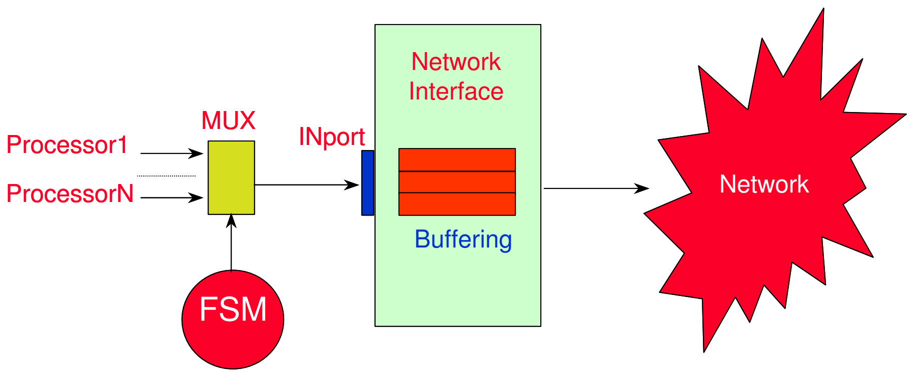


Figure 6.1: Concept network interface architecture

network via standard point-to-point communication protocols (e.g., OCP, AXI, DTL). Mergers and splitters have to be inserted on the physical point-to-point link in a transparent way with respect to the protocol, since traffic merging and splitting are handled at a lower level of abstraction than that specified by the standard interface sockets. As an example, when an OCP master interface intends to start a new command but looses arbitration to access the shared network interface, it must not be aware of this and the blocking condition should be signaled as the unavailability of the OCP slave interface to process a new command. In practice, this can be achieved by the merger driving the SCmdAccept signal low for that master until the network interface becomes available. Without lack of generality, the OCP protocol (revision 2.1) will be hereafter used as the reference point-to-point communication protocol.

**- Providing different levels of support for multiple outstanding transactions**. In order to relieve the performance overhead induced by NI sharing, multiple outstanding transactions could be supported. Even when the processor cores do not support multiple outstanding transactions, the cluster of cores sharing the same NI appears as a single core which is capable of them. Obviously, mergers and splitters should support multiple outstanding transactions only if the network interface supports them. Unfortunately, network interfaces usually reflect processor core complexity, and if cores are blocking on read transactions, the corresponding NI will not provide the support for more advanced communication features not to overdesign the system. In this work, we focus on a network interface architecture which supports multiple outstanding write transactions but a single oustanding read transaction. As a consequence, these are also the communication capabilities supported by our mergers and splitters. As will be proved later on, a few simple optimizations are enough to break the serial dependency between conflicting transactions from different processor cores in a cluster.

**- Minimization of area and delay overhead.** Area consumption depends on the level of support for multiple outstanding transactions and on the number of states in the FSM. The requirement is that the area of a traffic merger or splitter be less than that of a network interface, thus making NI sharing cost effective. As Fig.6.2 indicates, mergers (and splitters) might turn out to be particularly area demanding since they cannot handle the OCP signaling from

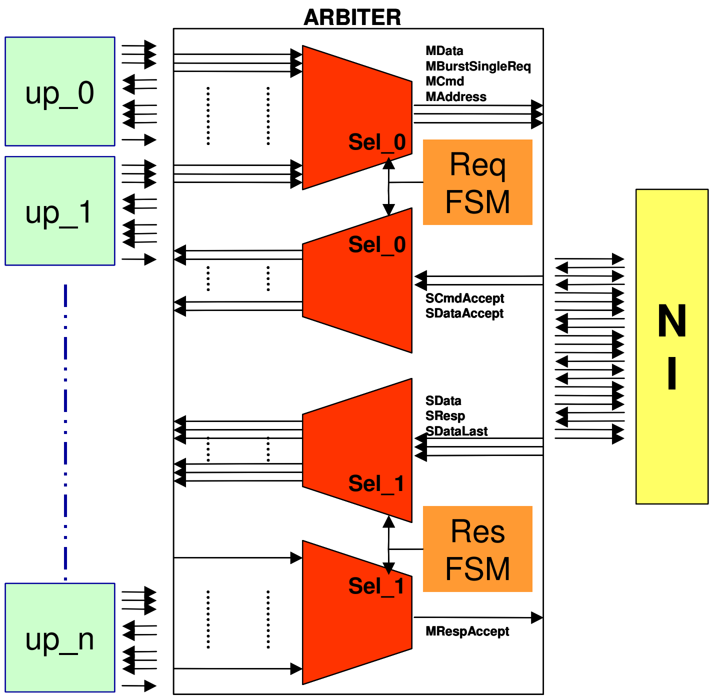


Figure 6.2: Detailed merger architecture

a processor core as a whole, but they have to handle both the request and the response channel, and inside each channel they have to deal with handshaking signals traveling in opposite directions. Moreover, since mergers break the timing path of point-to-point links, their critical path needs to be carefully designed not to degrade the operating frequency of the processor cores and of the network components.

- *Scalability.* Merger/splitter architecture needs to support a scalable number of attachable cores that will be decided based on system performance analysis (e.g., traffic patterns, real-time requirements).

## 6.2 NI sharing architectures

Before dwelving into the architecture details of the proposed merger and splitter, an overview of the network interface architecture that will be used throughout this chapter is provided. It is the interface module to the $\chi$pipes NoC fabric [27].

### 6.2.1 Network interface architecture

The network interface is designed as a bridge between an OCP interface and a NoC switching fabric. Its purposes are the synchronization between OCP and network timing, (de-)packetization, the computation of routing information (stored in a Look-Up Table, LUT) and flit buffering to improve performance. Differentiated bridges exist between communication initiators and the

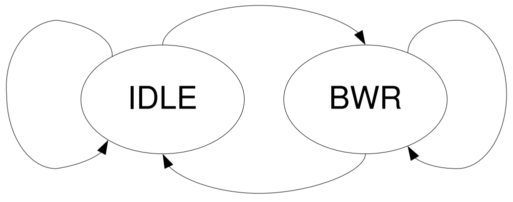


Figure 6.3: request FSM for the merger module

network (network interface initiator) and between communication targets and the network (network interface target).

Each NI is split into two sub-modules: one for the request and one for the response channel. These sub-modules are loosely coupled. Whenever a transaction requiring a response is processed by the request channel, the response channel is notified; whenever the response is received, the request channel is unblocked.

The request path of the NI is built around two registers (Fig.4.3, page 59): one holds the transaction header (1 refresh per OCP transaction), while the second one holds the payload (refreshed at each OCP burst beat). A set of flits encodes the header register, followed by multiple sets of flits encoding a snapshot of the payload register subsequent to a new burst beat. Header and payload content is never allowed to mix, and padding is eventually used. Routing information is attached to the header flit of a packet by checking the transaction address against a LUT. The length of this field depends on maximum switch radix and maximum number of hops in the specific network instance at hand.

The NI performs clock domain crossing, however in order to keep the architecture simple the ratio between network and core clock frequencies needs to be an integer divider.

### 6.2.2 Merger architecture

The detailed architecture of the proposed traffic merger is illustrated in Fig.6.2. Each OCP channel (request and response) is handled by a separate couple of multiplexer/demultiplexer blocks and by a dedicated FSM generating the control signals for the (de-)multiplexing logic.

Control logic has been carefully optimized for minimum complexity, and the resulting FSM is illustrated in Fig.6.3 (for the request path). The state is IDLE whenever no commands are received on the MCmd lines from the processor cores. However, the state stays the same also when single writes or single or burst reads are performed. This is because for all these transactions only one clock cycle is required by the network interface to store them. In fact, the OCP protocol allows to generate a single address even for long burst transfers, the addressing mechanism being implicit in the kind of burst transaction being executed. In contrast, whenever burst write transactions need to be performed, the merger switches to the BWR state, and the control signals are kept stable until all the write data words have been stored into the NI. Then, the merger goes back to the idle state. In essence, in the idle state an arbitration mechanism is active, discriminating between multiple requests from the processor cores. A fixed priority arbitration is currently supported, even though the extension to round robin is straightforward.

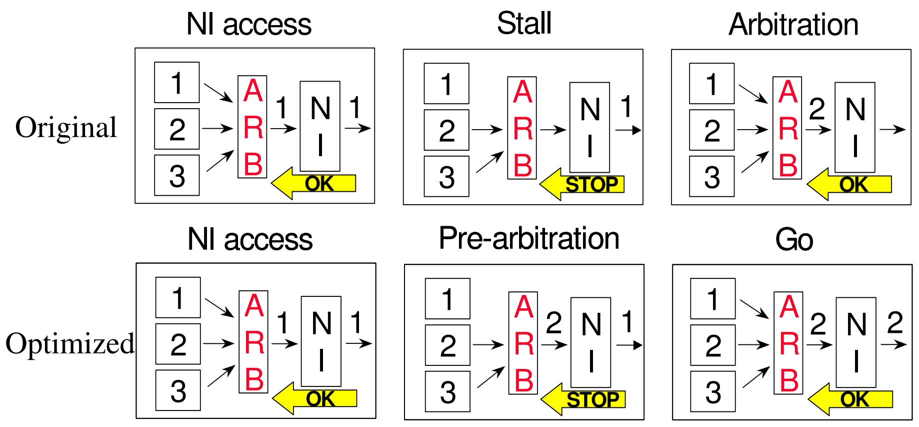


Figure 6.4: Comparison between original and pre-arbitrated architecture

All other state machines (for the response path in the merger, for the request and the response path in the splitter) also exhibit two states and have similar characteristics, and hence are not reported here.

## 6.3 Optimizations

The network interface architecture we are targeting supports multiple oustanding write transactions but only one pending read transaction. The support for multiple outgoing reads would require a deep modification of the network interface and is outside the scope of the chapter. However, since our on-chip network supports posted writes, multiple posted writes could be performed while a read transaction is pending. The merger might allow several cores with pending OCP write transactions to access the NI while the ongoing high priority core is waiting for response data of a read transaction previously stored by the NI. During this state, other cores with pending OCP read transactions would be kept out of the arbitration even though they had the highest priority. Once response data is stored by the waiting processor core, then the normal arbitration priority is restored. The support for this optimization involves the selector signals for the request and the response paths (Fig.6.2) to drive different values. The request path selector selects cores with pending OCP writes, which will be serialized on the network interface based on their priority. The response path selector is instead fixed at the value of the processor core with the pending read, since it has to receive incoming data.

Another optimization was implemented in our merger architecture to efficiently utilize the network interface. The original merger architecture features the behavior in Fig.6.4. Assume that cores 1, 2 and 3 have pending OCP commands. Following arbitration, the command of core 1 is processed. Because of network congestion, the network interface might delay the assertion of the next SCmdAccept. Meanwhile, the MCmd line between the arbiter and the NI is idle since a new arbitration round can take place only on the next SCmdAccept edge. When this latter is finally asserted, it takes a round trip delay before the new MCmd line (the one of processor core 2) can be driven to the NI. In order

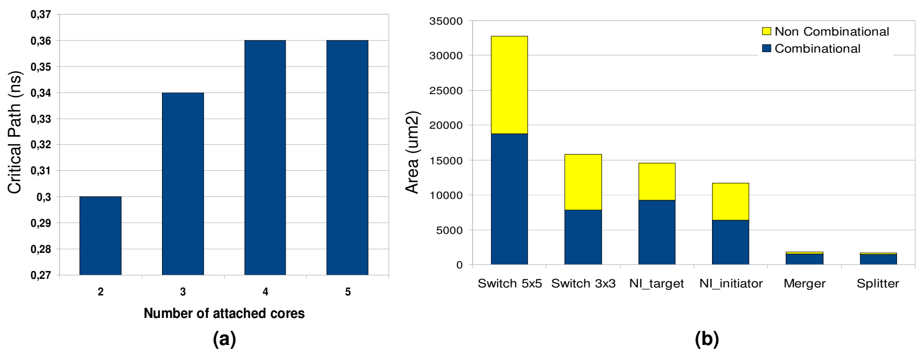


Figure 6.5: Synthesis results for the merger. (a) Critical path (b) Area

to avoid this delay overhead, the optimized scheme illustrated in Fig.6.4 was implemented. Immediately after the transaction of core 1 is stored at the NI, a new arbitration round takes place, so that the OCP signals of the next high priority core immediately drive NI inputs. As a consequence, when SCmdAccept will transition from low to high again, the new command is immediately stored without any additional penalty. The inconvenience of this approach is that if core 1 is the highest priority one, it cannot acquire the network interface for two consecutive transactions in case of contention, since the NI will be re-arbitrated to another waiting processor before a new command from core 1 can be driven. However, full exploitation of network resources was our primary design objective.

Similar optimizations were implemented in the FSMs of the traffic splitter. Here multiple target devices share the same network interface target. Again, only one target core will be accessed for a read transaction at a time. However, while a read transaction is taking place, all other target cores can be accessed for write transactions. These optimizations reflect the NI capability to perform one outstanding read transaction but multiple write-after-read transactions. Finally, the same pre-arbitration mechanism was implemented in the splitter, allowing prompt addressing of a new target core.

## 6.4 Synthesis results

Merger and splitter were synthesized with Synopsys Physical Compiler [26] (thus accounting for placement effects during logic synthesis) to assess their critical path and area, as well as the scalability of these parameters with an increasing number of clustered cores. A 65nm STMicroelectronics technology library was used.

When synthesizing for maximum performance, the critical path results illustrated in Fig.6.5-a are obtained for the merger. Timing paths in the request and in the response sections of the merger are quite close to each other, therefore the critical path is in one module or in the other one depending on the number of inputs and on the effectiveness of the synthesis heuristics. In any case, the critical path is always in the control logic and does not involve the datapath. Moreover, the critical path scales quite smoothly for the realistic cases anal-

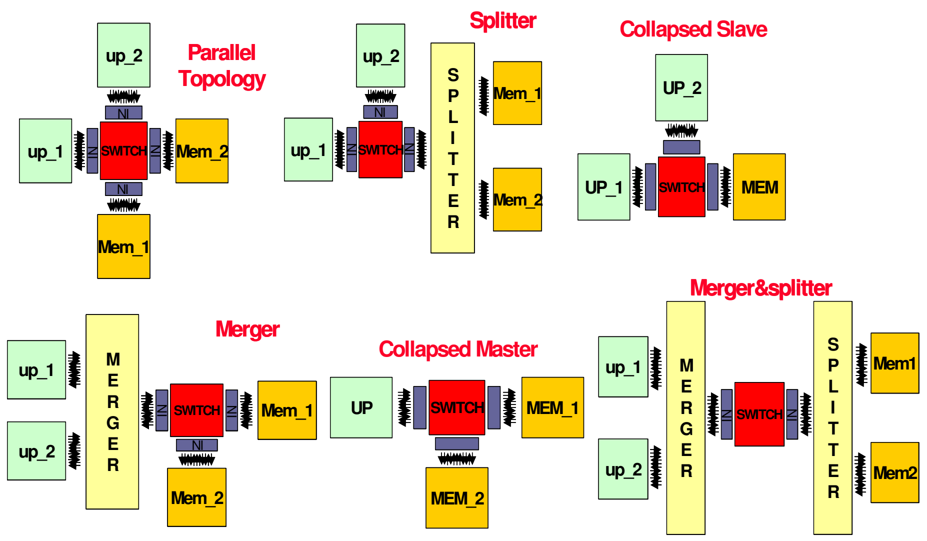


Figure 6.6: Simple test cases for performance characterization

ysed in Fig.6.5-a. As suggested by [24], realistic target frequencies for the OCP section of the system might be around 500 MHz, while the network could be operated at 1 GHz. By synthesizing the NI initiator under these conditions, the critical path of its OCP frontend is 1.30ns and goes to the SData output. Even assuming that in the response path of the merger the maximum delay of 0.37 ns is incurred (but it will be less than that, because this is the delay measured in the control logic and for the highest number of attached cores), the total delay would amount to 1,67ns. If we assume that the master OCP interface does not implement combinational dependencies (hence its inputs are latched), the total delay is just increased by a library setup time (around 0.10 ns) and the worst case delay of the OCP section is well below the target 2ns delay in the worst case. As a consequence, for a realistic target operating frequency of high-performance systems, our traffic merger does not degrade the target frequency of the OCP section. Similar considerations hold for the merger.

As regards the area overhead (Fig.6.5-b), the merger for a cluster of 2 attached cores exhibits an area which is 1/6 that of the corresponding network interface and 1/18 that of a 5x5 switch commonly found in 2D mesh topologies. This paves the way for significant area savings at network level, as will be illustrated in section 6.6.

## 6.5 Basic performance tests

In order to characterize the performance degradation induced by our traffic conditioning blocks and the effectiveness of the proposed optimizations, we set up a few simple test cases which are reported in Fig.6.6. All test cases were modeled in SystemC with clock cycle accuracy. Microprocessor cores were emulated by means of OCP traffic generators, injecting parameterizable traffic patterns. Parameters include burst length of OCP read and/or write transactions and idle

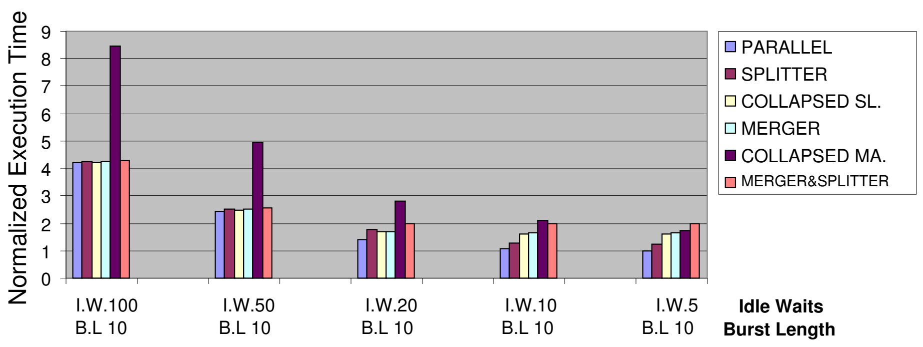


Figure 6.7: Execution time results for the basic topologies

cycles between consecutive bursts. Target cores are instead SystemC models of OCP memories with tunable access latency. Each memory is assumed to be private to a correspondent processor core.

The fully parallel topology (see Fig.6.6) represents the ideal case: each processor core accesses its private memory without any contention. We compare this topology with the ones featuring a traffic splitter for the memories or a traffic merger for the processor cores or both of them. Finally, we also analysed a collapsed topology wherein the two processor cores sharing the same NI initiator are replaced by a single processor core generating twice the number of transactions of the original cores. This test case will unveil whether using a traffic merger is equivalent to a pure serialization of network accesses or not. Similarly, a topology wherein the two memory cores are replaced by a single memory core with doubled capacity is assessed.

We at first generated a mixed traffic pattern consisting of alternating read and write transactions. Burst length was set to 10 data words (similar results were obtained by varying this parameter), and a total of 500 read and 500 write data words were transferred between each initiator core and its corresponding private memory. Execution time results are reported in Fig.6.7, and are normalized to the best case. We notice that as the idleness in the system increases, execution times of the different solutions are very similar to each other, except for the collapsed master case, which takes on average two times longer than the parallel topology (as expected). With a lot of idleness in the system, network accesses of the processor cores can be interleaved without any interference, therefore the price to pay for having mergers and splitters is just the delay for the first serialization, which is negligible.

As the number of idle waits decreases, we have a neat performance differentiation. Interestingly, using a splitter does not impact performance significantly. This is due to the support for write-after-read transactions at the splitter, and hence is an effect of our optimizations. Clearly, using a traffic splitter is more convenient than collapsing the two memory cores in a single memory of increased capacity and with 2 split addressing regions.

This trend is even more apparent when we compare the topology with a merger and the one with collapsed processor cores. The former significantly outperforms the latter as the idleness in the system increases, indicating that for realistic network access traces mergers do not vanish the advantages of par-

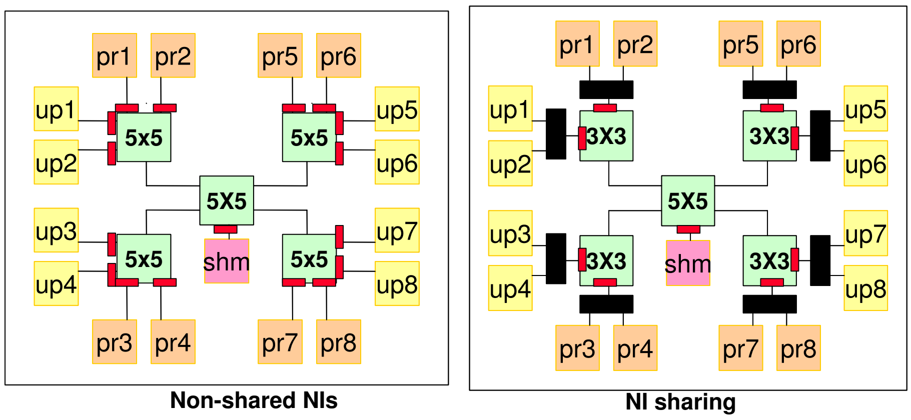


Figure 6.8: Application-specific topologies under test. *uP* denotes processor cores, *pr* private memories and *shm* the shared memory.

allelism. The worst case for the mergers occurs when the processor cores initiate two read transactions at the same time. In fact, the lowest priority one waits for the highest priority one to complete before being processed. In contrast, with only a splitter, such contention occurs closer to the target core, thus improving performance. Finally, the combined utilization of mergers and splitters halves the performance for short idle waits, while it keeps almost the same performance of the parallel architecture from 50 idle waits on.

With a traffic pattern made up of write transactions, we obtain similar results with high levels of idleness. When this latter decreases, all non-parallel architectures exhibit an execution time which is almost two times that of the parallel topology, without any further differentiations. This is an effect of the posted write semantics and of the intensive traffic patterns, which avoid resource under-utilization. Hence, by introducing additional arbitration rounds in the system, parallel writes get simply serialized. Finally, a traffic pattern made up of only read transactions provided execution time results similar to those in Fig.6.7. This behavior is related to the distance of the congestion point from the target destination.

## 6.6 System-level analysis

Let us now take system-level effects into account in determining the area-performance trade-off spanned by architectures with NI sharing. Our scheme can be applied both to application-specific and general purpose NoC topologies, as illustrated hereafter.

### 6.6.1 Application-specific topology

We consider the MPEG-4 application as a case study. In particular, we focus on the MPEG-4 decoder Profile High 4:2:2 Level 5, which supports a resolution of 1920x1088 at a frame rate of 72.3 frames/sec. The workload is assumed to be split among 8 parallel processor cores by means of an horizontal slicing technique. We consider a shared memory system, where each processor core reads

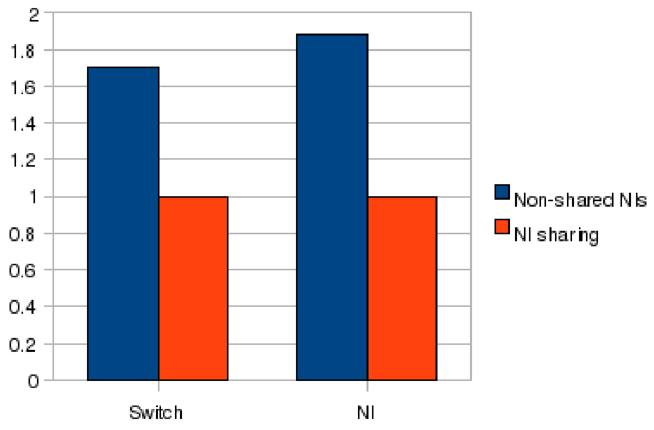


Figure 6.9: NI and switch area reduction in the topology with NI sharing

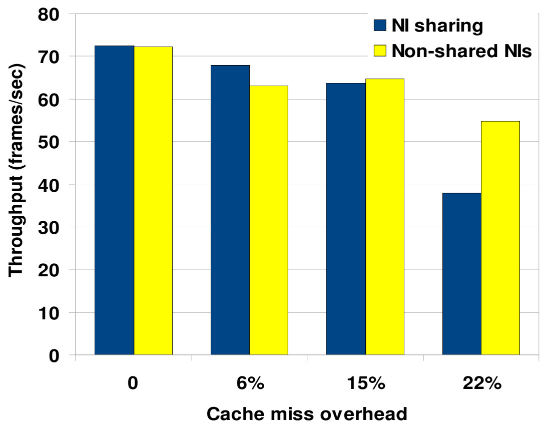


Figure 6.10: Throughput degradation as a function of cache miss overhead with shared and non-shared NIs

encoded data from the shared memory, performs its computation (eventually accessing its private memory for cache line refills) and writes decoded data back to shared memory. Based on the specific MPEG-4 profile and resolution and on real-time constraints, we extrapolated read/write communication requirements of the processor cores from/to the shared memory. The number of cache line refills was kept as a parameter of the traffic pattern, and they will end up degrading the nominal throughput in our assumptions. A synthetic traffic pattern reflecting the above scenario was therefore generated by means of the OCP traffic generators. In practice, since we are focusing on video decoding, the amount of write data will be much higher than that of read data. We set the network clock frequency to be twice the frequency of the OCP section, which is a reasonable assumption for the $\chi$pipes architecture [24].

Since the traffic pattern is shared memory dominated, we selected the H-star topology suggested in [7] for this application and placed the shared memory in the central switch, as illustrated in Fig.6.8-left. Peripheral switches connect processor cores and their associated private memories.The switch radix in all cases is 5. This topology with non-shared NIs was compared with the shared NI topology reported in Fig.6.8-right. Here clusters of two processor cores and those of two memory cores share the same network interface initiator and target, respectively. Because of the central switch, the maximum switch radix (determining the maximum network clock frequency) does not change. However, the amount of hardware resources changes a lot: switches have lower radix on average, the number of NIs decreases from 17 to 9, with only the addition of 4 mergers and 4 splitters.

Physical synthesis of the two topologies provided the area reports of Fig.6.9. NI sharing allows an impressive reduction of total NoC area by 41%. In particular, total switch area is reduced by 41.3% while total NI area by 46.7%. Mergers and splitters alltogether account for only 6% of total area of the topology with NI sharing. These results prove the effectiveness of our technique in providing area-efficient NoC realizations.

Performance results are instead illustrated in Fig.6.10. Nominal throughput is guaranteed for the topology with non-shared NIs in the absence of cache misses. For the same case, NI sharing does not incur any throughput degradation, since there is a strong congestion to access the shared memory, therefore an additional arbitration round at the network boundary does not result in bubbles in shared memory utilization. Moreover, the traffic pattern is dominated by posted write transactions, effectively supported by our mergers and splitters.

We express the impact of cache misses as the percentage stretching of computation time due to cache misses computed as though processor core and memory were directly connected without the network in between. Hence, it is the time needed by a processor core to directly access its private memory for cache line refills. The network latency then introduces an additional and unpredictable overhead.

With an increasing role played by cache misses, the non-shared topology exhibits a progressive throughput degradation, as expected. For 6% and 15% cache miss overheads, we can consider the variability of throughput results as a statistical variation associated with the interleaving of traffic patterns on the network. As the cache miss overhead becomes significant, the impact of NI sharing on throughput becomes apparent.

### 6.6.2 General purpose topology

The 2-D mesh is currently the most popular regular topology used for on-chip networks in regular tile-based architectures, because it perfectly matches the 2-D silicon surface. However, this topology shows poor latency scalability and is very area demanding. Concentrated topologies are a straightforward optimization of 2D mesh topologies[18, 24]. They envision the connection of more cores per switch. This way, less switches are required and the average number of hops is reduced at the cost of a lower bandwidth. The main problem with this solution is at the physical layer: a concentrated architecture makes use of switches with a higher radix, which reduces the maximum operating frequency of the network. On one hand, the total number of cycles for the execution of a given benchmark is expected to decrease with respect to a 2D mesh, while on the other hand if we consider the achievable clock frequency the performance is almost the same or even worse than a 2D mesh. See [24] for further details.

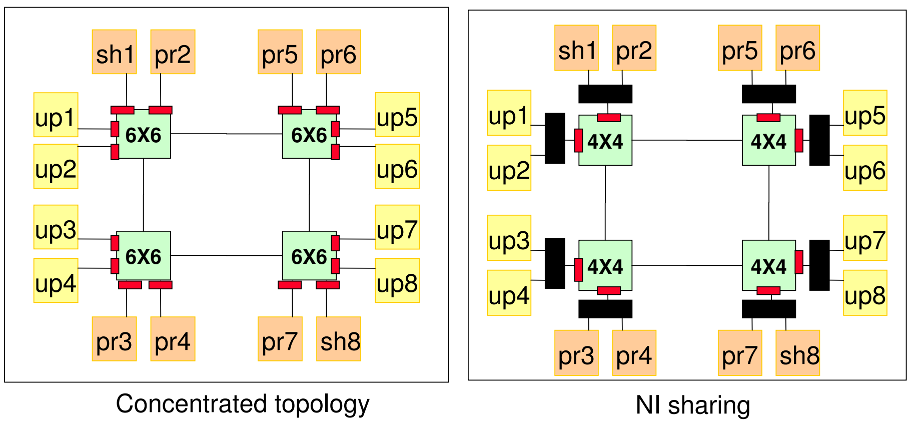


Figure 6.11: Concentrated topologies without and with NI sharing

The proposed traffic mergers and splitters provide an effective workaround for this problem. In fact, they allow to share the same NI, and hence to keep the switch radix unaltered. This prevents a degradation of the maximum operating frequency. As a case study, we propose the topologies illustrated in Fig.6.11. A concentrated topology (resulting from the optimization of a 4x4 2D mesh) is compared with a concentrated variant making use of NI sharing. This time, NI sharing reduces the maximum switch radix, and therefore raises the maximum operating frequency from 1 GHz to 1.4 GHz. Since the ratio between network frequency and OCP section frequency was kept to 2, NI sharing allows OCP cores to be operated at 700 MHz instead of 500 MHz. For what follows, we assume that OCP cores can keep up with these speeds.

Both topologies were synthesized for their target frequencies, thus coming up with realistic area numbers. The only exception concerns mergers and splitters, always synthesized for maximum performance not to degrade the frequency of the OCP section. Total area turns out to be reduced by 42%.

As regards performance, we simulated the traffic pattern of a parallel application with a producer-worker-consumer workload allocation policy. One core on one side of the chip (say, core 1) accesses an I/O device and writes computation data to shared memory $sh1$. All other processor cores (from cores 2 to 7) read their computation data from this memory, perform computation and finally write output data to the output shared memory $sh8$ on the opposite side of the chip, from where another processor core (say, core 8) moves them off-chip. This way, a longer number of hops to read input data is counterbalanced by a shorter path to write output data or viceversa. During computation, cores 2 to 7 access their private memories for cache line refills. A number of conflicts arise in the system when NI sharing is used: core 1 write accesses to $sh1$ are in conflict with read accesses of cores 2 to 7 always from $sh1$. There are also conflicts for cache line refills on the splitters serving private memories. Finally, there is a conflict between read accesses of core 8 from $sh8$ and write accesses of cores 2 to 7 always to $sh8$.

Performance results are illustrated in Fig.6.12. Execution cycle statistics show a negligible 3% degradation of the topology with NI sharing, proving that the merger/splitter architecture optimizations have been quite effective in

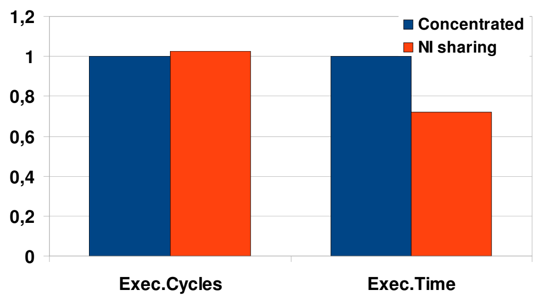


Figure 6.12: Impact of the lower switch radix on execution time of topologies with NI sharing

handling this traffic pattern. However, if we consider the operating frequency boosting that NI sharing enables, we get a 28% improvement of total execution time.

## 6.7 Conclusions

In this chapter it is proposed an area efficient NoC design technique relying on NI sharing. All NI resources are shared by a cluster of initiator or target cores, including buffering resources. This paves the way for significant area savings. Performance degradation of this solution cannot be reduced to a mere serialization of network access requests, in that optimizations implemented at mergers and splitters allow performance of architectures with NI sharing to more closely follow that of fully parallel architectures. Moreover, physical (improvement of maximum operating frequencies) as well as system level effects (centralized slave bottlenecks) play in favour of our solution. Our experimental results prove the effectiveness of NI sharing in reducing NoC area footprint.

# Bibliography

[1] P. Kongetira, K. Aingaran, and K. Olukotun,
"Niagara: A 32-way Multithreaded Sparc Processor",
IEEE Micro,
pp.21-29, 2005.

[2] STn8811A12 Mobile Multimedia Application Processor,
available online: http://www.st.com

[3] SPEAr Plus600 dual processor cores,
available online: http://www.st.com

[4] F.Angiolini, P.Meloni, S.Carta, L.Benini, L.Raffo,
Contrasting a NoC and a traditional interconnect fabric with layout awareness,
in DATE, 2006, pp.124-129.

[5] F.Steenhof et al.,
Networks on Chips for high-end consumer-electronics TV system architectures,
DATE, 2006, pp.148-153.

[6] A.Radulescu, J.Dielissen, K.Goossens, E.Rijpkema, P.Wielage,
An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration,
DATE 2004, pp.873-883.

[7] Kim et al.,
Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC,
Int. Symp. on Networks-on-Chip, pp.30-39, 2007.

[8] A.Pinto et al.,
Efficient Synthesis of Networks on Chip,
ICCD 2003, pp. 146-150, Oct 2003.

[9] T. Ahonen et al.,
Topology Optimization for Application Specific Networks on Chip,
pp.53-60 Proc. SLIP 04.

[10] K. Srinivasan et al.,
An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks,
pp.231-237 Proc. ICCAD 2005.

[11] J. Hu, R. Marculescu,

Application Specific Buffer Space Allocation for Networks on Chip Router Design,
pp.354-361 Proc. ICCAD 2004.

[12] U. Y. Ogras, R. Marculescu,
Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion,
pp.246-253 Proc. ICCAD 2005.

[13] G.Palermo, C.Silvano, G.Mariani, R.Locatelli, M.Coppola,
Application-Specific Topology Design Customization for STNoC,
DSD 2007, pp.547-550.

[14] U. Ogras and R. Marculescu,
Energy and Performance Driven NoC Communication Architecture Synthesis Using A Decomposition Approach,
pp.352-357 DATE 2005.

[15] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo,
Designing Application-Specific Networks on Chips with Floorplan Information,
in International Conference on Computer-Aided Design (ICCAD), pp. 355-362, 2006.

[16] K. Srinivasan, K.S. Chatha, and G. Konjevod,
Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures,
IEEE Transactions on VLSI, 14(4):407-420, 2006.

[17] Dongkook Park Nicopoulos, C. Jongman Kim Vijaykrishnan, N. Das, C.R.,
"A Distributed Multi-Point Network Interface for Low-Latency, Deadlock-Free On-Chip Interconnects",
International Conference on Nano-Networks and Workshops,
On page(s): 1-6, 2006.

[18] Balfour, J., Dally, W.J.,
"Design Tradeoffs for Tiled CMP On-Chip Networks",
ACM International Conference on Supercomputing,
pp.187-198 2006.

[19] Gilabert, F., Gomez, M.E., Lopez, P.J.,
"Performance Analysis of Multidimensional Topologies for NoC",
ACACES 2007, poster session with proceedings at the Hipeac Summer School.

[20] P.Meloni, S.Murali, S.Carta, M.Camplani, L.Raffo, G.De Micheli,
"Routing Aware Switch Hardware Customization for Networks on Chips",
Int. Conf. on Nano-Networks and Workshops, pp.1-5, 2006.

[21] Bertozzi D.,
"Network Interface Architecture and Design Issues",
book chapter from "Networks on Chips: Technology and Tools", edited by Benini L., G.De Micheli,
Morgan Kaufmann, 2006.

[22] P. Bhojwani and R. Mahapatra,

"Interfacing cores with on-chip packet-switched networks",
In Proc. VLSI Design, pp.382-387 2003.

[23] C. A. Zeferino, M. E. Kreutz, L. Carro, and A. A. Susin.,
"A study on communication issues for systems-on-chip",
In Proc. SBCCI,pp.121-126 2002.

[24] F. Gilabert et al.,
"Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework",
Network-on-Chip Symposium, pp.107-116, 2008.

[25] OCP protocol specification, release 2.1.

[26] Synopsys Inc.,
Physical Compiler, www.synopsys.org

[27] S.Stergiou et al.,
"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips",
DAC, pp.559-564, 2005.

# Chapter 7

# Designing Regular Network-on-Chip Topologies under Technology and Architecture Constraints

The execution of many multimedia and signal processing functions has been historically accelerated by means of specialized processing engines [27]. With the advent of multi-processor system-on-chip (MPSoC) technology, performance of hardware accelerators is becoming accessible by combining multiple programmable processor tiles within a multicore system [28]. In addition, the performance of latest application specific integrated processors (ASIPs) [30] together with the high availability of transistors is making the design of custom hard-wired logic always less convenient (time-to-market, respin risks). The underlying principle is that efficient computation can be achieved while only marginally impacting programmability and/or configurability, and architectures can be devised that address the computation requirements of an entire application domain [29].

In this context, tile-based architectures cope effectively with the productivity gap, in that they provide parallelism through the replication of many identical blocks placed each in a tile of a regular array fabric [32, 33, 29]. This approach makes performance scalability more a matter of instantiation and connectivity capability rather than architecture complexity.

Perhaps the most daunting challenge to make MPSoC technology mainstream is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processing cores. This task is on burden of the global intrachip communication infrastructure. Networks-on-chip (NoCs) are generally believed to be the long term solution to the communication scalability issue [34].

Topology selection is a NoC design issue which needs to be addressed in the early design stages and which has deep implications both on final system performance and on physical network feasibility. NoC architectures can be designed with both regular and custom topologies. The primary advantages of a regular NoC architecture are topology reuse, reduced design time, ease of routing, better control of electrical parameters and hence less design respins and a higher degree of performance predictability. The 2D mesh is currently the most popular regular topology used for on-chip networks in tile-based architectures, because it perfectly matches the 2D silicon surface. Unfortunately, 2D meshes show very poor scalability properties in terms of diameter, average minimal hop count and bisection bandwidth and it feels the need to find new topologies.

On one side, we will consider topologies with more than 2 dimensions that are attractive for a number of reasons. First, increasing the number of dimensions in a mesh results in higher bandwidth and reduced latency. Second, the number of dimensions can be traded-off with the number of cores per switch, thus giving rise to concentrated topologies saving network components and trading bandwidth for latency.

On the other side, we will pose attention on Fat-tree. In this case the higher wiring irregularity and the larger switch radix of most fat-tree configurations raise some skepticism about their practical feasibility. Moreover, instead of aiming strictly for speed, designers increasingly need to consider energy consumption constraints, and fat-trees are expected to pay the increased connectivity they provide with a significant area and power cost. In spite of these concerns, constant attention has been devoted to tree-based topologies in the NoC community, proving their superior performance with respect to normal 2D meshes under different kinds of synthetic traffic patterns [20, 2].

Moreover, wiring on a chip comes at a lower cost with respect to off-chip interconnections. However, wiring is also the challenging aspect of these topologies, since their mapping on a bidimensional plane involves the existence of wires with different lengths. Depending on the physical design technique, the more complex connectivity pattern may impact performance, area and power in different ways, such as a decreased operating frequency or a higher link latency.

The objective of this chapter is to assess performance of many topologies ($k$-ary $n$-mesh and $k$-ary $n$-Tree) while considering design constraints posed by real-life HW/SW MPSoC platforms. This makes the analysis more insightful and trustworthy than traditional abstract exploration frameworks based on pencil-and-paper floorplanning considerations. These latter often ignore the presence of non-routable hard IP blocks, the asymmetric tile size, the use of link pipelining to sustain network speed or the dependence of switch critical path on its radix. The relentless scaling of silicon technology to the nanoscale regime is making the interconnect delay issue even more critical and is causing the network critical path to move from the logic to global network links. By leveraging a backend synthesis flow for regular NoC architectures, we characterize the mapping efficiency of a given topology on the silicon layout, targeting a 65nm technology node.

We extract physical parameters from the physical synthesis and expose them to the system-level simulation tool, thus coming up with silicon-aware performance figures.

This chapter is structured as follows. Backend synthesis flow is illustrated in Section 7.1. Topologies under test are presented in Section 7.2 (7.2.2 for $k$-ary $n$-mesh, and 7.2.3 for $k$-ary $n$-Tree), while their physical design is addressed in Section 7.3 and 7.4. Post-Layout analysis for $k$-ary $n$-mesh

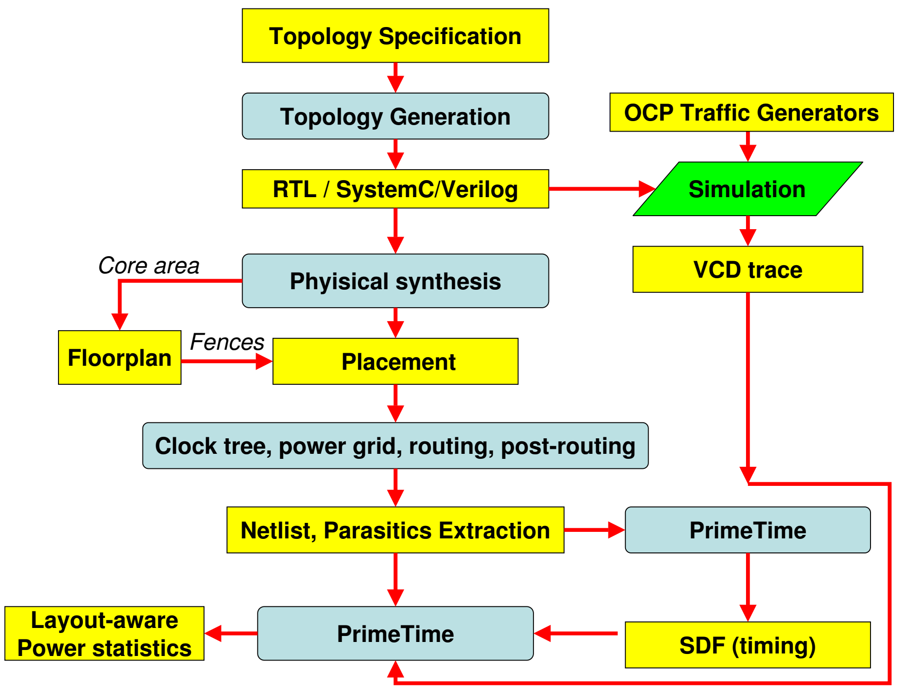


Figure 7.1: Backend synthesis flow.

in Sec. 7.5 and $k$-ary $n$-Tree in 7.6. Finally conclusions are drawn in Section 7.7.

## 7.1 Backend synthesis flow

The practical feasibility of topologies under test was explored by means of a semi-automated design flow spanning from RTL description to layout-level verification. This enables us to explore and validate topologies down to the placement and routing steps, thus accounting for the effects of nanoscale technologies.

The flow starts with a topology specification file (see Fig.7.1) indicating the number of dimensions, the number of hops in each dimension and the number of cores connected to each switch. The XpipesLite architecture is used for NoC synthesis[8]. The file is then processed by a script interacting with the xpipescompiler tool[5], resulting in the generation of self-contained SystemC code for RTL-equivalent simulation and for synthesis.

Synopsys Physical Compiler is used for placement-aware logic synthesis. The technology library is a low-power low-Vth 65nm STMicroelectronics library available through the CMP project [21]. The final place-and-route step is performed with Cadence SoC Encounter. Hard IP blocks representing computation tiles are defined and rendered as black-boxes obstructing an area of $1mm \times 2mm$, reflecting our assumption on a computation tile architecture consisting of a processor core and a memory core. To account for the most challenging scenario, over-the-cell routing is disabled for the hard obstructions.

The designer then performs a first floorplanning phase, where he places the hard black boxes on the floorplan. Fences are defined to limit the area where the cells of each module of the interconnect can be placed. Subsequently, the tool automatically places cells without trespassing the fences.

The next step is clock tree synthesis. We assume that each computation tile and the on-chip network are independent clock domains with their own clock trees. All computation tiles will be operated at the same nominal frequency, while only the network will be allowed to operate at a higher frequency.

At this point, the power supply nets are added. We choose the power grid scheme, which distributes power nets from the topmost metal layer of the chip and which minimizes IR drops. After the power nets have been routed, the tool begins to route the logic wires. After an initial mapping, search and repair loops are executed to fix any violations. As a final step, post-routing optimizations are performed, including crosstalk and antenna effect minimization. Finally, a signoff procedure is run by Synopsys PrimeTime to accurately validate the timing properties of the design.

Post-layout power estimation is performed as follows. PrimeTime PX [6] calculates average power based on toggle rates. The annotated activity comes from RTL simulation-generated VCD files, using parameterizable OCP traffic generators to inject OCP compliant transactions into the network. Zero-delay simulation is then used to propagate switching activity to unannotated nets to calculate the power consumption of the complete design. In practice, we collect from SystemC RTL-simulation the VCD traces expressing the switching activity of the signals at the boundary of the NoC (i.e., signals connecting the tiles to the network interfaces). The tool then performs propagation to the internal nodes and provides average power estimates. We adopted this methodology as a reasonable trade-off between accuracy and characterization time.

## 7.2 Topology exploration

### 7.2.1 NoC architecture

Our realistic topology exploration framework utilizes the the xpipesLite NoC architecture [31]. The switching fabric implements a 2-cycle-latency (one for switch operation and one for traversing the output link), output-queued wormhole-switched router supporting round-robin arbitration on each output port. The implemented flow-control scheme is stall/go [51].

The switch is parameterizable in the number of its inputs and outputs, its link width as well as in the size of the output buffering. For this work, 6-flit buffers are assumed and the link (and flit) width is set to 32 bits.

The network interface (NI) is designed as a bridge between an OCP [1] interface and the NoC switching fabric. Its purposes are the synchronization between OCP and network timing, (de-) packetization, the computation of routing information (stored in a Look-Up Table, LUT) and flit buffering to improve performance. The NI performs clock domain crossing, however in order to keep the architecture simple the ratio between network and core clock frequencies needs to be an integer divider.

[1]Open Core Protocol – standard end-to-end communication protocol

### 7.2.2 Topologies under test: $k$-ary $n$-meshes

The target of this analysis is a *tile based architecture* where each tile is assumed to include at least *one processor and one local memory core*. Therefore, the asymmetric tile size needs to be accounted for when laying out the topology: this puts traditional assumptions on mesh and hypercube wiring in discussion.

Meshes can be referred to as $k$-ary $n$-meshes. The topology has $k^n$ routers in a regular $n$-dimensional grid with $k$ switches in each dimension and links between nearest neighbors. Moreover, the $n$-hypercube topology is a particular case of a mesh where $k$ is always 2. Also, each switch can have one or more tiles attached.

In this Section we devise topologies for two system scales, namely topologies for 16 tiles and topologies for 64 tiles. In the 16 tiles category we analyze a 4-ary 2-mesh (referred to as 2D-mesh from now on) with one tile per switch, a 2-ary 4-mesh (4-hypercube) with one tile per switch, and a 2-ary 2-mesh with 4 tiles per switch. The 4-hypercube is a representative topology for those ones featuring a number of dimensions higher than 2, while the 2-ary 2-mesh illustrates the properties of *concentrated* topologies, connecting more nodes to the same switch.

In the 64 tiles category, we analyze a 8-ary 2-mesh (also referred to as 2D-mesh) with one tile per switch, a 2-ary 6-mesh (6-hypercube) with one tile per switch and a 2-ary 4-mesh with 4 tiles per switch. These topologies were chosen with the same criteria as for the 16 node systems.

Table 7.1 shows some representative data for the studied topologies. Please note that even with 1 tile per switch, 2 switch input and 2 switch output ports are required to connect the tile, since it includes an initiator and a target NI, each with one input and one output port to the switch for receiving/sending data. The initiator NI uses the output port to send out packets and the input port to receive packets carrying read response data. The target NI uses the input port to receive packets carrying write data or read requests for the connected target. Read responses are packetized and sent out through the output port.

Now we enter more in detail in the 16 tiles category.

Both the 2D mesh and its concentrated counterpart feature 2 dimensions and homogeneous interswitch wire lengths. However, such wire length will not be the same in the two topologies due to floorplan constraints reflecting a well-known trade-off for these topologies: spread-around topologies on one hand (large number of low-radix switches) as opposed to more concentrated ones with a small number of high-radix switches *placed further apart* to optimize connectivity with a large number of tiles. In fact, these latter have to be placed around the switches they have to be connected to, thus separating the switches in space. In these two topologies, all the links have to be performance-optimized in order to speed-up the entire topology, hence the associated cost will be more

| | 16 tiles | | | 64 tiles | | |
|---|---|---|---|---|---|---|
| Topology | 4-ary 2-mesh | 2-ary 4-mesh | 2-ary 2-mesh | 8-ary 2-mesh | 2-ary 6-mesh | 2-ary 4-mesh |
| Max Arity | 6 | 6 | 10 | 6 | 8 | 10 |
| Total Switches | 16 | 16 | 8 | 64 | 64 | 16 |
| Tiles x Switch | 1 | 1 | 4 | 1 | 1 | 4 |
| Total Ports | 80 | 96 | 40 | 352 | 384 | 192 |
| Bisection Cut | 4 | 8 | 2 | 8 | 32 | 8 |
| Ideal Diameter | 6 | 4 | 3 | 14 | 6 | 4 |

Table 7.1: Topologies under test: $k$-ary $n$-meshes

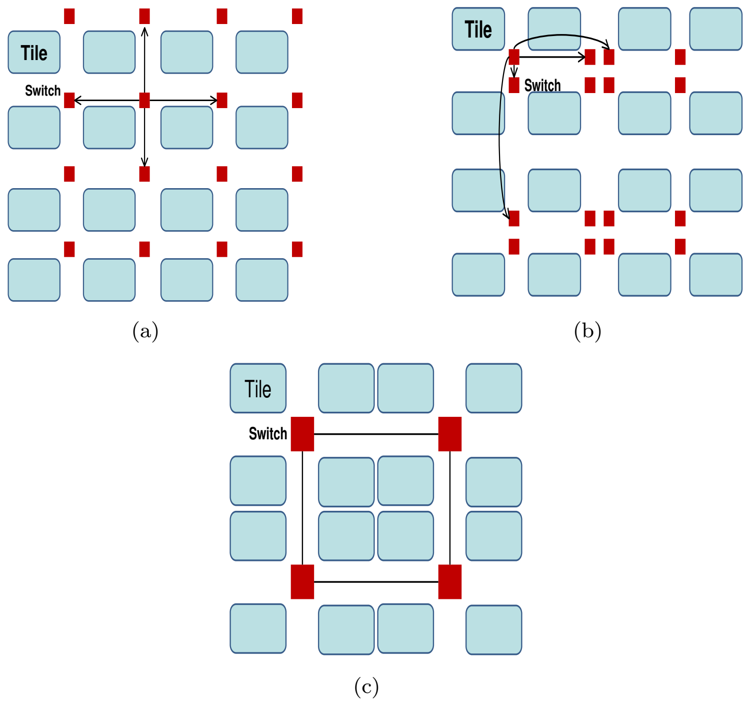


Figure 7.2: Floorplan directives for (a) the 2D mesh, (b) the 4-hypercube and (c) the 2-ary 2-mesh.

relevant in relative terms with respect to the baseline unoptimized topologies.

A different design point is represented by hypercubes. The $n$-hypercube topology is a particular case of a mesh where $k$ is always 2. For 16 tile systems, a 2-ary 4-mesh (4-hypercube) can be obtained from the 2D mesh by increasing the number of dimensions and by reducing the number of switches in each dimension. Interestingly, the maximum switch radix stays the same, only the 4-hypercube has all the switches with the same radix while the 2D mesh has the central switches with a higher radix than the peripheral ones. Of course, the 4-hypercube has larger bisection bandwidth and lower network latency. Unfortunately, this comes at the cost of links with uneven length. In fact, in a mesh with more than two dimensions the links used to connect the dimensions greater than two (often denoted as *express links*) are longer, and this holds for 50% of the 4-hypercube interswitch links.

In theory, the length of a link of the dimension $t$ is generally assumed to be $k^{(d-2)/2}$, where $d$ is equal to $t$ if $t$ is an even number and $d$ is equal to $t+1$ if it is an odd number. Unfortunately, placement and layout constraints put this picture in discussion, thus making it very difficult to predict the impact that the cost of a specific link performance boosting technique might have on the cost metrics of the entire topology.

They are reported in Fig.7.2(a), Fig.7.2(b) and Fig.7.2(c). The asymmetric tile size plays in favor of the 4-hypercube wiring, since the length of the horizontal and of the vertical express links turns out to be comparable to that of horizontal wires in the 2D mesh. This latter also features horizontal and vertical

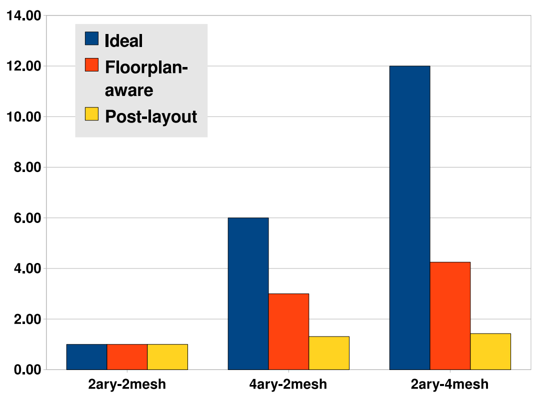


Figure 7.3: Total wire length.

links of unequal length, indicating that the layout regularity often assumed in high-level considerations does not materialize in practice. Our guiding principle for floorplan definition consists of *shortening the longest links in each topology* and of coming up with a scalable floorplanning style. For the 2-ary 2-mesh, we placed the computation tiles around the switch they are attached to, which is an ideal scenario for pipelining interswitch links. In all cases, network interfaces were placed close to their tile but also to the connected switch, so to move the critical path away from these critical links.

As a result of topology synthesis and place-and-route, Fig.7.3 shows the total wiring length for the three topologies (*Post-Layout* curve), normalized to the least wire-hungry topology. It is compared with the results of traditional pencil-and-paper floorplanning considerations. Curve *Ideal* computes wire length based on the ideal formula given above, which only considers the number of hops crossed by a wire. Curve *Floorplan-aware* updates the previous formula with the knowledge of the asymmetric tile size and of switch placement. The ideal analysis largely overestimates the amount of wiring needed for the 4-hypercube. Floorplan awareness allows to account for specific floorplanning techniques that optimize wiring of a given topology, and therefore leads to more conservative estimations of the wiring overhead. However, this is still far away from real-life, where the post-layout report of total wire length gives only a 10% overhead of the 4-hypercube wiring with respect to 2D mesh one and a 43% with respect to the 2-ary 2-mesh. This is because switch-to-switch and switch-to-network interface wiring only accounts for a relatively small percentage of total wiring, ranging from 7% for the 2-ary 2-mesh to 26% for the 4-hypercube. This explains the relatively small total wire length difference between the different topologies. This scenario plays in favor of engineering performance-optimized interswitch links with a possibly minor impact on topology cost metrics.

### 7.2.3 Topologies under test: $k$-ary $n$-Tree

For each topology, we focus on two network sizes: 16 cores and 64 cores, thus getting scalability indications. Like in section 7.2.2 half of the cores are processor cores, while the remaining half consists of the private memory cores of the

| Network size | **16 Cores** | | | |
|---|---|---|---|---|
| Topology | 4-ary 2-mesh | 2-ary 4-tree | Unidir 2-ary 4-tree | Unidir 4-ary 2-tree |
| Switch radix | 5 | 4 | 2 | 4 |
| Total switches | 16 | 32 | 32 | 8 |
| Total Ports | 64 | 112 | 64 | 32 |
| Diameter | 6 | 6 | 3 | 1 |
| Bisect. Cut | 8 | 16 | 8 | 8 |

Table 7.2: Topologies under test: $k$-ary $n$-Tree.

processors. Table 7.2 shows some representative data for the studied networks.

In the 16 cores category, we analyze a 4-ary 2-mesh while in the 64 cores category, we analyze an 8-ary 2-mesh. In all cases, we refer to them as **2D meshes**.

**Fat-trees (FT)** are a particular sub-set of a family of topologies known as multistage interconnection networks (MINs). In MINs, switches are structured in multiple stages. Each switch can only be connected to switches belonging to their previous or to their next stage. Cores are connected to the switches of the lowest stage. For instance, Fig. 7.4(a) depicts a MIN with three stages and eight interconnected cores.

Fat-trees are based on complete trees, but differ from them for preserving bandwidth near the root. For this purpose, the switch radix grows as we move up to the root, which makes the physical implementation impractical. Therefore, some alternative implementations have been proposed to use switches of fixed arity [19].

In particular, we focus on a specific implementation: the $k$-ary $n$-trees (see Fig.7.4(a)), a parametric family of regular multistage topologies. The number of switch stages is $n$ and $k$ is the arity or the number of links of a switch that connect to the previous or to the next stage (i.e., the switch radix is $2k$). Links of a switch can be classified as ascending or descending, depending on whether they connect to switches located in the higher or lower stage, respectively. All the switches have the same number of ascending and descending links. A $k$-ary $n$-tree is able to connect $N = k^n$ processing nodes using $nk^{n-1}$ switches and $2nk^n - k$ unidirectional links.

Routing in Fat-trees is performed in two phases: ascending and descending. In the ascending phase, several minimal paths for each source-destination pair are possible. Packets are forwarded upwards in the tree until one of the *nearest common ancestors* between source core and destination core is reached. At this point, the descending phase is started. This latter is deterministic by construction and the path used in this phase depends on the nearest common ancestor that has been reached in the ascending phase. To provide a deterministic routing algorithm for the ascending path, a unique path must be selected for each origin-destination pair among all the possible ones. The deterministic routing algorithm for fat-trees used is the one presented in [15].

In this routing algorithm, during the ascending phase consecutive destinations are shuffled among the different ascending links of the switches. Figure 7.4(a) shows how destinations are distributed in each switch among the different

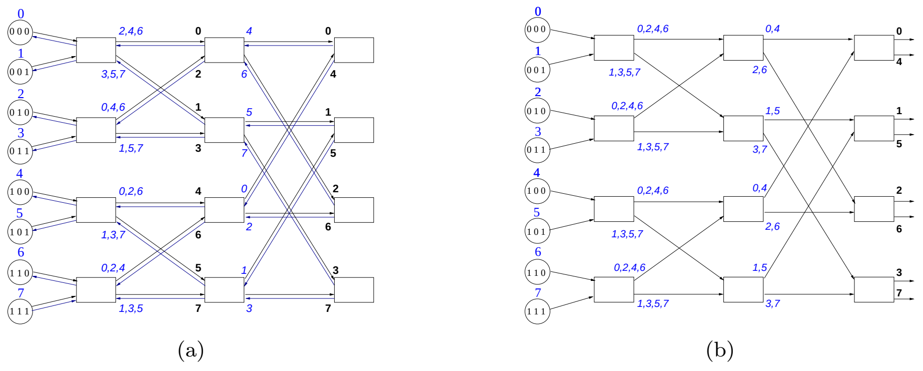


Figure 7.4: **(a)A 2-ary 3-tree topology. (b)A RUFT derived from a 2-ary 3-tree. Each switch port shows its reachable destinations.**

ascending links: each ascending port is labeled in italics with the destination cores that are reachable through it. Also, Figure 7.4(a) shows how destinations are distributed in the descending phase; in this case, descending ports show in bold their sets of reachable destinations. As can be seen, each descending link is only used by a single destination. We analyze a 2-ary 4-tree for a 16-core system.

**Reduced Unidirectional Fat-Tree (RUFT)** is a topology resulting from the simplification of the above mentioned $k$-ary $n$-tree when using the deterministic routing algorithm proposed in [15]. RUFT was first presented in [16] as a conceptual topology scheme, without any implementation analysis. When using this deterministic routing algorithm, the whole descending phase can be reduced to a single long link that connects the output ports of the switches of the last stage with the input port of the corresponding destinations. In this way, switches become unidirectional and all packets must reach the last stage of the network (Figure 7.4(b)).

Although the use of long links may compromise the feasibility of this topology, all the hardware resources related to the descending phase are reduced to these long links, simplifying the switch architecture. The resulting topology resembles an unidirectional butterfly, with a permutation of the reachable destinations from the last stage.

An unidirectional reduced $k$-ary $n$-tree is able to connect $N = k^n$ processing nodes using $nk^{n-1}$ unidirectional switches and $nk^n$ unidirectional links.

When evaluating RUFT, we consider two different topologies for each network size. The first ones are the unidirectional networks resulting from the simplification of standard $k$-ary $n$-tree fat-trees, that is, we analyze an unidirectional 2-ary 4-tree in the 16 cores category, while we analyze an unidirectional 2-ary 6-tree in the 64 cores category.These unidirectional networks have the same number of switches of the original fat-trees but, as can be seen in Table 7.2, the switch radix is reduced to one half.

This leads us to explore an alternative RUFT implementation, denoted as **S(implified)-RUFT** hereafter, trading switch radix (which we make equal to that of the fat-tree) for the switch count. In this direction, we define an unidirectional 4-ary 2-tree for the 16-core system and a 4-ary 3-tree for 64 cores.

| | 16 tile | | | 64 tile | | | | |
|---|---|---|---|---|---|---|---|---|
| Topology | 4-ary 2-mesh | 2-ary 4-mesh | 2-ary 2-mesh | 8-ary 2-mesh | 2-ary 6-mesh | 2-ary 6-mesh High-Speed | 2-ary 4-mesh | 2-ary 4-mesh Reduced |
| Max. switch arity | 6 | 6 | 10 | 6 | 8 | 8 | 12 | 12 |
| Post-synthesis freq. | 1 Ghz | 1 Ghz | 850 Mhz | 1 Ghz | 900 Ghz | 900 Ghz | 790 Mhz | 790 Mhz |
| Post-layout. | 786 MHz | 640 Mhz | 600 Mhz | 786 Mhz | 640 Mhz | 786 Mhz | 500 Mhz | 500 Mhz |
| Core speed (max. 500) | 393 MHz | 320 Mhz | 300 Mhz | 393 Mhz | 320 Mhz | 393 Mhz | 250 Mhz | 500 Mhz |
| Cell Area | $949k\ \mu m^2$ | $1108k\ \mu m^2$ | $733k\ \mu m^2$ | $4461k\ \mu m^2$ | $7356k\ \mu m^2$ | $22784k\ \mu m^2$ | $2610k\ \mu m^2$ | $2611k\ \mu m^2$ |
| Power | 0.67 W | 0.64 W | 0.32 W | — | — | — | — | — |
| Latency on top dimensions | | | | | | | | |
| Dimension 3 | — | — | — | 1 | 1 | 2 | 1 | 1 |
| Dimension 4 | — | — | — | 1 | 1 | 2 | 2 | 2 |
| Dimension 5 | — | — | — | 1 | 2 | 2 | — | — |
| Dimension 6 | — | — | — | 1 | 3 | 3 | — | — |

Table 7.3: Physical Parameters of Topologies under Test

This way, the total number of switches is lower than that in the original fat-tree, as reported in Table 7.2.

Please observe that we were not able to perform the same switch count reduction in the original bidirectional fat-tree since this would have led to switches with an overly high radix (i.e., 8x8), which results in a significantly lower maximum operating frequency [12]. In essence, moving from performance considerations, the topologies explored in our experiments have a switch radix which is always lower than 5.

## 7.3 Physical design: $k$-ary $n$-meshes

Link latency and maximum achievable frequency are key parameters to determine performance, area and power of each topology. However, they can only be quantified by post-layout analysis. This motivates our bottom-up approach to topology exploration. Due to synthesis time constraints, real physical parameter values were obtained for 16 tile systems, while some for 64 tile systems were extrapolated based on the synthesis experience on the smaller systems and on ad-hoc scalability experiments.

### 7.3.1 16 tile networks

Network building blocks have been synthesized in isolation for maximum performance. Post-synthesis achievable frequencies are reported in Table 7.3 - 3rd row. They only account for timing paths in network logic and ignore those going through switch-to-switch links. We always found the critical paths to be in the switches and never in the network interfaces, and this explains why the network speed closely reflects the maximum switch radix of each topology.

When post-layout speed is considered, we observe that inter-switch wiring has caused a significant performance drop for all topologies, depending on the wiring intricacy of each of them. As reported in Table 7.3 - 4th row, the more complex connectivity pattern of 2-ary 4-mesh results into a larger frequency drop than the 2D mesh. The 2-ary 2-mesh pays its lower number of switching resources with a larger switch-to-switch separation, and hence with a severe degradation of network performance due to link delay.

Since frequency-ratioed clock domain crossing is implemented in xpipesLite network interface, network speed affects IP core speed. For this latter, a maximum value of 500 MHz is assumed in the context of multi-core embedded microprocessors. In spite of the post-layout speed drop, IP cores cannot sustain the network speed just at the same and therefore a divider of 2 is applied (Table 7.3 - 5th row).

The total larger number of switch I/O ports used by the 4-hypercube well motivates its larger area footprint than the 2D mesh. Cell (floorplan) area is considered in Table 7.3 - 6th row, while chip floorplan area is not reported since no specific optimizations were applied to it. Although the 2-ary 2-mesh has half the number of switches, its area is not halved as well, due to the fact that those fewer switches have a larger radix.

### 7.3.2 64 tile networks

For 64 tile networks, we had to assume a different physical design technique than for smaller scale systems. In fact, should switch-to-switch link delay still impact overall network speed, this latter would become unacceptably low. For this reason, link pipelining becomes mandatory. While it might turn out to be expensive for 16 tile systems, for larger networks it can break long timing paths across on-chip interconnects and sustain network speed even in presence of long links.

The number of pipeline stages depends on the link length on the layout. As a first approximation, layouts for 64 node topologies can be obtained in a modular way by replicating those of 16 node topologies 4 times. So, for instance, the layout of a 2-ary 6-mesh can be derived from that of the 4-hypercube while from a 2-ary 2-mesh it is possible to draw the layout of a 2-ary 4-mesh. Since in the 64 tile systems the switch radix is larger, the main inaccuracy may only regard the size of some routing channels. The 2D mesh is an exception to this scaling policy, in that its extension to 64 tiles is straightforward and retains the switch radix.

From the layouts, we were able to project link lengths for each topology dimension and, consequently, link latencies when link pipelining is applied. Table 7.3 reports latency results for the top dimensions, since the lower ones always feature 1 cycle latency. This way, the same post-layout frequency of the originating 16 tile topologies could be retained. The only exception regards the 2-ary 4-mesh, where the maximum frequency is determined by the large switch radix and not by link delay anymore.

The mapping function between wirelength and link delay has been experimentally derived. We placed two switches in the layout at increasing distance, and measured the post-routing critical path going through the switch-to-switch link as a function of the distance. The resulting curve is illustrated in Fig.7.5. We have two physical design options. Repeater stages can be used in link segments to speed up signal propagation, resulting in an almost linear increase of critical path delay. This technique also allows to limit the number of link pipeline stages. In contrast, some previous works advocate the use of unrepeated global wires to avoid an exponential increase in area and power. As can be observed from Fig.7.5, the consequence would be a steep increase of link latency, and hence of retiming stages. We leave the exploration of the most power-efficient solution for previous work, and consider the combined use of repeater and retiming stages in this work.

In order to point out the key role of physical design techniques in determining system performance, we consider two topology variants featuring the same connectivity pattern of presented topologies but different physical parameters.

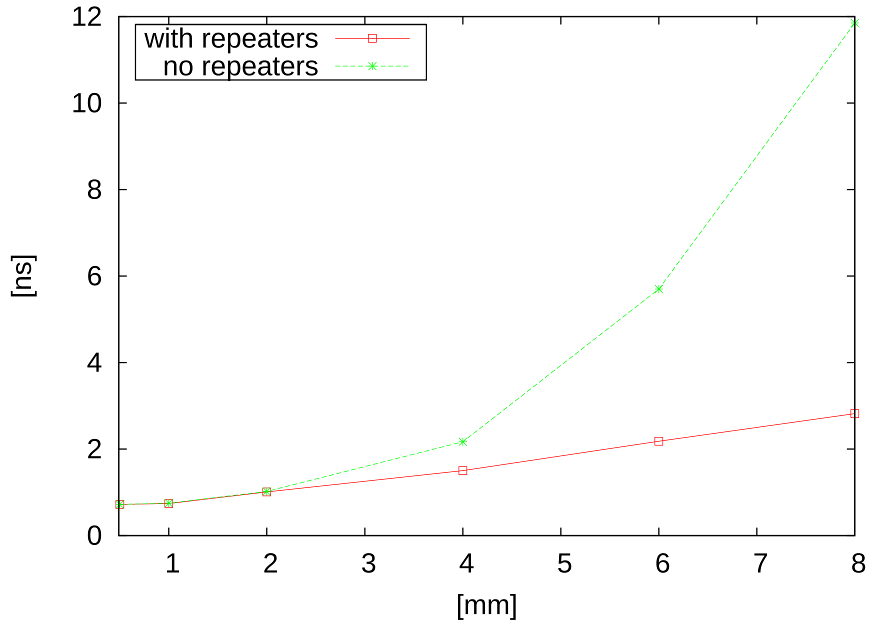


Figure 7.5: Switch-to-switch distance and associated critical path.

The *high-speed* version of the 2-ary 6-mesh topology illustrated in Table 7.3 (7th column) differs from the reference 2-ary 6-mesh only for the higher post-layout operating frequency, which was made equal to that of the 2D mesh. This was allowed by an aggressive use of link pipelining, applied also to the links in the third and forth dimensions. Since each pipeline stage is not just a retiming stage but also a flow control stage, it needs to have a 2 slot buffer, and this leads to the significant area overhead illustrated in Table 7.3 with respect to the baseline 2-ary 6-mesh. The same design technique could not be applied to the 2-ary 4-mesh, since the high switch radix poses an upper bound to the performance optimization achievable by link pipelining.

On the other hand, maximum speed of the 2-ary 4-mesh is so low that a modification of the frequency ratio at the network interface might become convenient. In fact, if we change the ratio from 2 to 1, we enable IP cores to run at the same speed of the network. So, the network slowdown might be offset by speeding up the IP cores. We refer to this topology with NI divider set to 1:1 as the *reduced* 2-ary 4-mesh.

## 7.4 Physical design: $k$-ary $n$-Tree

This section discusses the criteria for floorplan design of the topologies under test. Synthesis time constraints forced us to limit the physical design to 16 core systems. However, projections of 64 core system designs will be extrapolated as well, thus getting useful scalability indications.

The 2D mesh floorplan is straightforward (see Fig.7.6(a)) due to its regular grid structure matching the 2D silicon surface.

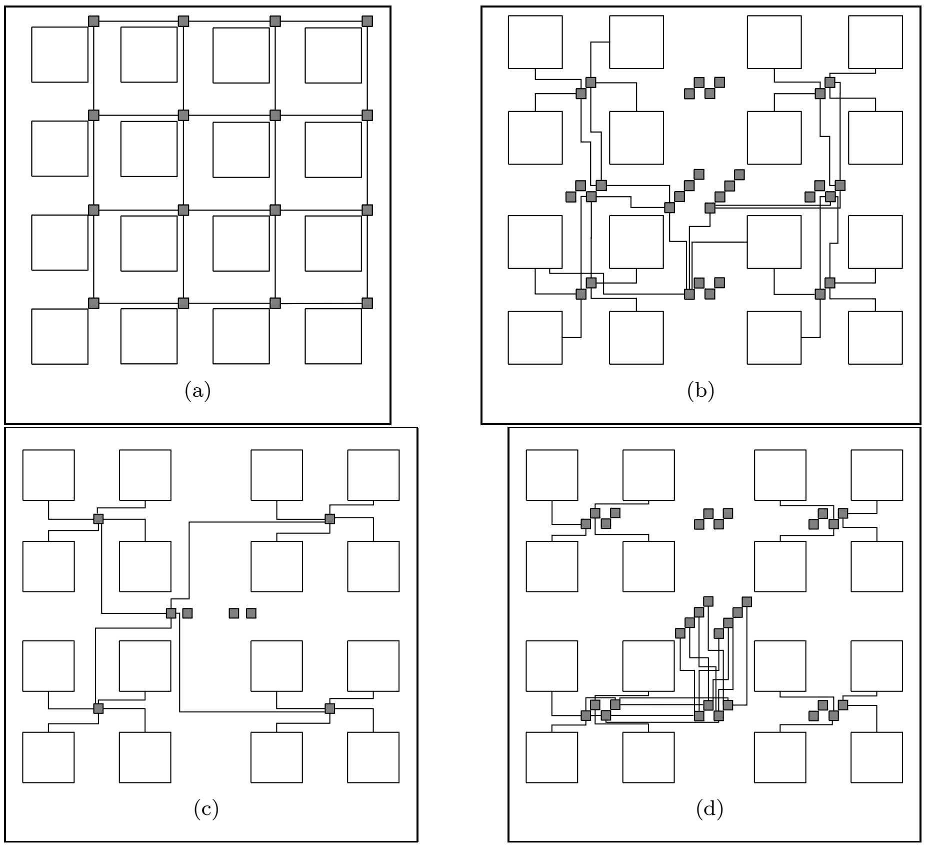


Figure 7.6: Floorplans of topologies under test. a) 4-ary 2-mesh b)2-ary 4-tree RUFT c)4-ary 2-tree S-RUFT d)2-ary 4-tree FT. Only the main wiring patterns are reported.

Things are more complex for 2-ary 4-tree FT (Fig.7.6(d)). The topology consists of 4 switch stages with 8 switches each. Our floorplanning strategy was to minimize wirelength between consecutive switch stages. For this reason, cores are clustered in groups of four and the connected switches (of the first and second stage) are placed in the middle of each cluster. The third switch stage is split into 2 subgroups and placed between the upper and lower clusters. Each subgroup serves its relative counterpart from the first and second stage. The last switch stage is located in the center of the chip. The presented layout exhibits equalized wirelengths between the second, the third and the last stage of switches. Another appealing characteristic is the straightforward scalability of this floorplanning strategy to 64-core systems.

The 2-ary 4-tree RUFT (Fig.7.6(b)) is a novel unidirectional fat-tree which has never been laid out before. This time, a switch belonging to the last stage is directly connected to the network interface of a core. This link is viewed in [16] as the intuitive weakpoint of the layout of this topology. To go around this problem, our floorplanning directive in this case is to minimize the wirelength of this critical set of links. Thus, switches from the last stage are positioned in the middle of each 4-core cluster. Obviously, also the first stage has to be close to the appropriate cores. Therefore, it is placed above and below the middle of the chip between two neighboring clusters, so to equalize the link length and keep the delay as homogeneous as possible on the wires of the first stage. As the third stage has to be connected to the last one and to the second one, two groups of switches belonging to the third barrier are placed at the left and at the right of the chip center. This also achieves an easy connection with the second stage, which is positioned in the center of the chip. An interesting property of the presented floorplan is that the link length is kept almost constant on a stage-by-stage basis. Unfortunately, although the aforementioned layout elegantly solves the placement problem for this 16-core RUFT, its scalability to 64 cores is not straightforward and as efficient. In this sense, it can be considered an ad-hoc floorplan for 16 connected nodes.

Finally, the floorplan for a 4-ary 2-tree S-RUFT is illustrated in Fig.7.6(c). Although the number of switch stages is small (just 2), this is a challenging topology from a physical layout viewpoint. The problem stems from the fact that each switch of the first stage is directly connected to all the switches of the second stage. Moreover, a second stage switch is connected to network interfaces of cores, since this is again a unidirectional topology. Following the same floorplanning strategy of the 2-ary 4-tree RUFT, a switch from the last stage has to be placed in the middle of a 4-core cluster.

Unfortunately, in this case the switch has to be interconnected also to all the switches of the first stage, which should be necessarily placed in the center of the chip to equalize link length. This results in very long links going from each cluster to the switches at the center of the layout. As we will show later on, this dramatically impacts the achievable post-routing performance of this topology.

The 2D mesh is easily scalable to 64 cores, resulting in an 8-ary 2-mesh. Also the floorplan of the fat-tree (which becomes a 2-ary 6-tree FT) can be easily scaled with our strategy. In fact, a 64 node topology can be viewed as built up by four clusters of 16 cores with two additional switch stages connecting them to each other. The four clusters can be internally placed as previously described. This approach features a high level of modularity along with an adequate link length scalability. Moreover, all the link lengths from the fourth stage (the last stage in the 16-core system) to the two additional ones in the 64-core system can be tuned to be the same in the final layout.

For the 2-ary 6-tree RUFT, again 4 clusters of 16 cores are formed. However, the main issue is that unlike the usual fat-tree, the center of each cluster is now occupied by the second stage of switches and not the fourth one (i.e., the last in the 16-core system). Therefore, the fourth stage being scattered on the edges of the chip, the connection with the additional switch stages is not equalized, leading to links of uneven lengths. The problem stems from the fact that the layout was customized for a 16-core system and afterwards a modular approach was applied in order to re-use previous effort. A better floorplan could be obtained by customizing it for a 64-core system, which is not as intuitive as for a 16-core system and falls outside the scope of this chapter.

Finally, the S-RUFT topology now becomes a 4-ary 3-tree (3 switch stages). Without going into further details, this floorplan turns out be as inefficient as that of the RUFT topology, and results in very long links between the last stage of switches and the network interface of connected cores.

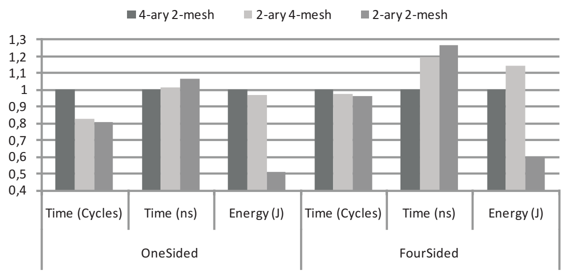


Figure 7.7: Normalized execution time for 16 tile topologies.

## 7.5 Post-Layout analysis: $k$-ary $n$-mesh

In order to simplify topology analysis, we assumed a workload distribution between the tiles which de-emphasizes the role of the topology mapping algorithm. In fact, we consider a parallel benchmark consisting of one or more producer tasks, a scalable number of worker tasks and 1 or more consumer tasks. Every task is assumed to be mapped on a different hardware tile. The producer task(s) reads in data units from the I/O interface of the chip and distributes it to the worker tasks. There are no constraints on which worker tile has to process a given data unit. Output data from each worker tile is then collected by one or more consumer tiles, which write them back to the I/O interface. The following assumptions were made on the I/O interface. A maximum of 8 I/O ports is assumed for 64 tile systems, each one used for input or for output. This number was reduced to 2 I/O ports for 16 tile systems. Such ports are accessed through sidewall tiles. The mapping of producer(s) and consumer(s) tasks is therefore constrained to these tiles. This I/O architecture is compliant with that of commercial embedded microprocessors, such as [29]. We set latency for access to the off-chip I/O devices as a function of their frequency. We considered 20 cycles at 500 MHz and 15 cycles at 350 MHz.

While insensitive to worker tile mapping on the topology, our benchmark is still sensitive to I/O tile mapping on the chip periphery. For this reason, two scenarios are considered:
*OneSided*: all the I/O tiles are placed on the same side of the chip. This mapping has a high probability of I/O streams collision.
*FourSided*: I/O tiles are spread across the four sides of the chip. For 64 tile systems, at least one input and one output is placed at each side. For 16 tile systems, I/O tiles are placed at opposite sides to balance the average number of hops to input and output tiles for all workers.

### 7.5.1 16 tile

Figure 7.7 shows performance of 16 tile topologies in clock cycles and elapsed time. The left side shows results for *OneSided* mapping, while the right side shows results for *FourSided* mapping. In *OneSided*, the hypercube (2-ary 4-mesh) reduces total number of cycles by 27.4%. In this mapping, inputs and outputs are placed at the top of the chip. Therefore, path length is very irregular, as tiles located at the top of the chip can reach input and output through a shorter path than the tiles located at the bottom. This makes topologies with

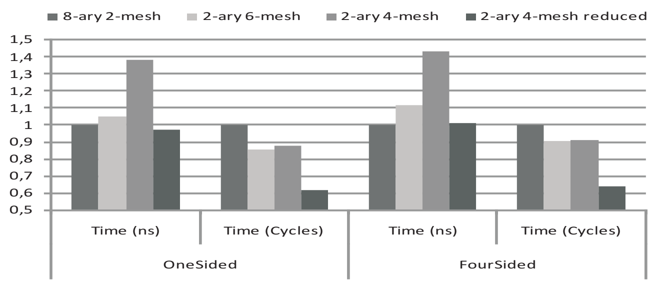


Figure 7.8: Normalized execution time for 64 tile topologies.

higher network diameter more sensitive to this effect. Moreover, the probability of collision between I/O streams is quite high, thus penalizing topologies with fewer dimensions. The concentrated hypercube (2-ary 2-mesh) reduces cycles only by 1.6% over the hypercube, despite its lower diameter. The main reason for this lies in the chip I/O: as the bottleneck introduced by the topology is alleviated, the external I/O bottleneck arises. So, the maximum improvement that can be achieved in the 16 tile system is bounded by I/O speed. On the other hand, *FourSided* mapping (see Figure 7.7) reduces I/O streams collision probability while providing homogeneous path length, thus decreasing performance differences among topologies.

Unfortunately, these results become irrelevant when considering the real operating frequency of each topology. While in *OneSided* mapping the reduction of cycles of the 4-hypercube barely compensates for its lower frequency, in *Foursided* mapping it is not enough, and the 2D mesh turns out to be the best topology overall.

Finally, Figure 7.7 shows the energy consumed by each topology. These numbers are measured on the post-layout netlists illustrated in Section 7.3. While the 4-hypercube consumes almost the same or even more energy than the 2D mesh depending on the I/O tile mapping, the concentrated hypercube shows superior energy saving properties (from 40 to 50% less than the 2D mesh). .

### 7.5.2 64 tile

Fig.7.8 shows performance of 64 tile topologies for both mappings in cycles and real elapsed time. The projected link latencies from Table 7.3 are considered for each topology. The trend observed in 16 tile systems with respect to the I/O tile mapping seems to be confirmed also in 64 tile systems. With this system scale, the impact of the different operating frequencies of the topologies is even more apparent. For instance, performance in cycles of both non-reduced hypercubes seems quite similar, but when frequency is taken into account, performance results are very different.

Again, the 2D mesh outperforms both the non-reduced hypercubes, due mainly to two reasons. The first one is of course the decreased operating frequency of the hypercubes, which is the price to pay for the more intricate physical routing (6-hypercube) or for the longer links used to interconnect few network resources sparse all around the chip (4-hypercube). Performance improvements in cycles are not such to offset the lower operating speed. However,

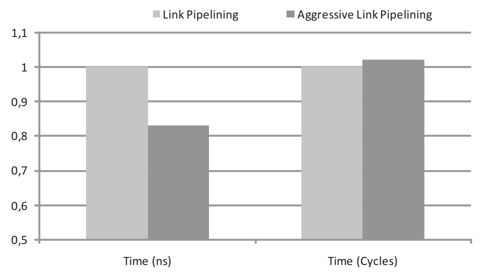


Figure 7.9: Relative performance comparison between 2-ary 6-mesh and its high-speed variant leveraging aggressive link pipelining.

one might expect a larger cycle reduction from an hypercube, which conflicts with the real 10% reduction measured in our simulations. This is due to the fact that the systems under test are I/O constrained, since computation tiles spend around 50% of their time waiting to send data to the consumer tile. This constraint introduces an upper bound to topology-related performance optimization. Removal of the I/O bottleneck has to be considered as mandatory to achieve performance differentiation between topologies.

An interesting effect can be observed when looking at the results of both 2-ary 4-mesh topologies. In the reduced one, clock domain ratio is set to 1, so NoC and tiles work at the same frequency, allowing to increase tile speed to 500 MHz. In this case, although the number of cycles is greatly reduced (29% less cycles than the 2D mesh), the low network frequency compensates for that, so performance is very similar. However, the reduced 2-ary 4-mesh requires 4 times less switches than the 2D mesh, half the number of ports and works at half the frequency, so power requirements are going to be lower, while preserving performance.

Finally, the effects of an aggressive utilization of re-timing stages over performance was analyzed for the 2-ary 6-mesh. Figure 7.9 shows the performance of 2-ary 6-mesh and of its high-speed variant in both cycles and elapsed time. As can be observed, the high-speed 2-ary 6-mesh slightly increases the number of cycles. However, the increased operating frequency is high enough to reduce elapsed time by 27% over the baseline topology. This might cause the high-speed 2-ary 6-mesh to become competitive with the 2D mesh performance-wise, but a significant area and power overhead needs to be taken into account, associated mainly with the additional buffers used in retiming stages.

## 7.6 Post-Layout analysis: $k$-ary $n$-Tree

| **Topology** | **Max Arity** | **Post-. Synthesis.** | **Post-place&route** | **Area 16 cores** | **Area 64 cores** | **Area overhead for Retiming** | **Tot. Wire Length** |
|---|---|---|---|---|---|---|---|
| 4-ary 2-mesh | 5x5 | 0.9 $ns$ | 1.19 $ns$ | 802k $\mu m2$ | 3691k $\mu m2$ | 0% | 8081 $mm$ |
| 2-ary 4-tree RUFT | 2x2 | 0.6 $ns$ | 1.15 $ns$ | 795k $\mu m2$ | 4769k $\mu m2$ | 37% | 8370 $mm$ |
| 2-ary 4-tree FT | 4x4 | 0.8 $ns$ | 1.29 $ns$ | 1280k $\mu m2$ | 8076k $\mu m2$ | 19% | 12389 $mm$ |
| 4-ary 2-tree S-RUFT | 4x4 | 0.8 $ns$ | 2.1 $ns$ | 400k $\mu m2$ | 2400k $\mu m2$ | 112% | 7346 $mm$ |

Table 7.4: Physical synthesis reports

**Timing**
In all topologies, the network building blocks have been synthesized for maximum performance. The post-synthesis critical paths (ignoring place&route effects) are reported in Table 7.4, 3rd column. We found the critical path to be always in the switch and to reflect the maximum switch radix of the topology. Obviously, the lower switch radix of the 2-ary 4-tree RUFT results in a much shorter critical delay. We then iterated place&route starting from the post-synthesis target frequencies.

For a 16 core system, we assessed the integration of a link pipelining technique in the backend synthesis flow overly expensive. For this system size, we aim at analysing whether the delay of switch-to-switch links already impacts overall NoC performance or not. For what follows, the mismatch between wiring of a topology and the 2D silicon layout will result in increased clock cycle time after place&route.

Timing closure was achieved at the post-layout speed reported in the 4th column of Table 7.4. Performance degradation turns out to be very significant, thus pointing out the critical role of interconnects. In fact, for all topologies (and even for the 2D mesh) the critical path goes through the switch-to-switch links. While data/flit wires are sampled at the input and output port of the switch, flow control wires go through the FSM of the flow control stages at switch I/O. As a consequence, these control wires go through logic gates whose delay adds up to the link delay, determining the critical path. The impact of the link delay is evident, in that the critical delays of the topologies are differentiated by the longest link in that topology.

The 2-ary 4-tree RUFT wastes part of its speed with respect to the 2D mesh due to the use of longer links. In practice, the theoretical performance enhancement associated with a lower switch radix does not materialize after place&route, but has served as timing margin against physical degradation effects.

The 2-ary 4-tree FT has incurred a lower degradation than the 2-ary 4-tree RUFT, but its post-synthesis performance was lower, therefore it ends up running even slower than the 2D mesh.

Finally, for the 4-ary 2-tree S-RUFT the above effects are even more apparent due to the longer wires that are needed to connect a low number of switching resources sparse all around with each other. The lower area footprint is achieved at the cost of a remarkable 162% speed degradation after place&route.

Overall, in spite of a good post-synthesis frequency, thanks to the lower radix MINs typically suffer from a more significant performance degradation after place&route due to their more intricate wiring compared to a 2D mesh. Hence, final performance cannot be predicted from early post-synthesis results in a straightforward way without accounting for interconnect-related effects.

When scaling the system to 64 cores, we think that the impact of link delay on maximum achievable speed of MINs cannot be tolerated any more. Advanced link pipelining techniques are required to exploit MIN topology properties while supporting high operating frequencies. From here on, we will assume the use of link pipelining for 64-node topologies (except for the 2D mesh). For this system size, we target a reasonable 750 MHz clock speed for the network, and place pipeline stages across links inducing timing violations at the target clock speed. The number of pipeline stages was computed based on an experiment where two switches were placed close together in a real layout. Their distance was then progressively increased, and for each inter-switch wire length the needed number of pipeline stages was computed to meet the target frequency. Please observe that a pipeline stage is not just a simple retiming stage, but needs to take care of flow control and hence is a flow control stage with 2 slot buffers (see [51] for more details).

In the experiment, we assume that repeater stages are not inferred across retimed wire segments, since their use for such large scale systems is still controversial. Projections indicate that power incurred by repeaters might be as large as tens of Watts, with also an exponential impact on area [7]. For this reason, we conservatively inferred unrepeated AND retimed switch-to-switch links for 64-node networks, which is certainly a worst case for MIN network latency.

After associating a proper number of retiming stages to each wirelength range, we identify on the 64-node floorplans of each topology the links that might be concerned by retiming. For RUFT we found that, as expected by [16], a high number of stages (11) need to be placed in the links connecting the last switch barrier to the network interfaces of destination cores. For the fat-tree, as expected by previous work (e.g., [9]), latency of links close to the root grows. However, due to the better scalability of fat-tree floorplan, the worst-case latency of the fat-tree is lower than RUFT.

**Area**

Since we only performed a coarse grain shrinking of placement blocks on the floorplan without a finer-grain layout area optimization, we only report here total floorplan cell area. See Table 7.4, 5th column. The 2D mesh and the 2-ary 4-tree RUFT exhibit almost the same area, in that they have exactly the same number of I/O ports for a 16-core system, which are the ones that mainly determine switch area. The 2-ary 4-tree FT has a significant 60% area overhead with respect to the 2D mesh, which can be correlated with a 75% increase in the number of switch ports. Interestingly, although featuring the same number of switches, the 2-ary 4-tree RUFT achieves a significant area saving with respect to the FT since it employs lower-radix switches. Obviously, the 4-ary 2-tree S-RUFT exhibits the lowest area footprint with just 8 4x4 switches.

When looking at area projections for 64-core systems (based on post-synthesis area reports of network building blocks), we observe that area of MINs has increased much more than that of the 2D mesh. This is due to a higher increase factor of the switch count in MINs than in 2D meshes to interconnect a larger number of cores. This makes area footprint of 2-ary 4-tree FT hardly affordable and that of 2-ary 4-tree RUFT larger than the 2D mesh.

When we account for area overhead induced by retiming and flow control stages, Table 7.4 shows that an impressive 37% overhead is incurred by RUFT due to the poor scalability of its floorplan. Obviously, S-RUFT is the more penalized topology, and more than 50% of its area with 64 nodes might be occupied by retiming stages. Please consider that this is a worst-case scenario pointing out scalability issues, which can be mitigated by repeater insertion in the links or by custom floorplans for a given system size.

**Wirelength**

A side effect of using lower switch radix in 2-ary 4-tree RUFT is the large saving in total wire length with respect to the 2-ary 4-tree FT, although they are using the same number of switches. 2-ary 4-tree RUFT and 2D mesh consume almost the same amount of wiring resources in spite of the higher routing complexity of RUFT. This further confirms the high efficiency of the customized layout

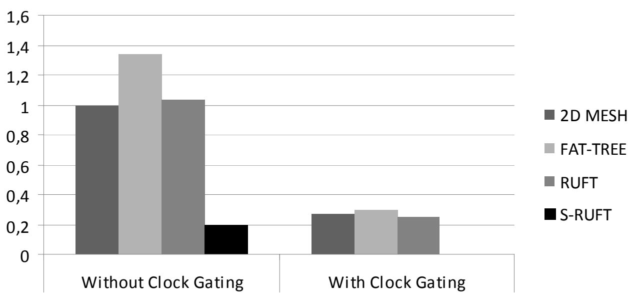


Figure 7.10: **Normalized total power.**

for this topology. Finally, the 4-ary 2-tree S-RUFT has obviously a lower total wiring length, mitigated by the need to interconnect few switch components that are far apart from each other.

**Power**
Power was computed with the same producer-worker-consumer benchmark of [17], which emulates the average communication bandwidth requirements of a partitioned MPEG-4 application. This traffic pattern models bursty accesses to 2 memories serving as I/O interfaces, and all-to-all accesses associated with a cooperative computation process. Processor and memory cores were mapped on each topology. For the 2D mesh, mapping was done in such a way that each processor core is always 1-hop away from its private memory core. In all fat-trees, a processor core and its memory core were connected to the same switch at the lowest stage of the topology. This mapping may turn out to be highly performance sub-optimal for fat-trees (indeed it is a worst-case for RUFT [16]), however we opted for it since it is very intuitive and avoids the use of high-level mapping tools (which is outside the scope of this work).

Power results are illustrated in Fig.7.10. Power of 2D mesh and 2-ary 4-tree RUFT turns out to be the same due to the equivalent number of I/O ports in both networks. The correlation of power with the number of buffering resources is further emphasized by power reports of the 2-ary 4-tree FT, which is the most power consuming topology: 34% more than the 2D mesh. It is worth noting that all aforementioned topologies work at a comparable frequency (the post-place&route one), whereas the 4-ary 2-tree S-RUFT works at approximately half of the speed. If we combine this feature with the far lower number of I/O ports of this topology, we can explain its significantly lower power consumption.

We also went through the physical synthesis process again and enabled the clock gating feature. Results are provided in Fig.7.10 for the most power-consuming topologies. On average, an impressive 75% power saving is materialized by clock gating. This brings power of the most power-hungry topologies close to that of the 4-ary 2-tree S-RUFT. From a timing viewpoint, we observed no major modifications with respect to previous results.

## 7.7 Conclusions

This chapter takes a bottom-up approach to the assessment of $k$-ary $n$-mesh topologies for regular tile-based architectures. Scalability of presented results to 64 tile architectures is considered. The chapter considers a number or real-life issues: physical constraints of nanoscale technologies (post-layout performance, area and power results are given), different physical design techniques, the role of the chip I/O, the communication semantics of middleware for MPSoCs and the role of I/O tile mapping.

### 7.7.1 $k$-ary $n$-meshes

We found that the intricate wiring of multi-dimension topologies or the long wires required by concentrated $k$-ary $n$-meshes can be changed into 2 different kinds of performance overhead by means of proper design techniques:

- operating frequency reduction. This is likely to be the technique of choice for small scale systems. In this case, in spite of a lower number of execution cycles, multi-dimension topologies loose in terms of real execution time due to lower working frequency. Nonetheless, reducing the number of dimensions and connecting more cores to the same switch represent a way to trade performance for power and area;
- increase of link latency. An aggressive utilization of retiming stages allows to sustain operating frequency while increasing network latency. The switch delay associated with its radix poses an upper bound to the effectiveness of this technique. Finally, a significant area and power overhead is to be expected, since retiming stages need to be also flow control stages.

Overall, we found the 2D mesh to still outperform the hypercubes even in 64 tile systems. This is counterintuitive, since hypercubes should scale better in principle. The motivation lies in the mismatch between multi-dimension topologies and the 2D silicon surface (whatever the kind of performance overhead it is changed into) and in the chip I/O bottleneck, which prevents an aggressive performance speed-up at least in clock cycles. Removal of this bottleneck is mandatory to achieve significant performance differentiation between topologies.

These considerations are architecture-specific to some extent. The xpipes-Lite NoC for instance uses frequency-ratioed clock domain crossing at the network interface. So, we found the opportunity to take profit of the low speed of concentrated hypercubes to change the frequency divider at the network interface and to have the cores running faster. We therefore got a topology that achieves the same performance of the 2D mesh while consuming much less hardware resources and power. The availability of more complex synchronization techniques such as asynchronous FIFOs at network interfaces may extend this benefit to other topologies as well.

### 7.7.2 $k$-ary $n$-Tree

Fat-trees are feasible for on-chip networks from a physical design viewpoint. Unfortunately, for small scale systems, they are not able to capitalize on their better performance figure scalability yet. 2D mesh is in contrast a very efficient solution from all viewpoints. Using MINs with less resources (namely S-RUFT) incurs serious physical design issues, mainly associated with long link delay. Moreover, circuit-level power control techniques such as clock gating can significantly cut down on power of the more complex topologies.

As the system size scales up, 2D meshes however suffer from poor performance scalability. Hence, the need for alternative topologies becomes more stringent. $k$-ary $n$-trees can provide that performance scalability, but at an impractical power and area cost. In this scenario, unidirectional MINs (like RUFT and S-RUFT) become attractive for their reduced power and area overhead. Performance-wise, they are effective for latency-sensitive traffic, while they cannot handle bandwidth-intensive traffic as effectively. Unfortunately, their advantages cannot be easily materialized due to a more intricate physical design. No clearly scalable floorplanning strategy is known, and a lot of link retiming and flow control stages are needed to sustain clock speed. The area and power overhead associated with these stages might turn out to be unacceptable. Due to link pipelining, area and power cost of $k$-ary $n$-trees becomes further prohibitive. From a performance viewpoint, the increased link latency of unidirectional MINs causes a lower performance but still better than 2D mesh and also than $k$-ary $n$-trees under latency-sensitive traffic.

# Bibliography

[1] S.Suboh, M.Bakhouya, S.L.Buedo, T.E.Ghazawi;
"Simulation-Based Approach for Evaluating On-Chip Interconnect Architectures",
South. Conf. on Programmable Logic, pp.75-80, 2008.

[2] Vu-Duc Ngo, H.Nam Nguyen, Hae-Wook Choi;
"Analyzing the Performance of Mesh and Fat-Tree Topologies for Network on Chip Design",
L.T.Yang et al. (Eds): EUC 2005, LNCS 3824, pp.300-310, 2005.

[3] S. Kumar et al.,
" A Network on Chip Architecture and Design Methodology",
IEEE Computer Society Annual Symposium on VLSI, April 2002. pp. 105-112.

[4] Rijpkema, E.; Goossens, K.; Radulescu, A.,
"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 (DATE'03), Mar. 2003, pp. 350-355.

[5] A. Jalabert et al.,
"xpipesCompiler: a Tool for Instantiating Application Specific Networks on Chip".
Proc. of Design, Automation and Test in Europe (DATE), pp.884–889, 2004.

[6] Synopsys PrimeTime PX,
http://www.synopsys.com/products/primetimepx/ptpx_ds.html

[7] D. Sylvester and K. Keutzer,
"Getting to the bottom of deep sub-micron II: A global paradigm",
Proc. IEEE Int. Symp. Physical Design, pp.193-200, 1999.

[8] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, G. De Micheli,
"XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips".
Proc. of DATE, pp.1188–1193, 2005.

[9] H.Matsutani, M.Koibuchi, D.F.Hsu, H.Amano,
"Three-Dimensional Layout of On-Chip Tree-Based Networks",
Int. Symp. on Parallel Architectures, Algorithms and Networks, pp.281-288, 2008.

[10] H.Matsutani, Michihiro Koibuchi, Hideharu Amano,
"Performance, Cost and Energy Evaluation of Fat H-Tree: a Cost-Efficient Tree-Based On-Chip Network",
IPDPS 2007, pp.1-10, 2007.

[11] C. Leiserson,
"Fat-trees: Universal Networks for Hardware Efficient supercomputing",
IEEE Transactions on Computer, vol. 34, no. 10, 1985.

[12] Antonio Pullini et al.,
"Bringing NoCs to 65 nm",
IEEE Micro 27(5): pp.75-85 (2007).

[13] A.Adriahantenaina, H.Charlery, A.Greiner, L.Mortiez, C.A.Zeferino,
"SPIN: a Scalable, Packet Switched, On-chip Micro-Network",
DATE'03, Embedded Software Forum, pp.70-73, 2003.

[14] A.Adriahantenaina, A.Greiner;
"Micro-Network for SoC: Implementation of a 32-port SPIN Network",
DATE'03, pp.1128-1129, 2003.

[15] C.Gomez et al.,
"Deterministic versus Adaptive Routing in Fat-Trees",
CAC'07, as part of IPDPS'07, 2007.

[16] C.Gomez et al.;
"Beyond Fat-Tree: Unidirectional Load-Balanced Multistage Interconnection Network",
Computer Architecture Letters, June 2008.

[17] Omitted for blind review.

[18] Omitted for blind review.

[19] F.Petrini, M.Vanneschi,
"k-ary n-trees: High Performance Networks for Massively Parallel Architectures",
Int. Parallel Processing Symposium, 1997, pp.87-93.

[20] P.P.Pande, C.Grecu, M.Jones, A.Ivanov, R.Saleh;
"Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures",
IEEE Trans. on Computers, Vol.54, no. 8, 2005.

[21] Circuits Multi-Projects, Multi-Project Circuits;
http://cmp.imag.fr

[22] P.Guerrier, A.Greiner,
"A Generic Architecture for On-Chip Packet-Switched Interconnections",
DATE'00, pp.250-256, 2000.

[23] P.P.Pande, C.Grecu, A.Ivanov, R.Saleh
"Design of a Switch for Network on Chip Applications",
ISCAS'03, pp.V.217-V.220, Vol.5, 2003.

[24] Open Core Protocol Specification, v.2.1.

[25] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh,
"Structured interconnect architecture: A solution for the non-scalability of bus-based SoCs",
Proceedings of the Great Lakes Symposium on VLSI, pages 192-195, 2004.

[26] S.Stergiou et al.,
"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips",
DAC, pp.559-564, 2005.

[27] STn8811A12 Mobile Multimedia Application Processor,
available online: http://www.st.com

[28] SPEAr Plus600 dual processor cores,
available online: http://www.st.com

[29] TILE64 PROCESSOR FAMILY,
available online:
http://www.tilera.com/pdf/ProBrief_Tile64_Web.pdf

[30] Tensilica LX configurable processor,
Tensilica Inc., http://www.tensilica.com

[31] S.Stergiou et al.,
"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips",
DAC, pp.559-564, 2005.

[32] S. W. Keckler et al.,
"A wire-delay scalable microprocessor architecture for high performance systems",
ISSCC'03, Feb. 2003, pp. 168-169.

[33] Z. Yu et al.,
"An asynchronous array of simple processors for DSP applications",
in ISSCC'06, Feb. 2006, pp. 428-429.

[34] Benini, L., De Micheli G.,
"Networks on Chips: a New SoC Paradigm",
IEEE Computer 35(1),
pp.70-78, 2002.

[35] Dalla Torre, A., Ruggiero, M., Acquaviva, A., Benini, L.,
"MP-Queue: an Efficient Communication Library for Embedded Streaming Multimedia Platform",
IEEE Workshop on Embedded Systems for Real-Time Multimedia,
2007.

[36] Balfour, J., Dally, W.J.,
"Design Tradeoffs for Tiled CMP On-Chip Networks",
ACM International Conference on Supercomputing,
2006.

[37] S. Vangal et al.,
"An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS",
ISSCC 2007,
pp.98-589, 2007.

[38] Coppola, M.; Locatelli, R.; Maruccia, G.; Pieralisi, L.; Scandurra, A.,
"Spidergon: a novel on-chip communication network",
International Symposium on System-on-Chip, 2004,
pp.16-18, 2004.

[39] Bourduas, S.; Zilic, Z.,
"Latency Reduction of Global Traffic in Wormhole-Routed Meshes Using Hierarchical Rings for Global Routing",
IEEE International Conf. on Application-Specific Systems, Architectures and Processors,
pp.302-307, 2007.

[40] Martinez Vallina, Fernando; Jachimiec, Nathan; Saniie, Jafar;
" NOVA interconnect for dynamically reconfigurable NoC systems",
IEEE International Conf. on Electro/Information Technology, 2007,
pp.546-550, 2007.

[41] Gilabert, F., Gomez, M.E., Lopez, P.J.,
"Performance Analysis of Multidimensional Topologies for NoC",
ACACES 2007, poster session with proceedings at the Summer School.

[42] Medardoni, S., Gilabert, F., Bertozzi, D., Gomez, M.E., Lopez, P.J.,
"Towards an Implementation-Aware Transaction-Level Modeling of On-Chip Networks for Fast and Accurate Topology Exploration",
INA Workshop, presented at the HiPEAC08 conference, 2008.

[43] Mirza-Aghatabar, M.; Koohi, S.; Hessabi, S.; Pedram, M.,
"An Empirical Investigation of Mesh and Torus NoC Topologies Under Different Routing Algorithms and Traffic Models",
Euromicro Conference on Digital System Design Architectures, Methods and Tools,
Page(s):19-26, 2007.

[44] Luciano Bononi, Nicola Concer, Miltos Grammatikakis, Marcello Coppola, Riccardo Locatelli,
"NoC Topologies Exploration based on Mapping and Simulation Models",
Euromicro Conference on Digital System Design Architectures,
Page(s):543-546, 2007.

[45] Wang, H., Peh, L.S., Malik, S.,
"A Technology-Aware and Energy Oriented Topology Exploration for On-Chip Networks",
Design Automation and Test in Europe,
Vol.II, pp.1238 - 1243, 2005.

[46] Pavlidis, V.F.; Friedman, E.G.,
"3-D Topologies for Networks-on-Chip",
International SOC Conference,
Page(s):285 - 288, 2006.

[47] I.Hatirnaz, S.Badel, N.Pazos, Y.Leblebici, S.Murali, D.Atienza, G.De Micheli,
"Early Wire Characterization for Predictable Network-on-Chip Global Interconnects",
SLIP'07, pp.57-64, 2007.

[48] Murali, S., De Micheli, G.,
"SUNMAP: a Tool for Automatic Topology Selection and Generation for NoCs",
Proc. of the Design Automation Conference,
2004, pp.914-914.

[49] Soteriou, V., Eisley, N., Wang, H., Li, B., Peh, L.S.,
"Polaris: a System-Level Roadmapping Toolchain for On-Chip Interconnection Networks",
IEEE Trans. on VLSI 15(8),
2007, pp.855-868.

[50] F.Gilabert, S.Medardoni, D.Bertozzi, L.Benini, M.E.Gomez, P.Lopez, J.Duato,
"Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework",
Proc. of the 2nd Int. Symposium on Networks-on-Chip,
to appear in 2008.

[51] A.Pullini et al.,
"Fault tolerance overhead in network-on-chip flow control schemes",
SBCCI, pp.224-229, 2005.

# Chapter 8

# Emerging Issue

## 8.1 Introduction

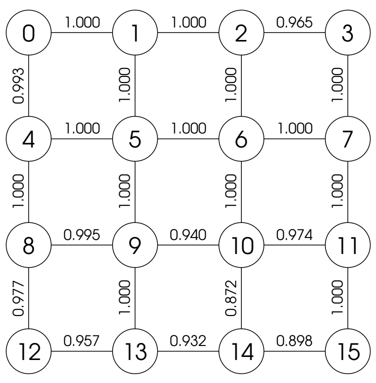
Due to silicon technology scaling effects, boosting clock frequency of monolithic high-performance microprocessors has clearly reached a point of diminishing returns. As an effect, we are experiencing a rapid shift towards massively parallel architectures with tens or even hundreds of cores integrated onto a single chip. Evidence of this trend is unmistakable as practically every commercial manufacturer of high performance processors is currently introducing products based on multi-core architectures[7, 12].

With the increasing number of cores, scalability of communication architectures becomes critical for system performance and traditional bus architectures are rapidly running out of steam [10]. We are moving from core-centric to interconnect-centric design flows and very likely buses will soon be replaced by a dedicated on-chip network providing point-to-point communications by means of dedicated routers [2].

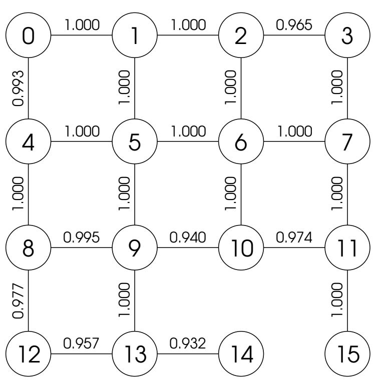
As transistor dimensions continue to shrink, it becomes harder and harder to precisely control the fabrication process. Transistors and wires become faster, but less predictable and, after fabrication, it is becoming more and more difficult to guarantee timing closure, and hence performance, across multiple copies of the chip [4]. Therefore, fault-tolerance is becoming a major concern when designing a multicore chip in general, and a NoC in particular. It is important to emphasize that, in the NoC context, fault-tolerance is not only considered for technical reasons but mainly as the way chip manufacturers keep yield and thus reduce fabrication costs and ensure their economical benefits.

Unfortunately, variability is expected to increase as technology scales down. The ITRS roadmap [20] predicts different percentages of variability in the medium and long term. Current and near term silicon variability is estimated to be around 5% for 45-65$nm$ technologies, while for 22$nm$ variability raises up to 18%.

The rest of the chapter is organized as follows. Section Section 8.2 presents the architectural implications while 8.3 presents the circuit implication of the problem. Finally, Section 8.4 provides the conclusions.



(a) Link frequencies under 5% variability



(b) Links selected for 932 MHz threshold

Figure 8.1: Example on the effect of variability on link frequencies.

## 8.2 Architectural Implications

When such variability affects transistors in the driver or in the buffering stages of NoC communication links, it may happen that one or more wires of the link do not properly switch at the target frequency. The situation is further worsened by the performance variability induced by link dishing effects. These links may turn out to be unusable, as they cannot be simply replaced as in off-chip networks.

As an example of the problem we face in this Section, Figure 8.1.a shows the network for a 16-core chip using a 4x4 mesh topology to interconnect the cores. In this system links have been initially designed to work at 1GHz. However, because of a 5% of variability (for current technologies), we will end up having links working at different frequencies. Figure 8.1.a shows the frequency of each link after applying our variability model (described in Section 8.2.2).

In order to keep the chip working and not decreasing yield, the simplest thing we can do is to lower the operating frequency of the entire network to the lowest link frequency. In this case, our example chip would work at 872 MHz. This, in fact, could be the only feasible option if, for instance, we implement the DOR routing in the network. DOR is designed for 2D meshes and cannot work on a network with one or more missing links. However, as variability increases, reducing chip frequency to the lowest link frequency in the NoC may be unacceptable. Notice that in the example chip, frequency is reduced down to 872 MHz just because one of the links is not able to switch at a faster rate. Actually, just by not considering two links in the network we could increase chip frequency up to 932 MHz. Figure 8.1.b shows the resulting topology for that frequency, where all the cores remain connected but two links are disabled (those not reaching that frequency). However, this would turn an initial regular topology into an irregular one and, therefore, the DOR routing cannot be used because there is a pair of unconnected nodes. If the routing layer does not take this into account, the chip would have to be discarded, reducing yield and increasing manufacturing costs (or would have to be operated at a lower frequency, if the design allows it, decreasing thus performance and benefit).

Fortunately, it is possible to avoid the constraints imposed by the use of the DOR routing algorithm, that is, it is possible to efficiently route packets in incomplete meshes. The solution is the use of topology-agnostic routing algorithms, which can be applied to any topology. Examples of this kind of routing algorithms are the up*/down* [27] and the segment-based routing algorithms [26]. Unfortunately, the implementation of these algorithms is much more complex than the DOR routing. The general way to implement such routing algorithms is by using routing tables either at end nodes (source-based routing, used by the Polaris chip) or at switches (distributed-based routing). In either case the provided implementation does not scale (in terms of power and area) since it requires as many tables as endnodes (or switches) with as many entries as endnodes, therefore exhibiting memory requirements with a quadratic increase.

Recently two solutions to overcome this problem have been proposed: RbR [29] and LBDR [28]. In [29] authors propose compressing the routing tables by destinations. At every switch all the destinations using the same set of output ports at the switch are grouped in one single entry thus achieving large area savings. The set of destinations is referred to as a *region* and RbR relies on the use of a bounded number of regions at every switch. Indeed, a logic block is used per each region defined at each switch. On the other hand, in [28] a further compression of the routing logic downto a small set of bits is proposed. In particular, routing tables are removed by a set of 12 bits per switch. These bits indicate connectivity of the switch and routing options. A small logic block has been proposed for LBDR.

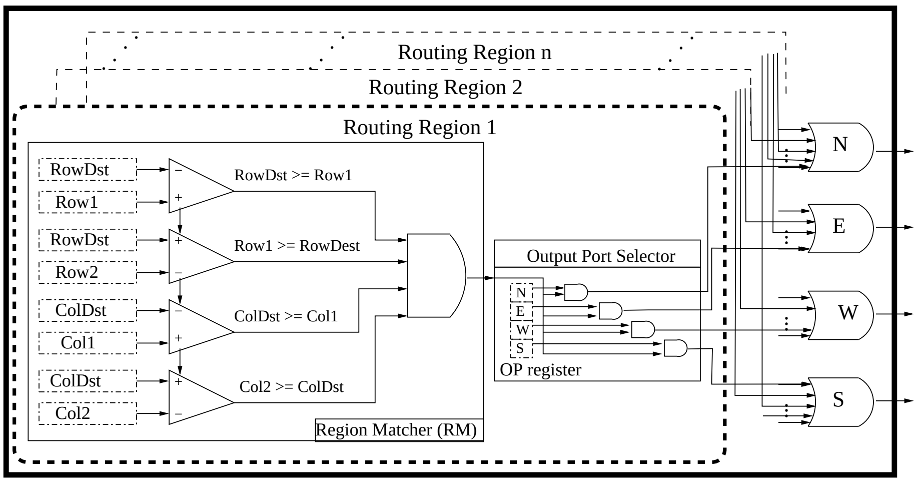
However, the efficient implementations of topology-agnostic routing algorithms mentioned above do not support all the topologies, as they are a simplification of the general table-based implementation and they could endup requiring many resources (regions in the case of RbR). Therefore, some combinations of topologies, routing algorithms and implementations will simply fail for a given chip, while other combinations will succeed. Moreover, with a given routing implementation we may succeed to operate a given chip at a higher link frequency. This is a challenging task since yield, production costs, and benefits are affected.

We will explore the effect of different routing algorithms and routing implementations on yield and link frequency, taking into account variability issues for chips with manufacturing defects. We will also analyze the impact of yield and link frequency on the manufacturer benefit. The main goal is to determine which is the best option for routing purposes when considering yield, performance (frequency), cost and benefit.

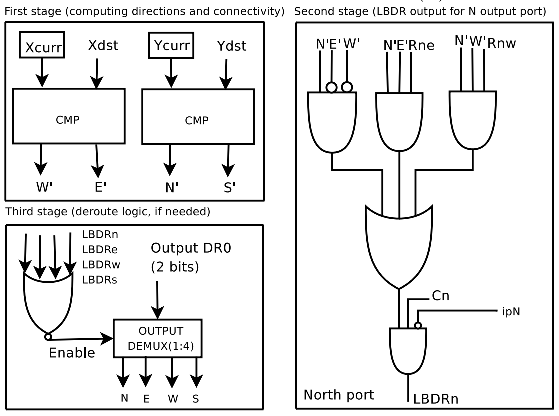
It has to be noted that variability on transistors is used as a source of irregular topologies. Therefore, we do not consider at this step topologies derived from other sources of manufacturing errors. Also, we assume that variability compensation techniques are implemented in the switches, so that switch performance variability cannot be a further source of topology irregularity.

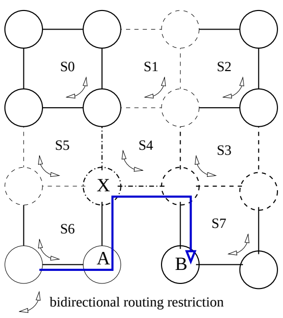
### 8.2.1 Routing Mechanisms and Algorithms

For this study we have selected the SR [26] routing algorithm and two routing implementations: RbR [29] and LBDR [28]. We chose them as they provide flexible computation of routing paths (SR) and efficient implementations of routing mechanisms (RbR and LBDR) without the need for routing tables. Also, they have been presented in the last three years and have been focused for on-chip networks. We have added also new functionalities to these routing



(a) RbR





(c) SR algorithm and deroute example

(b) *LBDR* and $LBDR_{dr}$

Figure 8.2: SR, RbR, and LBDR routing algorithms and mechanisms.

mechanisms (namely LBDR) in order to increase their coverage of non-regular topologies. The main constraint for these additions is that they must have small area overheads while increasing the coverage in an effective manner.

SR is a topology-agnostic routing algorithm that does not require the use of virtual channels and guarantees deadlock freedom and full connectivity among nodes. SR works by partitioning a topology into subnets, and subnets into segments. As segments are independent, placements of routing restrictions are independent from segment to segment. This locality independence property adds a new dimension to the placement of routing restrictions, and allows to disentangle from the limitations found in other routing algorithms. An example set of routing restrictions computed by SR is observed in Figure 8.2.c. The search for segments is initiated on the top left most switch and in a zigzag manner down to the bottom. Figure draws each node and link with different patterns showing that each of them belong to a different segment (from $S_0$ to $S_7$). A bidirectional routing restriction have been placed on each segment (8 in total) so traffic can be spread evenly across the topology. A routing restriction (see Figure 8.2.c) is set by two adjacent links and ensures no packet can cross them. Both routing mechanisms (RbR and LBDR) will make use of this routing algorithm.

Region-based Routing (RbR) framework allows to route messages by using simple and efficient blocks of logic referred to as regions. A different set of regions is implemented in logic at every input port of a given switch. Figure 8.2.a shows hardware required per input port. Each region is a square box of destinations that is identified by the possible output ports (OP register), the top left most switch (Row1 and Col1 registers) and the bottom right most switch (Row2 and Col2 registers) of the region. When a message header arrives to the input port, it inspects all regions in parallel in order to find out which are the possible output ports it can take. Regions are computed and identified by the RbR mechanism based on the routing restrictions and the network topology. They are used to setup hardware registers located at every input port of every switch. These registers must be programmed before routing any packet at network boot time. A nice feature of RbR is that it allows the implementation of topology-agnostic routing algorithms providing appropriate support for routing under the presence of link and node failures.

LBDR implementation is based on a small logic and uses two sets of bits on every switch: the $R_{xy}$ routing bits and the $C_x$ connectivity bits. As in RbR, all bits are set before normal operation and are computed based on the routing algorithm used and the current topology. A routing bit, $R_{xy}$, at a given switch indicates if a packet is allowed (by the applied routing algorithm) to leave the switch through output port $x$ and at the next switch to turn to direction $y$. Connectivity bits, $C_x$, are used to define the current topology being used. Each output port has a connectivity bit that indicates if a neighbor switch is attached through the output port. With all these bits a small logic block is set to effectively route packets in a deadlock-free and connected manner (Figure 8.2.b).

As commented by the authors, LBDR does not allow the use of non-minimal paths. And this fact limits its applicability to highly irregular topologies, thus potentially affecting yield. As an effort to reduce this shortcoming, we extended LBDR with an additional feature. The addition is referred to as $LBDR_{dr}$ and will provide non-minimal paths at strategic switches. To do so, we add a small logic and two bits per switch to LBDR. The addition is shown in Figure 8.2.b. The logic (NOR gate) is enabled when LBDR is not able to select a proper output port for a packet. Then, a multiplexer is enabled to provide a deroute output port for the packet. Two bits are used per switch to configure the multiplexer accordingly to the routing algorithm being used. Notice that when computing the routing bits and the deroute bits we need to guarantee that no routing restrictions are crossed by packets.

Figure 8.2.c shows an example where the deroute is used. The figure shows a case not supported by LBDR (the non-minimal path shown). At switch A the packet should be sent through the east output port, however the link is not operational. With $LBDR_{dr}$, now the provided example is supported. At switch A, the packet needs to be derouted. In the example we select the north port as the deroute port for the packet. The packet will leave the switch through the north port and will reach switch X. At that switch the packet is evaluated. In particular, the direction is computed. In this case direction is south-east. With LBDR the packet would select either east or south. Notice that in this case the

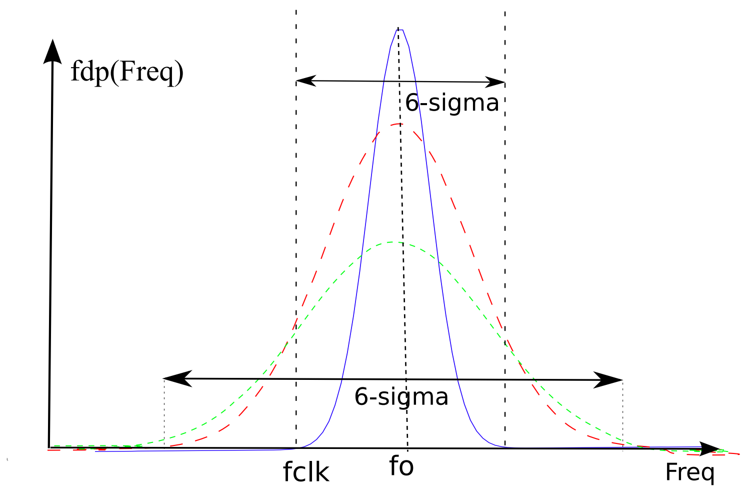


Figure 8.3: Several Gaussian density functions showing different process variability

packet should not take south since it was coming through the south port. To prevent this we filter $U$ turns with a small logic. From that point the packet will be routed with the basic LBDR mechanism. There is no need for new deroute ports.

### 8.2.2 Modelling within-die variability

Variability is caused by deviations introduced in the fabrication process and by intrinsic effects like atomic-level differences due to small variations in dopant levels. Process variation affects the delay of a signal propagating between two points in the die, and therefore, NoC links are also affected by variability.

When designing a NoC, the maximum theoretical operating frequency of the links ($f_{\text{max}}$), obtained through the worst-case delay analysis, follows a Gaussian probability density function [19] that depends on the parameter variation of the manufacturing process. Thus, the actual operating frequency targeted at design time ($f_{\text{clk}}$) must be lower than the mean of this density function ($f_{\text{o}}$), since not all wires would be able to reach $f_{\text{o}}$ as a consequence of process variations. Actually, $f_{\text{clk}}$ must be low enough to guarantee that all links in the network are able to switch at that frequency. This behavior is shown in Figure 8.2.2, where several Gaussian density functions corresponding to different process variations are shown. The narrowest curve is associated to a fabrication process with low frequency variation, $\sigma$ ($\sigma$ is the standard deviation). In order to minimize the probability of having faulty links, the operation frequency, $f_{\text{clk}}$, is chosen according to a $6\sigma$ criterion, i.e. $f_{\text{clk}} = f_{\text{o}} - 3\sigma$. However, as process variability is becoming higher, the shape of the probability density function of the maximum operating frequency becomes wider too. Consequently, as variability increases, a lower value of $f_{\text{clk}}$ must be used in order to keep all the links within its operating range, noticeably reducing frequency. This frequency reduction keeps yield but considerably reduces performance.

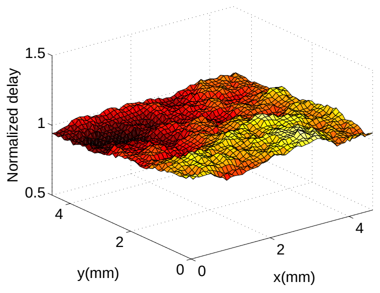
We consider two different values for link-delay variability (or more precisely, for link-segment delay variability), according to [16] [20]. The first one is set to 5% ($\sigma = 0.05$), representing the current technology aiming a 65$nm$ manufacturing process. The second one is 18% ($\sigma = 0.18$), corresponding to the estimated variability present in future 22$nm$ processes [21]. These values will provide us a good estimation on how an increasing variability may affect link delay and therefore design decisions.

Process variability also presents spatial dependencies, that can be modelled by using Gaussian Random Fields (GRF) [17] [22]. When using GRF, if it is assumed that the field is stationary and isotropic, then the variance ($\sigma_i^2$) of the random field $D(x, y)$, representing the delay in the (x,y) die position, depends only on the euclidean distance between two given locations. Then, the delay distribution ($D$) only depends on a correlation function. We are going to use the exponential model for characterizing the correlation function. In this model, $\lambda$ modulates the strength of the correlation in such a way that if $\lambda$ tends to infinite, then the delay at a given point has no neighborhood dependencies.

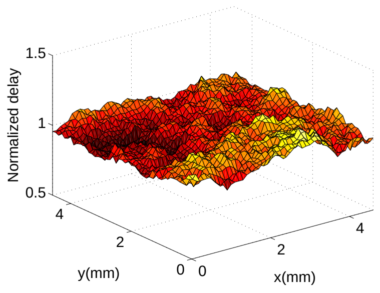
We have modelled spatial dependencies of variability by generating synthetic within-die variations following the methodology proposed in [17]. Moreover, we have considered two different families of scenarios. The first one models a CMP chip consisting of 16 cores arranged as a 4x4 2D-mesh showing variability of 5%. This chip family is intended to represent current or near term CMPs. The second family models a 64-core chip using an 8x8 2D-mesh. This family would represent future CMPs, and therefore presents a 18% variability. Within each family, the strength of the spatial correlation, $\lambda$, has been set to values 0.4 and 1.2, representing a high and low correlation, respectively. These values are representative of the typical correlation induced by fabrication processes [17] [22]. In summary (see Table 8.1), we are taking into account two chip sizes, each of them using two correlation strengths but showing different variability values. Finally, for each parameter combination we have analyzed 50 chips.

In order to accurately map spatial variability data to the physical layout of the chips, a real 65$nm$ implementation NoC layout [18] has been considered, where all cores are identical, and their size is 1$mm^2$. Additionally, the gap between cores is 0.2$mm$. Therefore, links connecting NoC switches are 1.2$mm$ long. According to these sizes, the 4x4 chips require a 4.7$mm$x4.7$mm$ die. Then, the methodology proposed in [17] has been applied by transforming the 4x4 die into a 47x47 grid. By doing so, every point in the grid represents the delay of a link segment whose driving repeater is located at that die location. Link segment length corresponds to the interval for repeater insertion and is derived by observing in [30] the behaviour of a commercial link physical synthesis tool for a 65nm technology. With respect to the 8x8 chips, we do not have implementation data for them (with a 22$nm$ technology). Therefore, we have assumed that the ratio between core side and intercore space is kept, and thus 8x8 dies will transform into 95x95 grids.

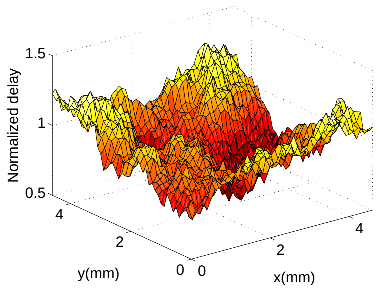
Figure 8.4 shows some examples of the data provided by the methodology described above when applied to the 16-core chips. The nominal delay of links considered is 1$ns$. It can be seen that as variability increases, differences in the delay among different parts of the chip also increase. This can be easily seen in Figures 8.4(a) and 8.4(c). The only difference between them is the increased process variability. Also, when Figures 8.4(a) and 8.4(b) are compared, it is clearly seen that fabrication processes showing large correlation (small values



(a) $\sigma = 0.05$ $\lambda = 0.4$



(b) $\sigma = 0.05$ $\lambda = 1.2$



(c) $\sigma = 0.18$ $\lambda = 0.4$

Figure 8.4: Spatial correlation of delay in a 4.7$mm$ x 4.7$mm$ die

of $\lambda$) produce much more homogeneous layouts from the point of view of delay. Figure 8.4(b) shows that a small correlation value causes that points closely located to each other may exhibit larger differences in delay.

The last step is obtaining link-delay data from the computed grids. To do so, we have taken into account that 1.2$mm$ links require two segments in a 65$nm$ technology when working at 1GHz. Therefore, we have added the values of the two grid points that map to the two segments of each link in the network. Therefore, we catch the possible timing compensation produced as a consequence of link length.

### 8.2.3 Evaluation

In this Section we provide the results in terms of coverage, performance, and benefit. However, we first introduce the evaluation methodology and define the set of metrics we have used.

#### Evaluation Methodology

Figure 8.5 shows the evaluation procedure we will apply to each of the chips shown in Table 8.1. For each chip, and as a result of analyzing link delay variability, we will generate different topologies by varying the operating frequency (step a). The highest considered frequency will be the one that first provides physical connectivity between all end nodes (932 MHz in Figure 8.1), while the lowest one will correspond to that of the slowest link. That frequency will pro-

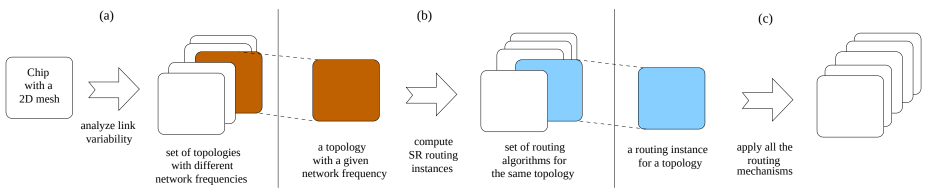

(a)
(b)
(c)
Chip with a 2D mesh
analyze link variability
set of topologies with different network frequencies
a topology with a given network frequency
compute SR routing instances
set of routing algorithms for the same topology
a routing instance for a topology
apply all the routing mechanisms

Figure 8.5: Steps followed to compute the fault tolerance of LBDR and RbR mechanisms.

vide the maximum connectivity in the NoC, that is, a 2D mesh network (872 MHz in Figure 8.1). The first topology will set the maximum frequency the network can work and the last topology will set the minimum frequency. The number of topologies for each chip will vary due to variability issues.

| # chips | $\sigma$ | $\lambda$ | topology | SR instances |
|---|---|---|---|---|
| 50 | 0.05 | 0.4 | $4 \times 4$ mesh | 16 |
| 50 | 0.05 | 1.2 | $4 \times 4$ mesh | 16 |
| 50 | 0.18 | 0.4 | $8 \times 8$ mesh | 8 |
| 50 | 0.18 | 1.2 | $8 \times 8$ mesh | 8 |

Table 8.1: Parameters used in the evaluation.

For each of the topologies obtained in the previous step, we will compute the SR routing algorithm prioritizing performance by searching segments from top to bottom and in a zig-zag manner. Additionally, we will compute different SR instances by starting from a different node. Thus, we will obtain 16 instances for all the topologies of the $4 \times 4$ chips and 8 instances for all the topologies of the $8 \times 8$ chips[1] (step b).

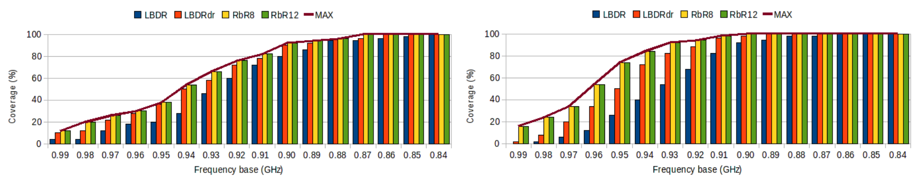
For every computed routing instance (belonging to a topology of a given chip and operating frequency) we will test if every routing implementation works or not (step c). A routing implementation will work if it is able to route packets from every source to every destination. The routing implementations we will test are LBDR, $LBDR_{dr}$ (LBDR with deroutes), $RbR_{8r}$ (RbR with eight regions per input port), and $RbR_{12r}$ (RbR with 12 regions per input port).

If a routing implementation works for at least one of the routing instances for a topology with a given operating frequency, then, it is said that the routing implementation covers the current chip at that frequency. We will rank all the chips with the maximum supported link frequency for every routing implementation.

### Evaluation Metrics

The main goal is to analyze the fault tolerance of the LBDR and RbR routing implementations with respect to variability issues. To do so, we define a new metric that represents the usability of each mechanism. The metric is named *coverage* and is defined as the percentage of chips that are usable for a given frequency and routing implementation. For instance, if we show a coverage of 80% for LBDR at 900MHz, this means that in 80 chips out of 100 chips working

[1]8x8 topologies allow for 64 instances, but for the sake of reducing analyzed data, we only compute 8 of them by starting in the diagonal nodes.



(a) high spatial correlation, $\lambda = 0.4$ (b) low spatial correlation, $\lambda = 1.2$

Figure 8.6: Coverage results for chips using a $4 \times 4$ mesh. $\sigma = 0.05$.

at 900MHz, LBDR successfully allows communication between all the cores in the chip.

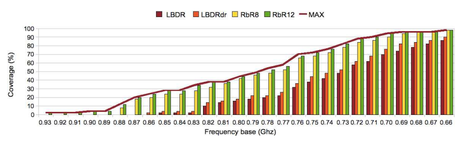
In addition to the coverage metric we will analyze the effectiveness of each implementation to provide coverage. In particular, we model each implementation in the xpipeslite switch architecture and derive area and delay results for each routing implementation. With the *coverage/area* metric we will assess the effectiveness of each implementation in providing coverage. As pointed out in [23], additions in area in a multicore chip must be done if the benefits are worth the addition. A *kill* rule is defined where 10% of additional area is worth being used if at least a 10% increase in performance is achieved. With the coverage/area metric we will provide a kill rule for the coverage.

As we analyze coverage at different link frequencies we endup with different distributions of chips along the frequency spectrum for each implementation. This has a strong correlation with the benefits the chip manufacturer may achieve. Indeed, chips working at higher frequencies will be targeted for high-end applications and thus these chips will be more expensive. In order to analyze this issue we define a new metric, referred to as *average benefit.* This metric will indicate the differences in benefits for each chip when using different routing implementations.

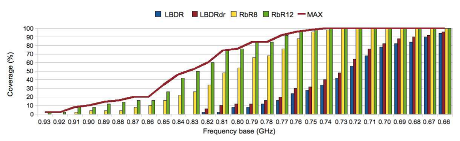
### Coverage Results

In this first evaluation we analyze the coverage that can be achieved for each routing implementation. To do so, we classify the chips according to the frequency they can work at. Figure 8.6 shows the distribution along the link frequency of all the $4 \times 4$ chips ($\sigma = 0.05$) analyzed for $\lambda = 0.4$ and $\lambda = 1.2$, respectively. The figure includes results for $LBDR$, $LBDR_{dr}$, $RbR_{8r}$, $RbR_{12r}$, and $MAX$ (routing tables). The $MAX$ curve represents the percentage of chips that have a connected network (with respect to the total amount of chips). The same frequency $f_0$ is assumed for all experiments, selected in the high-performance range of the network (e.g., 1GHz) where the reliability-performance trade-off appears more clearly.

As a first observation (see $MAX$ curve) we can see how the spatial correlation of the variability affects the connectivity of the network. With a low spatial correlation ($\lambda = 1.2$) more chips can be operated at a higher frequency (the topology gets connected at higher frequencies). The reason for this is that for low correlation values, the frequencies of the links of a given router spread out to a larger frequency range, and therefore, when a frequency threshold is applied, that router has more chances to get any of its links above the threshold. On the contrary, for high correlations, all the links of the router will work at



(a) high spatial correlation, $\lambda = 0.4$



(b) low spatial correlation, $\lambda = 1.2$

Figure 8.7: Coverage results for chips using a $8 \times 8$ mesh. $\sigma = 0.18$.

similar frequencies, and thus, for a given frequency threshold, the probability of all of them having a lower (or higher) frequency than the threshold is higher. If they are below the threshold, the router will remain unconnected.

It is interesting to see how link frequency is affected by the variability effect on transistors. 50% of the chips get connected topologies at $0.94 f_0$ for $\lambda = 0.4$ and at $0.96 f_0$ for $\lambda = 1.2$. Less than 20% of the chips have a connected topology at the designed frequency $f_0$ and all chips get a connected topology at low frequencies: $0.87 f_0$ for $\lambda = 0.4$ and $0.91 f_0$ for $\lambda = 1.2$.

As we focus on the coverage we can see that overall, RbR achieves the highest coverage in both scenarios (different $\lambda$ values), as it covers all the connected topologies in all the frequency ranges (bar plots of $RbR_8$ and $RbR_{12}$ equal the $MAX$ curve). In this case, only eight regions are enough to provide maximum coverage. This is an interesting observation since an 8-region implementation of RbR will be much simpler and efficient than 12- or 16-region implementations (we will show area results in the following subsection).

On the other hand, the LBDR mechanism is somewhat behind RbR in terms of coverage. The basic $LBDR$ mechanism (without deroutes) achieves half the $RbR_8$ coverage for chips operating in the range of $0.94 f_0$ to $f_0$ for $\lambda = 0.4$. However, for a lower spatial correlation ($\lambda = 1.2$, Figure 8.6.b) differences between $RbR_8$ and $LBDR$ are more significant. Notice that, 80% of chips can be operated at $0.94 f_0$ with $RbR_8$ whereas only 40% of them can be used with $LBDR$.

If we add the deroute option to LBDR, differences are significantly reduced. For a high spatial correlation ($\lambda = 0.4$), $LBDR_{dr}$ almost solves the coverage problem of LBDR. As an example, adding deroutes helps to increase coverage from 60% upto 72% for chips working at $0.92 f_0$. Similar results are achieved also in the case of lower spatial correlation ($\lambda = 1.2$), however differences in coverage still remain when compared to $RbR_8$.

As a conclusion for the current $4 \times 4$ chips with $\sigma = 0.05$, we can conclude that the best option for coverage purposes is RbR with only 8 regions and LBDR achieves a decent although minor result for the coverage. RbR achieves 100% coverage for all connected chips in all the frequency range. Instead, LBDR achieves, in most cases, 50% of coverage for the connected chips.

Now, let us turn our attention to future technologies with larger variability margins. Figure 8.10 shows, for different spatial correlations, the coverage of each implementation for chips with an $8 \times 8$ mesh with $\sigma = 0.18$. Frequencies are again normalized to $f_0$. Results are very interesting here since the mechanisms face much more irregular topologies. The first observation is that the span of frequencies is much larger (due to a larger variability parameter). For instance, all the chips get a connected topology at $0.65 f_0$ for $\lambda = 0.4$ and at $0.74 f_0$ for $\lambda = 1.2$.

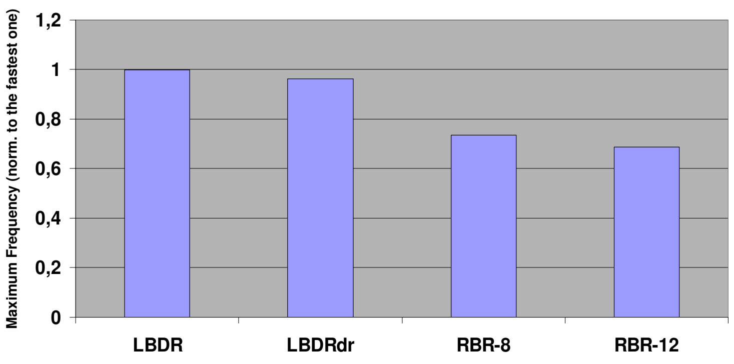


Figure 8.8: Maximum speed achievable by the switch with different routing mechanisms inside. Post-layout results.

Also, differences between LBDR and RbR are much more noticeable. In particular, LBDR looses all the coverage it achieved for current 4x4 chip designs. Neither $LBDR$ nor $LBDR_{dr}$ are able to achieve a reasonable coverage percentage at a high link frequency (e.g. 50% coverage is achieved at $0.73 f_0$ for both $\lambda = 0.4$ and $\lambda = 1.2$). On the other hand, for RbR we observe that maximum coverage of chips with connected topologies is achieved in all the frequency ranges for $\lambda = 0.4$ and in most cases for $\lambda = 1.2$. In some cases (for $\lambda = 1.2$) RbR requires 12 regions to achieve maximum coverage for chips with connected topologies. Up to 30% of coverage differences can be achieved between $RbR_8$ and $RbR_{12}$, and more than 50% of coverage difference in a wide frequency spectrum between LBDR and RbR.

To sum up, we see that a higher variability value increases the span of frequencies of the chip (many irregularities) and decreases the effectiveness of the routing implementations. Coverage is guaranteed in this case with RbR but requiring 12 regions instead of 8. LBDR looses ground in terms of coverage. In the next section we will analyze each mechanism in terms of performance and area.

### Performance and Cost Results

In the previous section we have analyzed the coverage each routing implementation provides for different values of $\sigma$ and $\lambda$. The analysis has been performed for the same design frequency $f_0$. In this section we analyze the maximum operating frequency reachable by each switch with the different routing implementations and the area overhead for their logic and registers. The intention behind this is to characterize the reliability-performance/area trade-off, and to ultimately assess whether the implementation complexity of a routing mechanism largely exceeds its coverage improvements or whether these latter are worth the cost.

The xpipesLite design platform is used to refine RTL descriptions into actual switch layouts. An STMicroelectronics 65nm low-power technology library was used, and commercial backend synthesis tools.

Figure 8.8 shows the maximum achievable switch frequency when using each possible routing mechanism. Results are normalized to the fastest solution (LBDR). Logic depth for each scheme is important, since it resides on the switch

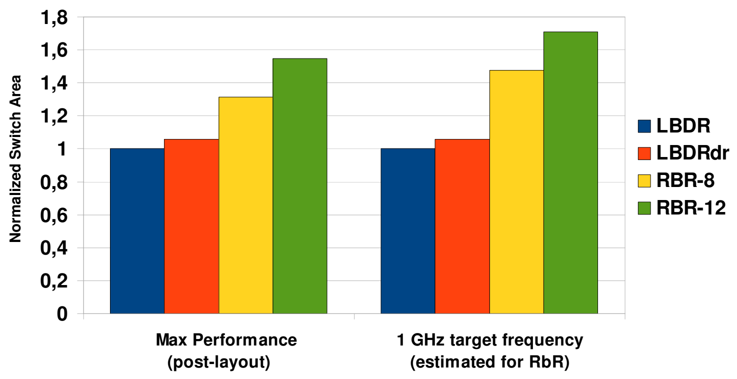


Figure 8.9: Switch area with different routing mechanisms inside. Area at 1 GHz is estimated accounting for the switch pipelining overhead (best and typical case area ratios).

critical path. Clearly, $LBDR_{dr}$ incurs only a minor degradation of the maximum speed (which is slightly more than 1 GHz for baseline LBDR). In contrast, RbR suffers from about 30% frequency reduction, with a further degradation when moving from $RbR_8$ to $RbR_{12}$. The main cause for this large critical path lies in the larger number of cascaded logic stages (local-port matching, region matching, output port selection) and in the use of high-fanout nets at switch inputs. In fact, the destination coordinates of the packet have to be checked in parallel with the upper-left and lower-right coordinates of each region. In practice, after place-and-route, a large delay arises to drive these high-fanout nets. LBDR has a much more layout aware implementation that reduces to a few more gates and registers for each switch input port.

The low performance of RbR can be hidden by pipelining switch operation. To reach 1GHz switch frequency the single cycle switch architecture can be retained only for LBDR solutions. RbR requires 2 cycle switches to keep up with that frequency.

As regards area, we can see in Figure 8.9 the results. The left-hand plot shows post-layout area at maximum performance, which is the one just found. In all switches, input and output buffers drain most of the area. Therefore, the larger routing logic used by RbR with respect to LBDR is not fully reflected in the switch area increment, which makes RbR overhead significant but not unacceptable.

On the right-hand side of Figure 8.9 an area estimation is reported for the case where all switches have to be operated at 1 GHz. Therefore, area overhead of RbR now also includes the estimated cost for pipeline registers and additional flow control stages. In practice, the left-hand plot shows area for single cycle routers at maximum performance, while the right-hand plot shows area for the two schemes to run at the same frequency.

Therefore, we endup having two mechanisms with opposite benefits, LBDR having low coverage but small area requirements and RbR having excellent coverage but large area requirements. In order to quantify this trade-off, we define the coverage/area metric, which expresses how effectively network area is used with respect to the reliability objective. Coverage/area is measured by

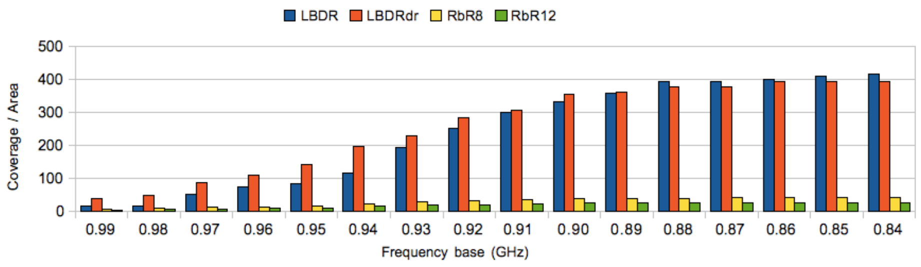


Figure 8.10: Coverage/area results for chips using a $4 \times 4$ mesh. $\sigma = 5\%$, $\lambda = 0.4$.

| Frequency | Chips | | | | Benefit |
|---|---|---|---|---|---|
| range | $LBDR$ | $LBDR_{dr}$ | $RbR_8$ | $RbR_{12}$ | weight |
| $0.91f_0$ - $f_0$ | 0% | 0% | 2% | 8% | 3.47 |
| $0.82f_0$ - $0.9f_0$ | 2% | 6% | 32% | 52% | 1.93 |
| $0.73f_0$ - $0.81f_0$ | 40% | 42% | 66% | 40% | 1 |

Table 8.2: Frequency bins, percentage of chips, and relative benefit.

dividing the coverage percentage by the percentage of chip area devoted to the routing mechanism (area of the mechanism multiplied by the number of routers). Results are reported in Figure 8.10. A value of one in coverage/area means that the same percentage of area increment leads to the same percentage of coverage increase. As can be seen, now when considering real implementation costs we get opposite coverage results. LBDR achieves a high coverage/area value since its implementation costs are very modest. Values higher than 300 are obtained (300 times more coverage than area required). However, RbR achieves a low coverage/area value because although it achieved an overall high coverage value, its area requirements are too high.

As we have seen in the previous section, different chips can be operated at different link frequencies, thus enabling a higher throughput for the entire chip. Chip manufacturers tipically classify chips at different operating frequencies and they are sold at different prices, thus achieving different benefits. As the use of different routing implementations can lead to different amounts of chips at different frequency bins, they (the routing implementations) will have a high impact on the potential benefits. To analyze this, Table 8.2 shows different frequency bins and a relative number of the benefit for chips at different bins. These numbers (frequency ranges and benefits) have been derived from the Intel web page, in particular from the Intel Core i7 processor family (i7-920, i7-940, and i7-965 models). Based on these numbers, Figure 8.11 shows, for $\sigma = 0.18$ and $\lambda = 1.2$ the average benefit of each chip when using a different routing mechanism. As can be seen, RbR allows much higher benefits than LBDR due to its higher coverage. This is due, mainly, to the fact that no chips can be operated at the highest frequency range with LBDR. Average benefit per chip can be cuadrupled when using RbR.

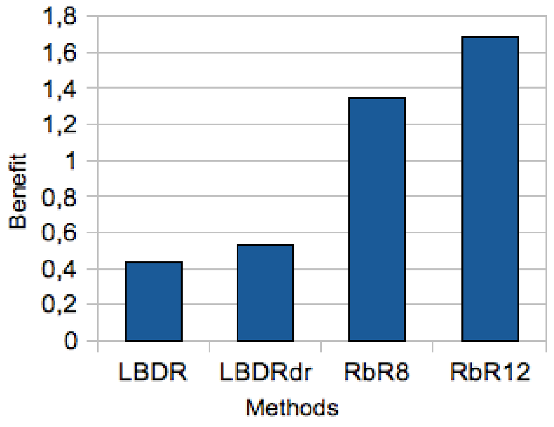
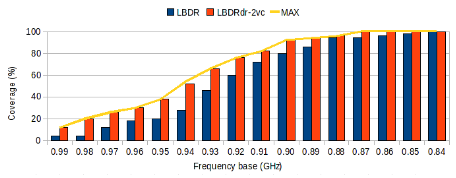
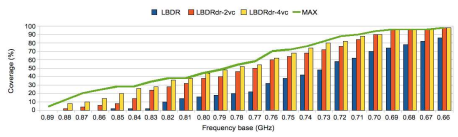


Figure 8.11: Average chip benefits. $8 \times 8$, $\sigma = 0.18$, $\lambda = 1.2$.



(a) $4 \times 4$ mesh, $\sigma = 5\%$, $\lambda = 0.4$



(b) $8 \times 8$ mesh, $\sigma = 18\%$, $\lambda = 0.4$

Figure 8.12: Coverage results for $LBDR_{vc}$.

### Virtual Channels on the Run

From the previous section we can derive conflicting interests when deciding which is the best routing implementations in terms of coverage, performance, and area. LBDR performs well (high frequency design, no pipeline required) and is area efficient, however has a low coverage (even with deroutes). On the other hand, RbR has a very good coverage but low performance and is inneficient in terms of area. Additionally, increasing RbR performance is difficult. It seems more interesting to improve LBDR coverage. In this section we analyze a further improvement over LBDR trying to increase significantly its coverage while keeping its area requirements low (and its high frequency design).

The idea behind the new improvement is applicable to routers using virtual channels. It is expected, as technology advances, that the use of virtual channels will be common in NoCs. LBDR is extended to allow a virtual channel transition once a routing restriction is used. By doing this, and by transitioning virtual channels in a sequential and ordered way we can ensure deadlock-freedom.

Figure 8.12 shows the coverage results for LBDR when using no virtual channels ($LBDR$) or when using two ($LBDR_{dr-2vc}$) or four ($LBDR_{dr-4vc}$) virtual channels. As can be seen, using virtual channels helps to maximize coverage with LBDR. Only two virtual channels are required with current $4 \times 4$ chips with $\sigma = 0.05$ whereas four virtual channels are required for $8 \times 8$ chips with $\sigma = 0.18$. These results, however, need to be further explored in terms of coverage/area and benefit. This is left for future work. However, results provide good indications about how coverage can be maximized with an area-

and performance-efficient solution.

## 8.3 Circuit Implications

With traditional worst-case design techniques, the risk is to be too conservative or to discard too many chips, thus resulting in very low levels of yield.

The emerging issue now is how to design differently than worst-case [3]. Self-calibrating designs have been introduced as an alternative to worst-case characterization of silicon [13]. Instead of relying on over-conservative worst-case assumptions, self-calibrating circuits tune their operating parameters to in-situ actual conditions. A run-time controller receives feedback from a checker that monitors correct operation of the circuit. When needed, circuit reliability can be restored at some power or performance cost, depending on actual silicon capabilities and noise conditions.

A milestone work exploring the use of self-calibrating links in the context of on-chip networks is [14]. It proposes dynamic voltage-swing scaling as a way to dynamically adapt to environmental variations, design uncertainties, manufacturing parameter deviations or communication bandwidth requirements.

While their transmission scheme adapts to variations of *link* physical parameters by discovering the real delay-voltage characteristic of the technology without any assumptions on it, there is no connection between link operating conditions and parameter variations affecting upstream or downstream functional units. In practice, self-calibrating links cannot prevent an upstream switch from violating its timing requirements due to delay variations.

In contrast, our work pioneers the use of digital self-calibrating links to compensate process variation induced delays in the upstream logic unit. We do not view the communication channel in isolation, but its operating parameters can be adapted to absorb and compensate the effects of process variations in connected modules. We also select link voltage swing as our compensation knob. We envision a design methodology where process variations of link upstream modules are measured during the post-fabrication test phase, and clock skew of such modules tuned accordingly. Then we make up for the reduced propagation delay margins on the link through voltage swing adaptation.

We show that the energy overhead induced by the increased swing is offset by deploying low-swing signaling on the links under nominal conditions. This creates some margin for power savings which is the budget for process variation compensation during post-silicon testing and therefore for yield enhancement.

Our technique can also be used during the normal life of the circuit in order to address wear-out problems. Periodically, the chip could be re-tested and new problems caused by delay variations might arise. Our NoC links could be reconfigured in order to cope with the new conditions, thus resulting in the possibility to extend the lifetime of the chip by recharacterizing the links and their power consumption.

The rest of the chapter is organized as follows. Section 8.3.1 presents our guiding principles. Self-calibrating link implementation is then illustrated in Section 8.3.2. Finally, the experimental results in Section 8.3.3 explore the reliability - power trade-off. Conclusions are drawn in Section 8.4.

### 8.3.1 Self-correcting flip-flop

A recently proposed technique to deal with process variations in pipelined designs consists of clock cycle time stealing, where the time slack of faster stages in the pipeline is transferred to the slower ones by skewing the clock arrival times to latching elements.

In a different context, a similar scenario occurs also in on-chip networks. Whenever the switch and the link can be viewed as two consecutive pipeline stages, a critical path delay degradation in the switch could be compensated through slack stealing from the link. However, there is a big difference between the NoC and the microarchitecture scenario. In this latter case, if not enough slack is available in other pipeline stages, donor pipeline stages should be inserted, which are empty stages added in the pipeline to donate slack to the slow stages. A larger flexibility does exist for NoC links, since such slack could be provided without increasing link latency (e.g., by means of pipelined links). Link operating conditions provide the needed tuning knob to adapt the link propagation delay to the skew applied to an upstream logic stage. In particular, electrical schemes to reduce the voltage swing of the interconnect are known and well studied [15]. Obviously, the variable voltage swing impacts the speed at which the interconnect driver is able to charge of discharge the load capacitance. In the past, this has been exploited to identify voltage/frequency pairs for reliable link operation in presence of changing environmental conditions or application requirements.

In this work, we intend to apply the slack stealing technique to switch-link pipelines in on-chip networks for process variation tolerant operation. In practice, we propose to skew the clock of the switch output latches to compensate for variation induced gate delays. We then propose to make up for the reduced margins for signal propagation across the link by increasing the voltage swing. Since the effects of process variations are typically known only during the post-silicon testing, we need a self-calibrating mechanism in the link that tunes clock skews and drives the proper voltage swing to the link driver and receiver.

One of the main trends in the design of self-calibrating circuits is to empower a selected subset of standard flip-flops with error detection and correction capabilities. Figure 8.13 shows the architecture of the self-correcting FF that we have designed to recycle switch-link pipelines in on-chip networks. If the incoming data, due to process variations, arrives late, the main FF will sample it incorrectly, but the circuit is equipped with a certain number of additional detection and correction paths that will be able to tolerate delays up to approximately 50% of the clock period.

Our enhanced self-correcting flip flop has two execution modes, with well differentiated performance and power requirements: *calibration mode* and *normal execution mode.*

In calibration mode, an operating point controller compares the output of the main and of the delayed-sampling flip flops, assuming that the worst-case transition delay of the input signal is within the correct sampling window of the flip flop with the most delayed clock. Contrarily to similar circuits that cope with dynamic timing errors[5], our enhanced flip flop needs not only to detect timing violations, but also to quantify them. In fact, the voltage swing on the link will have to be adjusted accordingly, and we should take care of not over-designing the voltage swing to avoid link power inefficiencies. In con-

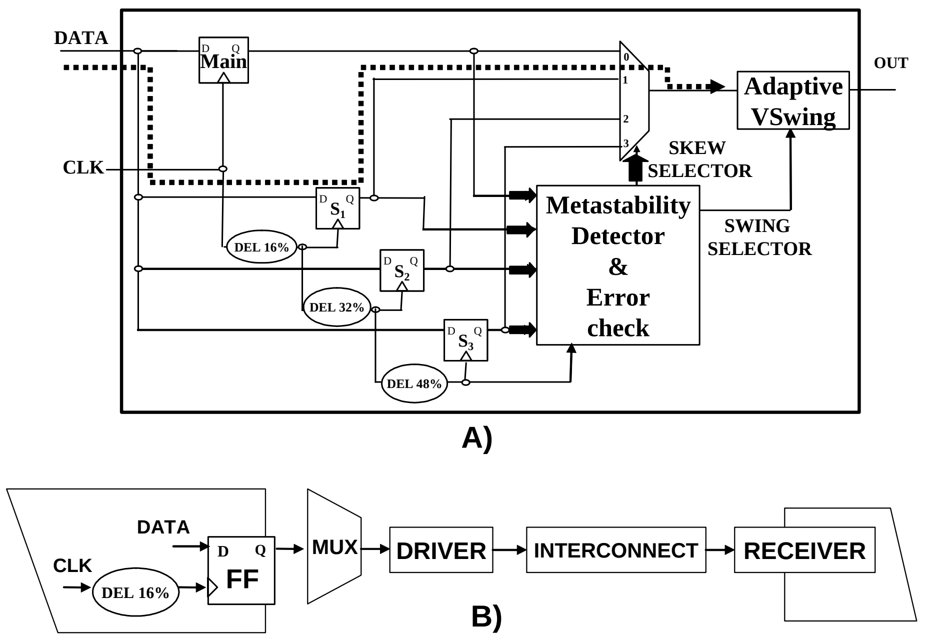


Figure 8.13: Self-correcting flip-flop followed by an adaptive swing driver

trast, similarly to [5], we need to cope with metastability. This scenario, where multiple delayed FFs sample the same input signal, is particularly sensitive to metastability, and we accounted for this in the controller design.

The controller compares all FF outputs (MAIN, S1, S2) with that of S3. When the output of a FF differs from S3 or is metastable, then the next one in the timing chain is selected for reliable operation. If there are no evidences of process variations, the controller selects the data from the main FF. The truth table of the controller is reported in Table 8.3.

| S2 | S1 | MAIN | SELECTED FF | Sel1 | Sel2 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | MAIN | 0 | 0 |
| 0 | 0 | 1 | S1 | 0 | 1 |
| 0 | 1 | 1 | S2 | 1 | 0 |
| 1 | 1 | 1 | S3 | 1 | 1 |

Table 8.3: **Truth table of the operating point controller.**

The calibration phase takes place during post-fabrication testing, and requires proper input patterns to be fed to the switch. In fact, switch critical paths need to be stimulated for proper detection of timing violations. Since then due to process variations paths that are non-critical under nominal conditions may become critical under certain variation scenarios, statistical static timing analysis (SSTA) needs to drive the test pattern selection process. Such analysis methodologies are so critical for efficient nanoscale designs that proper tooling support is becoming available [1] and will soon be mainstream. In addition, switches are easily testable as far as we are concerned. In fact, they usually exhibit a well identifiable number of critical paths going through the arbitration stage and through the crossbar selection signals. Moreover, a large timing gap does exist between the critical and the non-critical paths, thus making SSTA easier.

After calibration, the circuit enters in its normal execution mode. All FFs that do not belong to the selected path can be powered down using either clock- or power-gating techniques in order to save power. Active circuits during both calibration and execution modes are illustrated in Fig. 8.13(a) and (b) respectively. It is important to stress that no power overhead is caused by our self-calibrating link during normal execution mode.

Periodically, the chip could be re-tested and new problems caused by delay variations might arise depending on the age and the history of operating conditions of the chip. Our NoC links could be reconfigured in order to cope with the new conditions, thus resulting in the possibility to extend the lifetime of the chip by recharacterizing the links and their power consumption. This results in the possibility to effectively cope with wear-out effects.

#### Applicability

As regards applicability of our technique, a wide application domain can be envisioned in the NoC context. The requirement we pose on the NoC architecture is to expose the switch-link interface as a logic pipeline. As a consequence, switches implementing input buffering or virtual output queuing can be made variation tolerant with our circuit technique provided output latching is implemented. In principle, this might lead to an increased communication latency through the network, but a lack of output latching might give rise to long signal paths degrading global design performance. Circuit-switching architectures like the one reported in [8] or in Chapter 5 are ideal candidate for our self-correcting FFs, since they implement end-to-end flow control and feature retiming stages at switch inputs and outputs. Finally, our technique applies to output-buffered switches as well with a few modifications. First, the input to output buffers should feed our enhanced FF as well. During self-calibration, the controller selects the proper voltage swing for the switch output links. Moreover, it should drive a multiplexer that selects a proper delayed version of the clock for the whole output buffer. Second, input latching may be required to break long timing paths..

### 8.3.2 Link Design

Let us consider at first all circuit components that are active during normal execution mode and their relative design issues. We opted for full SPICE modelling of the whole communication channel, in order to accurately characterize the reliability-power trade-off. Our design effort can then be capitalized to develop cells for technology libraries to feed ASIC synthesis tools. We used 90nm Berekely Predictive Technology device models for our design and analysis framework.

A number of well-known schemes for low-swing signaling have been presented in the open literature[15]. After evaluation of several alternatives, we selected the same low-swing transmission scheme as in [11], and adapted it to our needs. The circuit schematic is reported in Fig. 8.14, except for the multiplexer whose

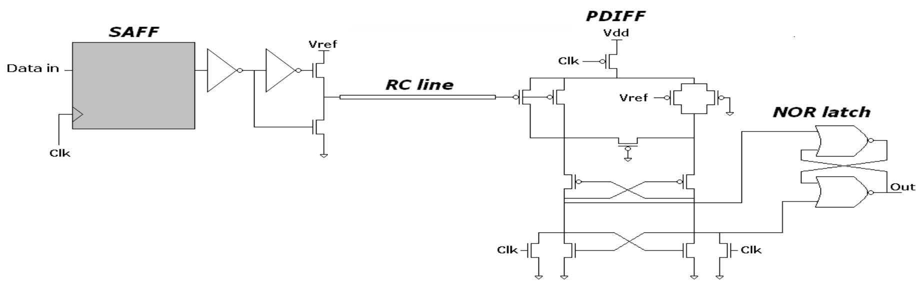


Figure 8.14: Low-swing signalling scheme

implementation was straightforward and which we could consider as a simple delay stage in what follows.

The interconnect is driven by a transmitter that takes the input stream from a flip flop. The receiver is composed by an amplifier and an output latch. We deployed a Sense-Amplifier Flip-Flop[6] scheme optimized for high-performance. It incorporates a precharged sense amplifier followed by a symmetric latch topology that significantly reduces delay and improves driving capability. The transmitter performs voltage-level conversion for low-swing signalling. The chosen receiver is a pseudo-differential circuit (*PDIFF*), consisting of a clocked sense amplifier and a static NOR output latch. It uses single wire per bit while at the same time retaining most of the advantages of differential schemes, namely low input offset and good sensitivity. Its major reliability degradation comes from the local device mismatch between the double input transistor pair, which usually can be controlled very well. The variation between distant REF's of the driver and the receiver also contributes some reliability degradations. The operation of the receiver is not sensitive to the supply noise, as opposed to other schemes. Since the FF samples on the rising edge of the clock and the receiver on the falling edge, it is possible to apply a clock signal with a phase shift between these two modules in order to increase the link throughput and/or to let the receiver sample more stable input signals. In order to simplify the clock network, we fed the *PDIFF* receiver with the negated clock, thus giving the signal an entire clock period for propagation across the link. A $\pi 3$ model was used for electrical level simulation of the interconnect[9], due to the reasonable trade-off between computation complexity and accuracy.

Our SPICE characterization of this interconnect scheme for 90nm technology indicated a maximum operating frequency of 2 GHz (2mm wire, 200mV voltage swing). This frequency is much larger than those achievable by state-of-the-art NoC prototypes, which hardly achieve more than 1 GHz. Therefore, we set our nominal performance constraint to an aggressive value of 1 GHz. For transistor sizing, we used the optimization engine that comes with HSPICE. For each component in the channel, we kept requesting increasing performance to the optimizer till an abrupt increase of transistor sizes was returned by the optimizer. An interesting trade-off is implicit in the sizing process. On one hand, we would like to have large slacks so to be able to allow more slack stealing to the upstream switch. On the other hand, large transitor sizing results in more power consumption and area overhead. The driver was resized for different interconnect lengths and hence wire load capacitances. We considered

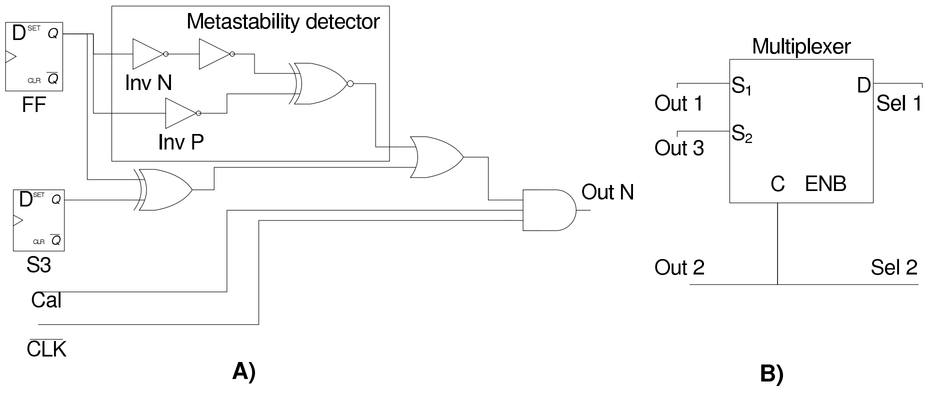


Figure 8.15: Operating point controller

link lengths ranging from 0.5 up to 4mm, which reflects typical NoC wiring scenarios.

We then designed the circuit components for the calibration phase. The most relevant optimization was performed on the operating point controller, whose logic structure is reported in Fig.8.15. In (a), we compared the output of a generic FF of index $N$ (MAIN, S1, S2) with that of the reliable S3 FF. At this stage, we have to consider that only 50% of the clock period is available for single-cycle calibration, since we need for wait for the sampling of S3 to have the reference logic value for comparison. When values differ or metastability is detected, a logic 1 generated on $OUTN$. All outputs at this stage then feed a combinational logic which returns the binary code of the selected FF. Observing the truth table of this combinational function from Table 8.3 we devised a straightforward mux-based implementation which is illustrated in Fig.8.15(b). $OUT2$ is not only the mux selector, but also the first digital output of the controller, the second being the mux output. Interestingly, in contrast to [5] our scheme does not suffer from short path concerns. In fact, while razor flip flops are employed at runtime to cope with dynamic timing errors, our enhanced FF only operates during predefined calibration phases wherein selected test patterns stimulate only critical paths.

We implemented the same metastability detectors as in [5], which detect the presence of a metastable voltage level. A detected metastability event is corrected the same way as a regular delay failure. The circuit consists of two inverter gates with different skewed P/N ratios, such that they switch at different voltage levels. If the two inverters interpret the result differently, the flip flop voltage is not definitive and may be metastable. Note however that complete system failure due to metastability cannot be completely avoided and only its probability of occurrence can be reduced to negligible levels. In fact, in our circuit there is a small but finite probability that the $OUTN$ signal itself becomes metastable, possibly causing a failure in the following cascaded decision logic. A number of standard techniques do exist to reduce the probability of such an event, trading reliability with latency. We are also aware that a reduction of the probability of having an input signal violating FF sample and hold constraints can be reduced by limiting the number of sampling flip flops. In the simplest case, a 2 FFs scheme could be used. However, we would have a coarser granu-

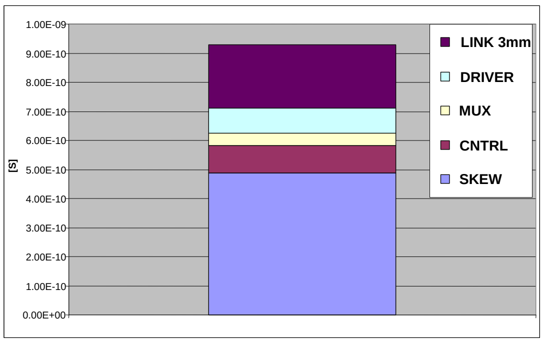


Figure 8.16: Critical path breakdown

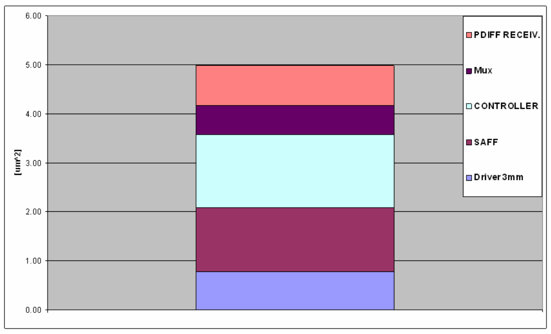


Figure 8.17: Area breakdown

larity in voltage swing adaptation, leading to conversative and power inefficient solutions. Techniques for a further reduction of the probability of metastability events are left for future work.

We optimized the critical path of the entire enhanced FF (going through the calibration circuitry) in order to allow for single-cycle calibration. Even forcing the voltage swing to its maximum value (1V), we achieved our goal only in presence of 2mm wires or below. For longer links, either adaptive strength drivers should be implemented, resulting in high area impact, or two cycles should be used for calibration. In this latter case, a retiming stage could be inserted along the link or clock stretching could be implemented at the receiver sampling stage. The critical path breakdown of the enhanced self-correcting FF is illustrated in Fig.8.16 for a 3 mm wire. As mentioned, calibration only has 50% of the clock period (i.e., 500ps) to evaluate.

The critical path goes through the controller, the mux selector signals, the driver and the link. For a 3mm link, the link contribution to the critical path is significant. The breakdown also shows the results of controller delay optimization. Even though the total delay is lower than 1ns, calibration does not work at 1 GHz since these delays are taken at 50% of the voltage swing (1V) and the receiver setup time is not accounted for in the plot.

Finally, area breakdown is illustrated in Fig.8.17. We can see that controller performance has been achieved at a significant area cost. Driver, receiver and FFs account for an almost equal share of area. Total area incurred by the

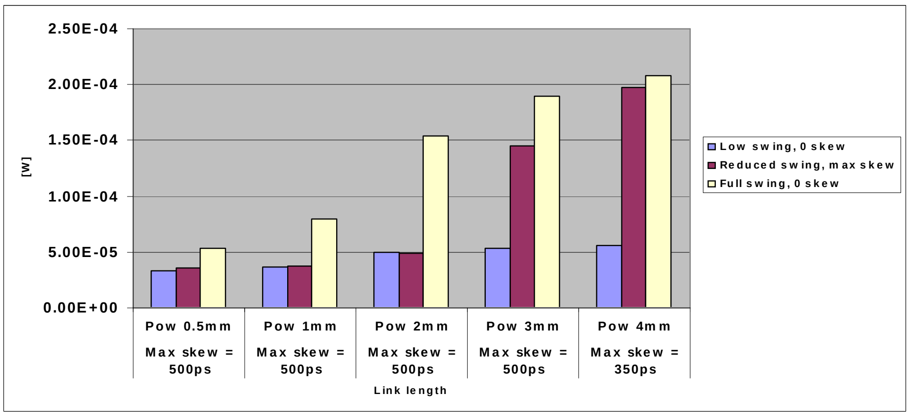


Figure 8.18: Link power saving reduction with clock skew

enhanced self-correcting FF amounts to 5 $um^2$. Of course this depends on the number of instantiated FFs (4 in this case, as in Fig. 8.13 A).

### 8.3.3 Experimental results

#### Mode power

Our first experiment intends to characterize power consumption of our self-calibrating link in calibration and normal execution mode. We chose a target 2mm link and instantiated 4 FFs in the enhanced self-correcting FF. In all modes, the link works at 1 GHz.

In calibration mode, measured power was 0.388mW. Please consider that this value includes not only link power, but the power of the entire communication channel, from the enhanced FF to the PDIFF receiver and NOR latch. Full swing voltage was enforced to meet the single cycle specification on calibration execution. In contrast, in normal mode the enhanced FF consumes like a normal flip flop, and power goes down to 50uW. In this condition, the voltage swing was kept at 200mV, since even with 50% skew the nominal swing was able to meet timing constraints. Even raising the voltage swing to 1V, in normal mode power would be bounded to 0.15mW.

#### Link power margins

The next experiment intends to characterize the margins for slack stealing made available by low-swing signaling interconnects and the link power overhead as a function of clock skew of the upstream logic stage. The plot in Fig.8.18 reports power consumption of our low-swing self-calibrating channel in three configurations and for different wire load scenarios. The leftmost bars indicate power when no clock skewing is applied, and hence the minimum voltage swing of 200mV is used. The rightmost bar reports power in the same case, but when voltage swing is arbitrarily raised to the full swing 1V value. Finaly, the intermediate bar describes the case where the maximum tolerable skew is applied in each configuration (up to a maximum of half the clock period), and

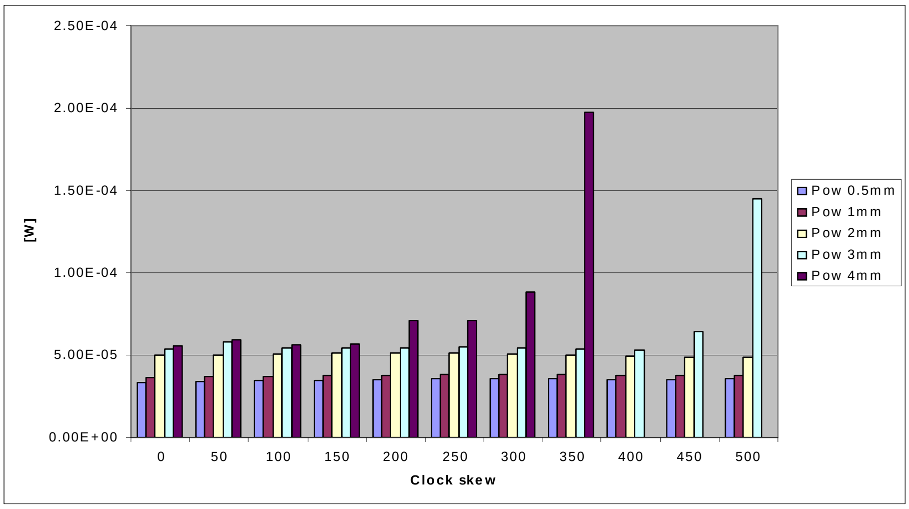


Figure 8.19: Voltage swing selection strategy

the relative power is highly impacted by the voltage swing needed to compensate that skew.

First, let us observe the first and the third columns. It is evident that low-swing signaling achieves a much smoother power behavior as a function of increasing wireload capacitance. The main difference is due to the interconnect capacitance being switched at its maximum swing. In practice, by implementing low-swing signalling in on-chip links we can accumulate power budgets that could be erased by process variations later on. In fact, the intermediate bar illustrates the way in which such budget is consumed by clock skew. For small wireload capacitances (from 0.5mm to 2mm links), the budget needs not to be consumed, since clock skew simply reduces the available slack. For longer links, such slack is not enough, and voltage swing needs to be increased with a significant reduction of power margins.

### Voltage swing selection policy

The third experiment aims at deriving a selection policy of the proper voltage swing on the link as a function of clock skew. The plot in Fig. 8.19 shows the power incurred by a link as a function of increasing clock skew in order to be able to continue working at the target frequency of 1GHz in normal execution mode.

A few things are evident. First, as the link length increases, even raising the voltage swing to 1V does not allow to compensate up to 50% clock skew. In contrast, for small wire loads the clock skew is compensated by absorbing the available slack, without the need for voltage swing raising. Second, as soon as the voltage swing is increased, power goes up rapidly. So, when possible, it might be better to drive a given wireload capacitance with a slightly overdimensioned driver instead of having a weak driver which is then forced to raise the voltage swing to compensate large clock skews. Unfortunately, this was not feasible for the 4mm link, since the optimizer would have returned overly large transistor sizes to appropriately drive its wire load. As a result, with 350ps skew, power incurred by the link is more than 4x with respect to the zero skew case.

## 8.4 Conclusions

This Chapter is split in two part: Architectural and Circuit Implications.

Regarding the architectural implications (Sec. 8.2) we demonstrate the benefits of using an emerging class of routing mechanisms in network-on-chip (NoC) to mitigate the impact of within-die process variability on the performance and yield of many-core silicon chips. Such a class of routing mechanism can be called variability-aware routing mechanism, in that they try to combine the need for scalable and compact routing implementation with the need to face variability-induced irregularity on the topology. We propose a new metric (coverage) to quantify the percentage of chips usable for a given frequency in the presence of variability. Using this metric, we analyze and evaluate the effectiveness of two specific routing implementations of different design complexity, namely LBDR and RbR, while considering link delay variations and spatial correlations that are representative in current (65nm) and future (22nm) processes. Our results based on comprehensive simulations of 4x4 and 8x8 2D mesh show that the two routing mechanisms provide a wide range of flexibility that allows the chip designer to choose either desired yield, performance, or a combination of both in post silicon configuration. Therefore, we believe that network-on-chip with variability-aware routing support will be increasingly important for improving the chip yield in the future. To this purpose, we prove that virtual channel availability would heavily relieve the requirements on the routing mechanism for designs targeting high irregularity coverage.

In Section 8.3 circuit implication we present the implementation and analysis of a variation tolerant version of a switch-to-switch link in a NoC. The goal is to tolerate the effects of process variations on NoC architectures using self-correcting links that automatically detect delay variations in upstream logic units and compensate them. The correction is applied without increasing the switch-to-switch latency by substituting the output flip-flops of the sending switch with a self-correcting flip-flop followed by an adaptive voltage swing selector. As a result, it is possible to tolerate delay variations at the cost of additional power consumption.

# Bibliography

[1] Synopsys PrimeTime VX Application Note - Implementation Methodology with Variation-Aware Timing Analysis.
Version 1.0, May 2007.

[2] L. Benini and G. De Micheli.
*Networks on Chips - Technology and Tools.*
Morgan Kaufmann, 2006.

[3] S .Bhunia, S. Mukhopadhyay, and K. Roy.
Process Variations and Process-Tolerant Design .
In *Int'l Conf. on VLSI Design*, pages 699–704, 2007.

[4] S. Borkar, T. Karnik, and V. De.
Design and Reliability Challenges in Nanometer Technologies.
In *Proc. of the 41st Annual Int'l Conf. on Design and Automation*, 2004.

[5] D. Ernst et al.
Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation.
In *Proc. 36th Int'l Symp. Microarchitecture (Micro-36)*, pages 7–18, 2003.

[6] A. Ghadiri and H. Mahmoodi.
Dual-Edge Triggered Static Pulsed Flip-Flops.
In *Proc. Int. Conf. VLSI Design*, pages 846–849, 2005.

[7] S. Kyo, S. Ohazaki, and T. Arai.
An Integrated Memory Array Processor Architecture for Embedded Image Recognition Systems.
In *Proc. of the 32nd Int'l Symp. on Computer Architecture (ISCA'05)*, 2005.

[8] S. Medardoni, D. Bertozzi, L. Benini, and E. Macii.
Control- and Data-path decoupling in the design of a NoC switch: area, power and performance implications.
In *Proc. of the International Symposium on System-on-Chip*, 2007.

[9] J. M. Rabaey.
*Digital Integrated Circuits: A Design Perspective.*
Prentice-Hall, 1996.

[10] M. Ruggiero, F. Angiolini, F. Poletti, D. Bertozzi, L. Benini, and R. Zafalon.
Scalability Analysis of Evolving SoC Interconnect Protocols.
In *Int. Symposium on System-on-Chip*, 2004.

[11] N. Terrassan, D. Bertozzi, and A. Bogliolo.

Spice-Accurate SystemC Macromodels of Noisy on-Chip Communication Channels.
In *Proc. of SPI-07*, 2007.

[12] Tilera Corporation.
http://www.tilera.com.

[13] F. Worm, P. Ienne, P. Thiran, and G. De Micheli.
An Adaptive Low Power Transmission Scheme for On-Chip Networks .
In *Proceedings of the 15th International Symposium on System Synthesis*, pages 92–100, Oct. 2002.

[14] F. Worm, P. Ienne, P. Thiran, and G. De Micheli.
A Robust Self-Calibrating Transmission Scheme for On-Chip Networks .
In *IEEE transactions on very large scale integration (VLSI) system,vol.13,no.1*, Jan. 2005.

[15] H. Zhang.
Low-swing On-Chip Signaling Techniques: Effectiveness and Robustness.
In *IEEE Trans. VLSI Systems*, pages 264–272, June 2000.

[16] , M. Mondal, T. Ragheb, X.W. Adnan Aziz and Y. Massoud, "Provisioning On-Chip Networks under Buffered RC Interconnect Delay Variations", ISQED '07.

[17] B. Hargreaves, H. Hult and S. Reda, "Within-die process variations: How accurately can they be statistically modeled?", ASPDAC 2008.

[18] A. Pullini et al, "NoC Design and Implementation in 65nm Technology", NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip, pp. 273–282, IEEE Computer Society, 2008.

[19] K.A. Bowman, S.G. Duvall, and J.D Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration", IEEE Journal of Solid-State Circuits.

[20] International Technology Roadmap for Semiconductors, 2007 Edition, available online at http://www.itrs.net/Links/2007ITRS/Home2007.htm

[21] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S. W. Keckler, and L. Peh, "Research Challenges for On-Chip Interconnection Networks", IEEE Micro, vol 27, no 5, pp 96–108, 2007.

[22] S. Herbert and D. Marculescu, "Characterizing chip-multiprocessor variability-tolerance", Proceedings of the 45th annual conference on Design automation, pp 313-318, 2008.

[23] A. Agarwal and M. Levy, "The Kill Rule for Multicore," in *Annual ACM/IEEE Design Automation Conference*, 2007.

[24] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," in *IEEE Micro Magazine*, Sept-Oct. 2007, pp. 51-61.

[25] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor," *IBM Journal of Research and Development*, Vol. 49, Num. 4/5, 2005.

[26] A. Mejía, J. Flich, J. Duato, S.A. Reinemo, and T. Skeie, "Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm for Meshes and Tori," in IPDPS, 2006.

[27] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C. Thacker, "Autonet: A High-speed, Self-configuring Local Area Network using Point-to-point Links," Technical Report SRC research report 59, DEC, April 1990.

[28] J. Flich, S. Rodrigo, and J. Duato, "An Efficient Implementation of Distributed Routing Algorithms for NoCs," in *Int. Symp. on Networks on Chip*, 2008.

[29] J. Flich, A. Mejía, P. López and J. Duato, "Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chip," in *Int. Symp. on Networks on Chip*, 2007.

[30] G. Paci, D. Bertozzi and L. Benini, "Effectiveness of Adaptive Supply Voltage and Body Bias as Post-Silicon Variability Compensation Techniques for Full-Swing and Low-Swing On-Chip Communication Channels", to be presented at *Date*, 2009.

# Chapter 9

# Power-Optimal RTL Soft-Macro Selection Strategy for Leakage-Sensitive Technologies

## 9.1 Introduction

Energy efficient design is an active area of research and development in semiconductor electronics [3, 5, 4]. The proliferation of multimedia-enabled portable consumer products urges designers to deliver high computation power while preserving battery lifetimes. Even for non-battery operated devices, thermal dissipation limits posed by conventional packaging and cooling technologies might slow down the rate of system integration. Moreover, in the nanometer regime an increasing fraction of total power in high-performance digital circuits is due to leakage currents [12]. For microprocessors, projections envision that a 50% share of the total power for sub-65nm technologies will be contributed by the leakage power dissipated by the functional sub-units [33].

As a consequence, traditional low-power design techniques and methodologies are being upgraded with careful consideration of all sources of power dissipation, including leakage-power [6]. Most techniques for leakage-control are at the physical, circuit and logic level [24]. Design time techniques exploit the delay slack in non-critical paths to reduce leakage, while runtime techniques for leakage control aim at placing idle circuits in a standby low-leakage state [27].

RTL low-power techniques are mostly devoted to dynamic power optimization [24]. Unfortunately, not properly accounting for leakage power at RTL or above might result in highly suboptimal designs from a power perspective, since decisions taken in the early design stages play a significant role in determining final performance and power figures. In particular, it cannot be neglected that leakage and dynamic power reduction are sometimes conflicting requirements.

For instance, the effectiveness of traditional RTL power management techniques such as clock gating can be impaired by the increasing role of leakage power. In addition, it turns out to be very difficult to apply conventional leakage power reduction techniques to circuits with clock gating [19]. Another example concerns voltage scaling which, under constant performance constraints, carries an implied trade-off between dynamic and leakage power savings. In fact, as voltage is reduced, logic cells will be slower and there will be less opportunity to use low-leakage cells.

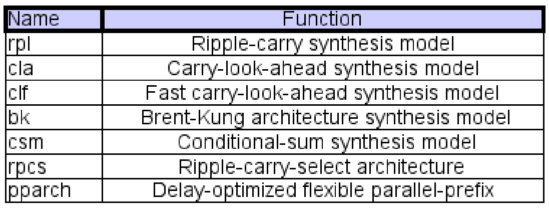
As a consequence, developing power efficient products will require consideration of static power in IC design front-ends, possibly exploiting high-level leakage estimators or macromodels incorporated in ASIC design flows [25, 26]. The most straightforward high-level technique for power optimization consists of design-time architectural inferences based on specific design constraints. This is a well-known technique to bring leakage-awareness to behavioural synthesis algorithms and tools [10, 20]. Functional units are identified, that can be replaced by low-leakage variants while not sacrificing performance.

In this chapter we will present a unexploited opportunity for leakage-aware low power RTL design that consists of properly tuning selection strategies for RTL soft macros. In particular, RTL synthesis tools automatically infer basic HDL arithmetic or functionals operators into specific unit implementations depending on performance and area optimization constraints. It happens also during the synthesis process of a switch or a network interface where, before to do the real synthesis, it is inferred some operator (like *increment* or *decrement*) or comparator.

Default soft-macro selection strategies need to be revisited in light of the new trade-offs spanned by designs in nanoscale technologies. First, power optimizations cannot be postponed to post-synthesis steps but need to be taken into account already at soft-macro selection. Second, leakage control needs to be an integral part of power-aware selection strategies. Third, suitability of different macro implementations to power-driven gate level transformations or to low-leakage cell insertion techniques performed by advanced synthesis tools need to be carefully assessed.

It is proposes a technique for optimizing the selection of RTL arithmetic soft macros with respect to total power dissipation. Without lack of generality, we restrict our analysis to adder macros and explore their total power as a function of different design constraints (performance, area), technology libraries (single vs multi-Vth) and working conditions (switching activity). We come up with design tables instructing the synthesis tool to make the most power-efficient inference choice for the design at hand. By applying our design methodology to the DesignWare (DW) library of synthesizable components [13], we achieved a significant 43% power saving with respect to the default macro selection strategy of Synopsys.

The chapter is structured as follows. Section 9.2 illustrates our synthesis methodologies, that lead to the experimental results of Section 9.3. Section 9.4 reports a comparison of our DW selection strategy with Synopsys default one and conclusions are drawn in Section 9.5.



| Name | Function |
|---|---|
| rpl | Ripple-carry synthesis model |
| cla | Carry-look-ahead synthesis model |
| clf | Fast carry-look-ahead synthesis model |
| bk | Brent-Kung architecture synthesis model |
| csm | Conditional-sum synthesis model |
| rpcs | Ripple-carry-select architecture |
| pparch | Delay-optimized flexible parallel-prefix |

Figure 9.1: DesignWare adder portfolio.

## 9.2 Synthesis methodology

We choose adder soft macros from the Synopsys DesignWare library to prove the effectiveness of our characterization and power-aware selection methodology. We assume that parametric, synthesizable and technology independent descriptions are provided for each soft macro. For instance, the DesignWare portfolio [14] of adder macros includes seven alternative implementations, as reported in Fig.9.1. We synthesized a set of adder macros for different values of the bitwidth, ranging from 8 to 16, 32 and 64 bits. We target 65nm low power technology libraries from STMicroelectronics, assuming the typical corner case. High-Vth (HVT), Standard-Vth (SVT) and Low-Vth (LVT) library variants were made available for synthesis tool-driven leakage optimizations. For the sake of RTL synthesis, the driving strength of a simple flip flop library cell is set for all inputs and a fan-out-of-four equivalent load is placed on each output.

### 9.2.1 Performance-optimized synthesis flow

Our first objective is to characterize critical path and area of the different adder implementations when synthesized for maximum performance and as a function of data parallelism. The synthesis flow for this purpose is illustrated in Fig.9.2-left. Each instance is inferred from HDL and synthesized with Physical Compiler by linking only the low-Vth technology library. The output of the physical synthesis step consists of timing and area reports, an $SDF$ file (information for timing-accurate analysis) and a $spef$ file (parasitic capacitance annotation). Then gate level simulation is performed, driven by a testbench where adders are instantiated and their inputs properly driven. The real switching activity is back-annotated and fed to Synopsys PrimePower[15] for power estimation. The use of the real switching activity and the small area of the adder instances under test are likely to result in accurate power reports. We refer to section 9.3.1 for the results obtained with this flow.

### 9.2.2 Power-optimized synthesis flow

The second synthesis flow we set up aimed at applying power optimizations to the different adder implementations, taking advantage of the slack, i.e., the difference between the target delay and the critical path delay of adders. This latter is always derived from the synthesis for maximum performance, so that for each target delay the maximum slack is available for power optimizations. The low-Vth technology library is again linked to Physical Compiler, since we

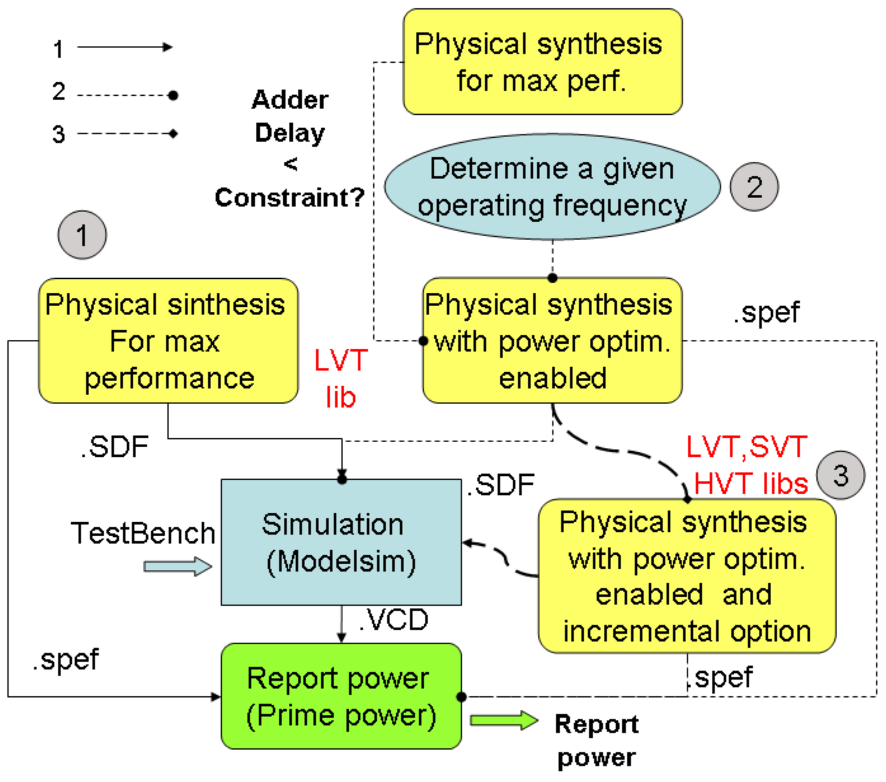

1
2
3
Physical synthesis for max perf.
Adder Delay < Constraint?
Determine a given operating frequency
2
1
Physical sinthesis For max performance
LVT lib
Physical synthesis with power optim. enabled
.spef
.SDF
LVT,SVT HVT libs
3
.SDF
TestBench
Simulation (Modelsim)
Physical synthesis with power optim. enabled and incremental option
.VCD
.spef
Report power (Prime power)
.spef
Report power

Figure 9.2: Synthesis flows. Left: optimized flow for maximum performance. Middle: flow for power optimization. Middle/Right: multi-Vth synthesis flow

intend to explore multi-Vth technologies only later on and restrict our analysis to power-driven netlist transformations.

Fig.9.2-middle shows the steps of the power-optimized flow. All adder instances are first inferred and synthesized for max. performance, thus deriving the shortest critical path for each of them. Then, we span the target performance parameter, and for each value of the target delay we select those adders whose delay meets the performance constraint. Their RTL description is then fed to a second physical synthesis step. For this latter, we enable dynamic and leakage power optimizations of the Physical Compiler. The power estimation methodology is the same as before.

This flow was set up in order to assess whether it is more power-efficient, for a given target performance, to deploy adders whose delay almost exactly fit the target delay (and which generally consume less area) or much faster ones (and generally more area consuming), which exhibit a larger slack available for power optimizations.

### 9.2.3 Multi-Vth synthesis flow

The third flow aimed at assessing the impact of deploying multi-Vth technology libraries for leakage power optimizations. We of course expect a reduction of leakage power, but the interaction of dynamic and leakage power optimizations is the specific focus of our analysis. It is important to observe that we only use the power optimization directives exposed by the synthesis tool, and we try to take the maximum advantage out of them through customized power-aware design flows.

As showed in Fig. 9.2-right, the first three steps are in common with the single threshold flow. We then added an incremental synthesis step where all LVT, SVT and HVT libraries are linked, keeping the same leakage and dynamic power optimizations as before, and even the same timing constraints. We got the best power savings with this methodology, as discussed in Section 9.3.3.

## 9.3 Experimental results

### 9.3.1 Energy characterization of adder units

To prove the practical viability of our design methodology, we focused on the DesignWare library of synthesizable adders, managed by Synopsys tools. We therefore applied the above synthesis flows to the adder implementations of Fig.9.1.

The performance-optimized design flow illustrated in section 9.2.1 led to the results of Fig.9.3 and 9.4. All 8 bit adders have similar critical path delays but *RPL* and *RPCS* exhibit poor scalability as data parallelism increases. From an area perspective, *PPARCH, RPL* and *BK* show lower occupancies for all bit-widths, while *CSM* and *CLA* exhibit the largest area. Interestingly, *RPCS* has limited area till 32 bits, but then shows an abrupt increase for 64 bit inputs, and ends up even surpassing other area-hungry architectures.

These data pave the way for a power characterization of the adder implementations. We assume each adder to work at its maximum performance, i.e., its critical path delay in the performance-optimized netlist is used as the clock

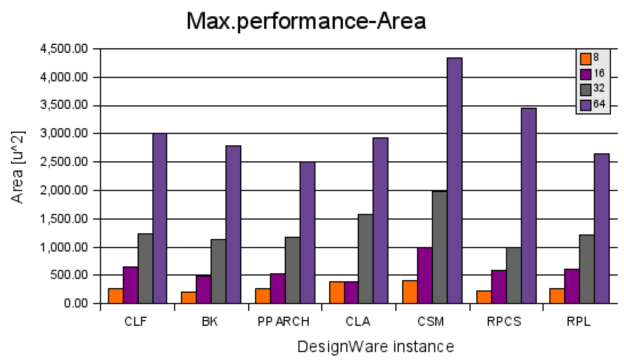


Figure 9.3: Area of adder implementations when synthesized for max performance. Scalability with bitwidth showed.

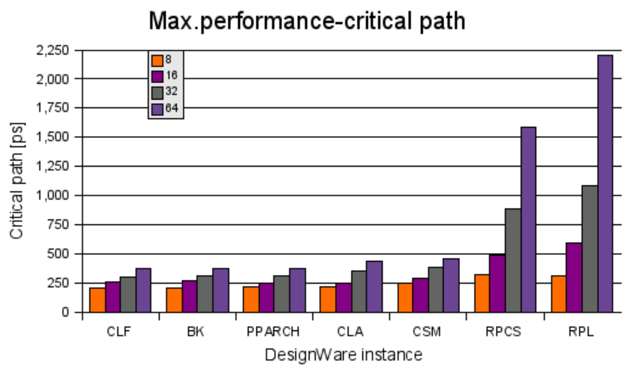


Figure 9.4: Critical path of adder implementations when synthesized for max. performance. Scalability with bitwidth showed.

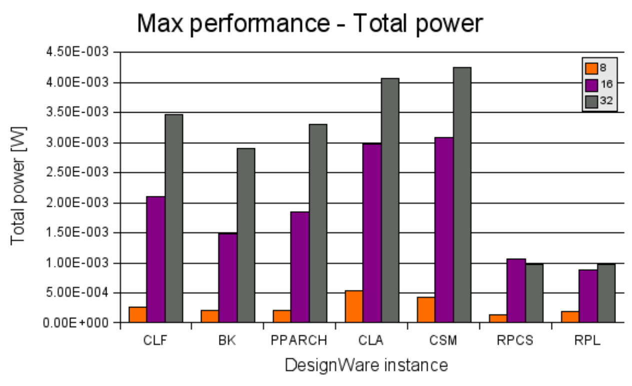


Figure 9.5: Power scaling as a function of bitwidth for high performance adder implementations.

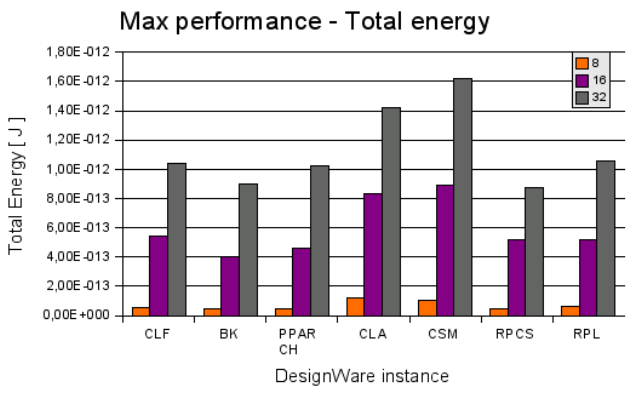


Figure 9.6: Energy scaling as a function of bitwidth for high performance adder implementations.

period which synchronizes application of new input patterns. We define the *Data Introduction Rate (DIR)* as the rate at which new data patterns are applied at adder inputs. This is an experimental way of controlling the switching activity. A *DIR* of $1/N$ means an addition computed every $N$ clock cycles. A maximum *DIR* of 1 is used for power characterization, where all permutations of input bits are generated for an 8-bit parallelism, while a random subset of 10000 input configurations is applied to higher bitwidth adders.

Power results are illustrated in Fig.9.5. For all bitwidths, *RPL* and *RPCS* are the most power-efficient instances, with power savings increasing with the bitwidth. Please also note that it is not always true that adders with higher area also consume more total power. Due to the deployment of a low-power technology library, leakage power was found to be negligible, therefore total power is almost completely determined by dynamic power. Even in these conservative conditions, we will prove that leakage power cannot be neglected at all for the selection of power-optimal adder instances from the DW library.

We are now able to answer the following question: given a certain amount of

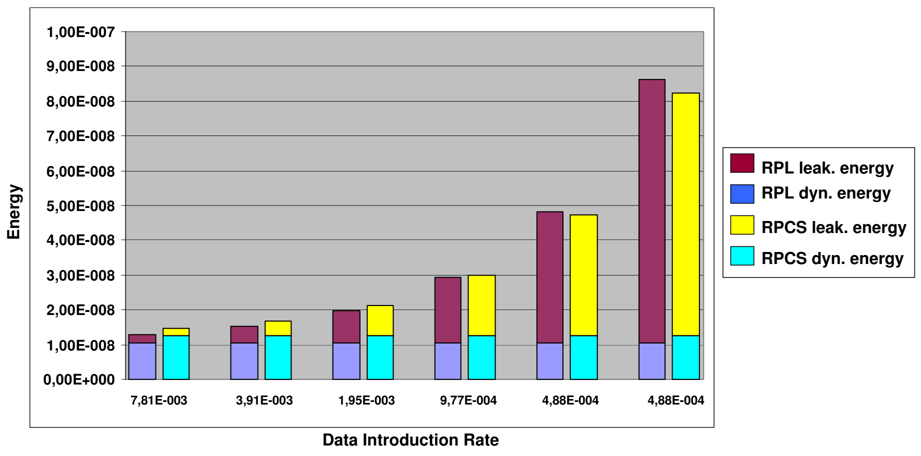


Figure 9.7: Total energy as a function of *DIR* for *RPL* and *RPCS* adder implementations.

additions to be executed, which adder instance can do the job while consuming the least energy? The answer is provided by the energy plot in Fig.9.6, showing that the most energy-efficient implementation depends on the target bitwidth: *RPCS* for 32 bit adders, *BK* for a lower data parallelism.

Till now, the maximum *DIR* was used. However, as the switching activity of adders is decreased, leakage power associated with idle cycles comes into play. To better understand this, the same clock period was considered for all adder instances, equal to that of the slowest implementation (*RPL*). Energy results are plotted as a function of decreasing *DIRs* in Fig 9.7. Total energies (and their breakdowns) of only *RPL* and *RPCS* are showed, since they are the most energy-efficient adder instances at a *DIR* of 1. The same number of additions is performed at each *DIR* value, therefore dynamic energy for each adder stays the same throughout the plot and is almost equal to the asymptotic value on the left. The increase in energy is only determined by leakage energy. Two interesting observations can be done here. First, a *DIR* does exist for each adder instance at which leakage energy becomes larger than dynamic energy. Second, the plot shows total energy crossing between *RPL* and *RPCS* for a *DIR* of $6.99e^{-4}$ (1 addition every 1.43 usecs). The crossing behavior indicates that an adder instance consuming less dynamic energy than another consumes more leakage power. As a consequence, more total energy is incurred as the *DIRs* decreases. As a result, had we considered *RPL* as the most energy-efficient adder instance irregarding of *DIR* and therefore of the impact of leakage power, we would have inferred a sub-optimal instance.In fact, for low *DIRs*, *RPCS* is the most energy-efficient implementation. We conclude that, even for low-power technology libraries, consideration of leakage power is key for selecting power-efficient functional module implementations for sub-65nm nodes, and the switching activity parameter becomes an integral part of selection algorithms.

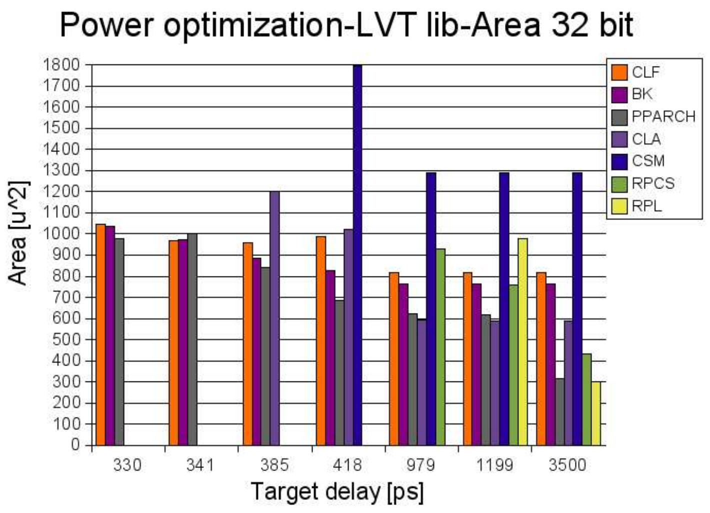


Figure 9.8: Area results with power optimizations enabled. Low-Vth technology library used.

### 9.3.2 The impact of power optimizations

Synthesis tools are able to apply a number of power optimizations while performing RTL synthesis which might put the above results in discussion. Most of these techniques exploit slack availability with respect to target performance requirements. For a given target delay, we wonder whether it is more power-effective to deploy adder instances that closely fit the requirement or much faster ones that can potentially result in larger power optimizations. Since adders belonging to the first category are likely to incur lower area, the overall power balance with leakage awareness is an interesting analysis topic.

We proceeded as follows. We adopted the synthesis flow for low-power illustrated in section 9.2.2. We considered a target performance requirement for all adders, and a maximum *DIR* was initially assumed. As we increase the target delay parameter, slower adder instances come progressively into play. Physical Compiler power optimizations were enabled, trying to exploit the slack of each adder and resulting in gate netlist transformations. Area and power results for this kind of analysis are reported in Fig.9.9 and Fig. 9.8. Since power proved almost independent of bitwidth, only results for 32 bit adders were showed. We observe that, in contrast to the results of section 9.3.1, there is a one-to-one correspondence between area and total power, meaning that higher area always results in higher total power. It can also be clearly observed that as the target delay increases, newly considered adders (critical path delay almost equal or slightly lower than target delay) are never the most power efficient solutions. Faster adders provide area and power savings when used with larger target delays, since the synthesis tool is able to scale their area and performance to maximally exploit the available slack. Some implementations (such as *PPARCH*) prove more scalable than others (e.g.,*CSM*) over a wide range of performance requirements, as a consequence of their specific architecture.

At a maximum *DIR* of 1, adder instances with higher area also have higher

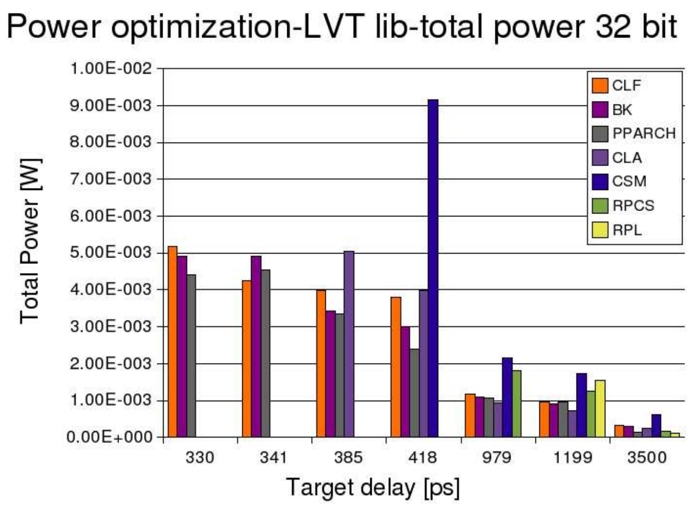


Figure 9.9: Power results with power optimizations enabled. Low-Vth technology library used

total power. Moreover, we experimentally verified that higher area also results in higher leakage power. Therefore, if we plot adder energy as a function of the *DIR*, we do not notice any curve crossings, since instances with higher total power at maximum *DIR* also consume more leakage power. As a consequence, for a single-threshold technology library and power optimizations enabled in the synthesis tool, the DW selection strategy for power-optimal adder instances does not need to take the *DIR* into account. More in general, we expect this to be true whenever higher area translates into higher leakage power and total power.

### 9.3.3 The impact of multi-Vth design

In order to decrease the impact of leakage energy as the *DIR* is decreased, we introduce multi-threshold design. Unfortunately, by just extending the design flow of section 9.2.2 to multi-threshold technology libraries (linking HVT, SVT and LVT libraries instead of just LVT), we interestingly get worse total power results, as illustrated in Fig. 9.10 (label $All - lib$ compared with $LVT - lib$).

This is because the synthesis tool is instructed to perform dynamic power optimizations in parallel with multi-Vth cell insertion. In some cases, inserting high-Vth cells shortens the total positive slack and prevents high-impact netlist transformations to cut down on dynamic power in the following optimization cycles of the tool. In our case, since dynamic power is dominant, we would like to give it higher priority in power optimizations, and to leave high-Vth cells insertion as an incremental optimization step. We therefore set up the optimized flow for multi-Vth design illustrated in section 9.2.3.

Incremental total power savings were obtained with respect to Fig.9.8, as indicated by Fig.9.10 (label $All - lib - new$) for the $PPARCH$ architecture. Please note that such savings are mostly related to a decrease of dynamic power, which takes advantage of the incremental optimization step. The decreasing

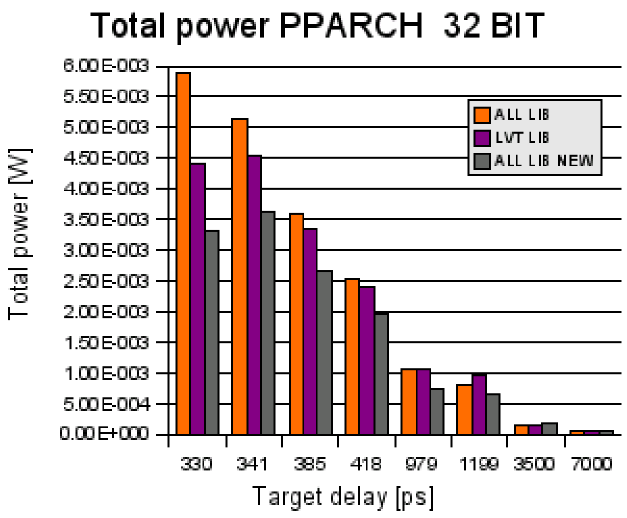


Figure 9.10: Impact of synthesis methodology on the total power of the PPARCH adder instance.

trend of total power is due to the decrease of operating frequency and to the reduction of the number of total cells placed by the synthesis tool. In order to assess the leakage reduction as well, we have to look at Fig.9.11. It shows total energy of the *PPARCH* adder as a function of decreasing *DIR* for single-threshold and multi-threshold libraries. Leakage energy savings for the multi-Vth instance span several orders of magnitude.

The reason behind this lies in the progressive replacement of low-Vth cells with high-Vth cells as the target delay is increased (performance requirement relaxed). The behavior of the synthesis tool is represented in Fig. 9.12, showing the percentage of HVT, SVT and LVT cells on the total number of placed cells. For short target delays, low-Vth cells will be on the critical path, and high-Vth cells in off-critical paths. As the timing constraint is relaxed, all cells tend to be progressively replaced by their high-Vth variants.

We do not report area and total power plots for all adder instances, since they are very similar to those already seen in Fig. 9.8 and 9.9, although absolute values are lower due to the incremental optimization run. There is however a significant difference in the energy behaviour as a function of *DIR*. In this case (i.e., multi-Vth design), we have a direct relation between area and total power, but not between area and leakage power. In fact, for some target delays there are adder instances featuring larger area but making use of high-Vth cells only. If compared with adders with smaller area but still deploying low-Vth cells, the former adders consume less leakage power. As a result, energy curve crossings can take place as the *DIR* decreases. An example thereof is illustrated in Fig.9.13.

When the target delay allows full insertion of high-Vth cells in all adder instances, then leakage power is again directly related to area, and no energy crossings will occur. This has heavy implications on the DW selection strategy. If crossings can occur, the *DIR* must be taken into account for the power-

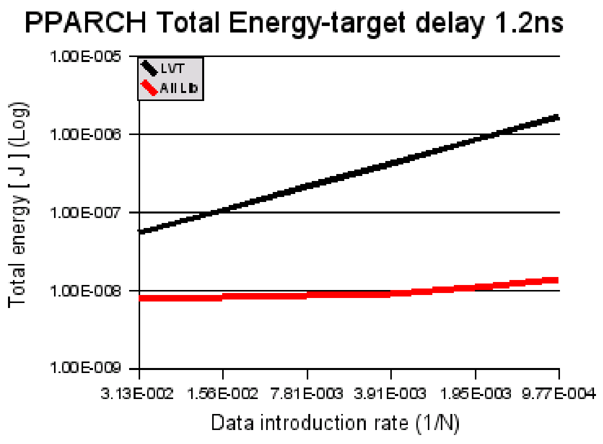


Figure 9.11: Savings on leakage energy as *DIR* decreases

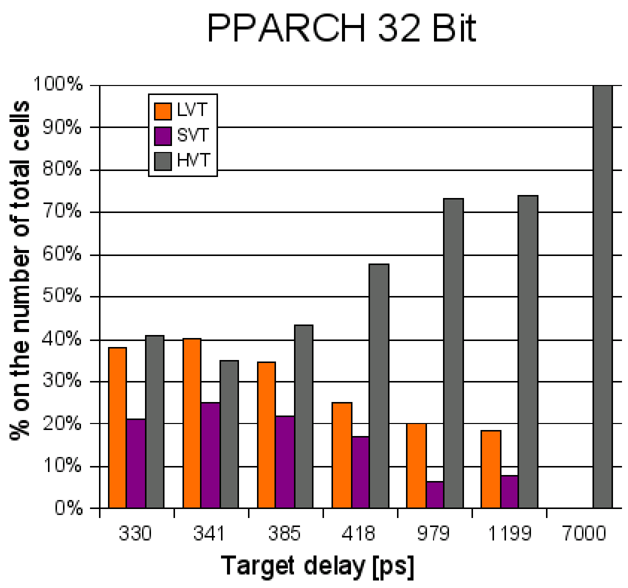


Figure 9.12: Percentage of cells with different Vth on the total number of cells for the *PPARCH* architecture.

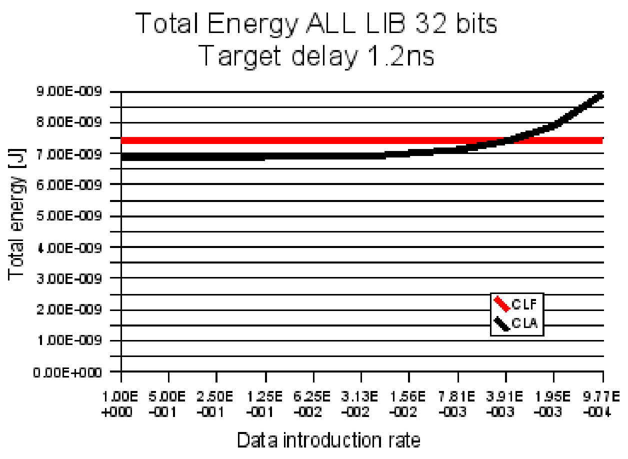


Figure 9.13: Total energy crossing example with multi-threshold library.

optimal inference of adder implementations. If not, the selection strategy does not depend on *DIR* considerations.

## 9.4 Comparison with default DW selection strategy

The default adder inferred by Synopsys tools in compliance with design constraints is *PPARCH*. We verified that this is indeed a good choice given the results reported in Fig.9.14. They illustrate the range of target delays (y-axys) over which the synthesis tool was able to perform power-driven gate netlist transformations for the different adder implementations. The lower point denotes the tighter performance requirement that each adder is able to meet. Clearly, some adders allow a better exploitation of the available slack, and allow power optimizations over a wider range of target delays. From the upper side of the bars on, the further slack which is made available cannot be exploited any more. *PPARCH* and *RPL* show the best scalability results. When multi-Vth technology libraries are used, the optimization window is obviously larger since replacement of low-Vth cells with high-Vth ones is an additional degree of freedom to exploit available slack.

These results perhaps explain why the *PPARCH* implementation is the default adder inferred by Physical Compiler in compliance with design constraints. However, our power exploration has showed that *PPARCH* is not always the most power-efficient solution (recall Fig.9.8 and 9.9), and power savings should be expected by a more accurate DW selection strategy. We summarize the results of our power exploration without and with multi-threshold libraries in the Tables 9.1, 9.2 and 9.3, selecting the most power-efficient adder implementation for varying design constraints and data parallelism (32 and 64 bits). Please note that when only an LVT library is used, the optimal selection does not depend on the *DIR* due to the direct relation proved between adder area,total power and leakage power.

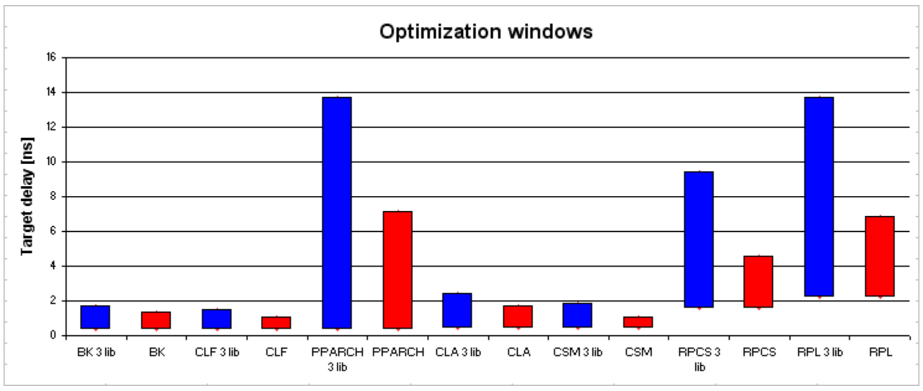


Figure 9.14: Optimization windows for the different adders with single- and multi-threshold design.

| Target Delay | Adder 64 bit | | Target Delay | Adder 32 bit |
|---|---|---|---|---|
| 400-700 ps | PPARCH | | 330-900 ps | PPARCH |
| 700-4000 ps | CLA | | 900-2500 ps | CLA |
| 4000ps-on | RPL | | 2500ps-on | RPL |

Table 9.1: Inference table of power-optimal 32- and 64-bit adder implementations for **LVT library**

Our DW selection strategy was then compared with the default one of Physical Compiler under different design constraints and with maximum *DIR*, and power dissipation was measured for the adder instances inferred by the two strategies. As Fig.9.15 and 9.16 prove, we achieved up to 22% power savings for a single-Vth library and up to 43% savings for the multi-Vth case. Please note that 100% dissipation means that our and Synopsys' strategies inferred the same adder instance. Larger savings are expected if the datapath works at a very low *DIR*, since in that region leakage energy dominates and instantiating the power-optimal adder (if different from *PPARCH*) achieves increasing energy savings with increasing idleness.

## 9.5 Conclusions

In this chapter is proposed an RTL technique for power reduction performing smart selection of RTL adder soft-macro implementations based on an extensive power pre-characterization framework. We proved that a combination of adder architecture features, synthesis tool capabilities and design flow optimizations for low power determine the most-power efficient adder implementation for different design constraints. Even conservatively deploying technologies already optimized with respect to leakage power and multi-Vth design, we show that leakage power cannot be neglected for power-aware selection strategies of RTL functional units, and that correctly estimating switching activity is key to come up with efficient solutions.

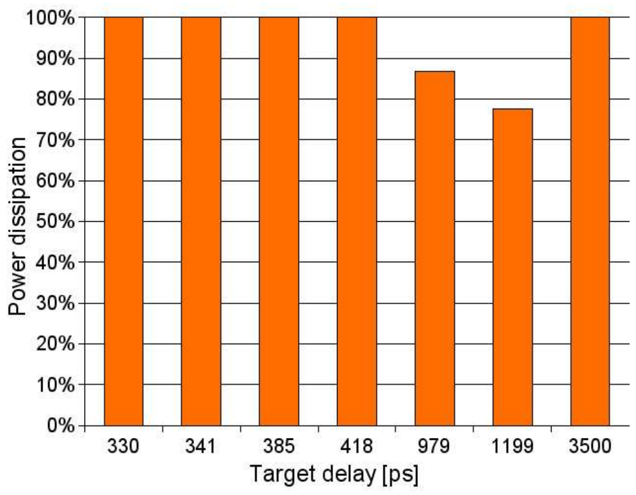


Figure 9.15: Power savings wrt Synopsys default inferences of 32 bit adders using single-Vth library.

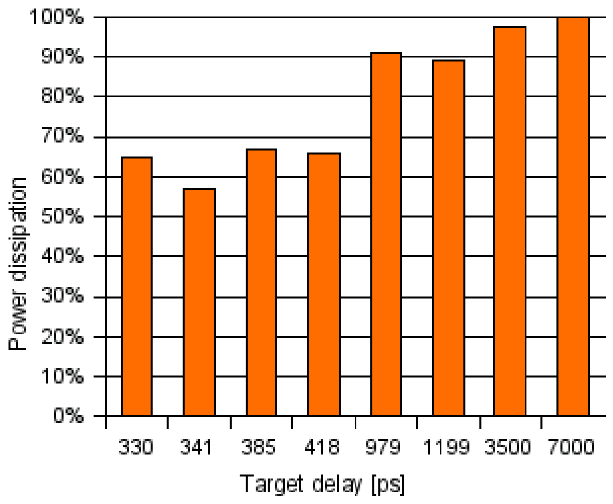


Figure 9.16: Optimization windows for the different adders with single- and multi-threshold design using multi-Vth library.

| | 1 | 2.5E-1 | 7.94E-3 | 5E-3 | 1.82E-3 | 1.28E-3 | 1.09E-3 | 1.05E-3 |
|---|---|---|---|---|---|---|---|---|
| 407 ps | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC |
| 484 ps | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC | BK |
| 500 ps | PPARC | PPARC | PPARC | PPARC | PPARC | PPARC | BK | BK |
| 1300 ps | CLA | CLA | CLA | BK | BK | BK | BK | BK |
| 1750 ps | CLA | CLA | CLA | CLA | BK | BK | BK | BK |
| 2420 ps | CLA | CLA | CLA | CLA | CLA | BK | BK | BK |
| 7100 ps | CLA | CLA | CLA | CLA | CLA | CLA | CLA | CLA |
| 7500 ps | RPL | CLA | CLA | CLA | CLA | CLA | CLA | CLA |
| 13800 ps | RPL | RPL | RPL | RPL | RPL | RPL | RPL | RPL |

Table 9.2: Inference table of power-optimal 64-bit adder implementations for **multi-Vth libraries**. x-dimension is *DIR*, y-dimension is target delay.

| | 1 | 1.67E-1 | 7.94E-3 | 3.77E-3 | 2E-3 | 1.79E-3 | 1.09E-3 | 1.05E-3 | 1E-3 |
|---|---|---|---|---|---|---|---|---|---|
| 330 ps | CLA | CLA | CLA | CLA | CLA | CLA | CLA | PPARCH | PPARCH |
| .340ns | CLA | CLA | CLA | CLA | CLA | CLA | CLF | CLF | CLF |
| .38ns | CLA | CLA | CLA | CLA | CLA | CLF | CLF | CLF | CLF |
| .48ns | CLA | CLA | CLA | CLA | CLA | CLA | CLA | CLA | PPARCH |
| .97ns | CLA | CLA | CLA | CLA | CLF | CLF | CLF | CLF | CLF |
| 1.2ns | CLA | CLA | CLA | CLF | CLF | CLF | CLF | CLF | CLF |
| 3.5ns | RPL | CLA | CLA | CLA | CLA | CLA | CLA | CLA | CLA |
| 3.8ns | RPL | RPL | RPL | RPL | RPL | RPL | RPL | RPL | RPL |

Table 9.3: Inference table of power-optimal 32-bit adder implementations for **multi-Vth libraries**. x-dimension is *DIR*, y-dimension is target delay.

# Bibliography

[1] S.Heo, K.Asanovic
"Leakage-biased domino circuits for dynamic fine-grain leakage reduction",
Symposium on VLSI circuits, pp.316-319, 2002.

[2] S. Narendra et al.,
"Scaling of stack effect and its application for leakage reduction",
International Symposium on Low Power Electronics and Design, pp. 195-200, Aug. 2001.

[3] E. Macii (Editor)
"Ultra Low-Power Electronics and Design",
Kluwer Academic Publishers,2004, Dordrecht, The Netherlands.

[4] M.Pedram and J.M. Rabaey (Editor)
"Power aware design methodologies",
Kluwer Academic Publishers,2002, Dordrecht, The Netherlands.

[5] Laurie Kelly, Piguet Piguet, Christian (EDT) Piguet
"Low-Power Electronics Design",
CRC Press,Series: Computer Engineering Series Volume: 1, 2005,CSEM, Neuchatel, Switzerland.

[6] K.Bernstein, C.Chuang, R.Joshi, R.Puri
"Design and CAD Challenges in sub-90nm CMOS Technologies",
2003 International Conference on Computer-Aided Design (ICCAD '03), p. 129-136.

[7] M. C. Johnson, D. Somasekhar, L. Chiou, and K. Roy,
"Leakage control with efficient use of transistor stacks in single threshold CMOS",
IEEE Trans. VLSI Syst., Vol. 10, No. 1, pp. 1-5, Feb. 2002.

[8] E.Macii, M.Pedram, and F.Somenzi
"High-Level Power Modeling, Estimation, and Optimization",
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 17, Issue 11, Nov. 1998 Page(s):1061 - 1079.

[9] A.Khan, P.Watson, G.Kuo, D.Le, T.Nguyen, S.Yang, P.Bennett, P.Huang, J.Gill, C.Hawkins, J.Goodenough, D.Wang, I.Ahmed, P.Tran, H.Mak, O.Kim, F.Martin, Y.Fan, D.Ge, J.Kung and V.Shek
"A 90-nm Power Optimization Methodology With Application to the ARM 1136JF-S Microprocessor",
IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 8, AUGUST 2006.

[10] K.S.Khouri,N.K.Jha,
"Leakage Power Analysis and Reduction During Behavioral Synthesis",
IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 10, NO. 6, DECEMBER 2002.

[11] K.Shin, T.Kim
"Leakage power minimisation in arithmetic circuits",
Electronics Letters - 1 April 2004 - Volume 40, Issue 7, p. 415-417.

[12] S.G. Narendra
"Challenges and Design Choices in Nanoscale CMOS",
ACM Journal on Emerging Technologies in Computing Systems (JETC) Volume 1 , Issue 1 (April 2005), Pages: 7 - 49, 2005.

[13] Synopsys,
"DesignWare Building Block IP User Guide",
Synospys documentation, SOLD.

[14] Synopsys,
"DW01_add DataSheet",
Synospys documentation, SOLD.

[15] SOLD,
Synospys documentation.

[16] Synopsys,
Online: http://www.synopsys.com/products/solutions/galaxy_platform.html

[17] Cadence,
Online: http://www.cadence.com/lowpower/index.aspx

[18] Magma,
Online: www.magma-da.com/articles/BlastPower.pdf

[19] K.Fukuoka, M.Iijima, K.Hamada, M.Numa, A.Tada,
"Leakage power reduction for clock gating scheme on PD-SOI",
ISCAS '04, Volume: 2, On page(s): II- 613-16, 2004.

[20] S.P.Mohanty, R.Velagapudi, E.Kougianos
"Physical-aware simulated annealing optimization of gate leakage in nanoscale datapath circuits",
DATE, Pages: 1191-1196 , 2006

[21] D.Stasiak, R.Chaudhry, D.Cox, S.Posluszny, J.Warnock, S.Weitzel, D.Wendel, M.Wang,
"Cell Processor Low-Power Design Methodology ",
IEEE Micro November/December 2005 (Vol. 25, No. 6), pp. 71-78.

[22] T.Karnik, Y. Ye, J. Tschanz, L. Wei, S. Burns, V.Govindarajulu, V.De, S.Borkar,
"Total Power Optimization By Simultaneous Dual-Vt Allocation and Device Sizing in High Performance Microprocessors",
39th Design Automation Conference (DAC'02) p. 486-491.

[23] J.Kao, S.Narendra, A.Chandrakasan,
"Subthreshold leakage modeling and reduction techniques",
Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, San Jose, California, Pages: 141 - 148, 2002.

[24] B.M. Al-Hashimi(Editor)
PART III: chapter 12, Power minimisation techniques at the RT-level and below,
"System-on-Chip:Next generation Electronics",
IEE Circuits, Devices and Systems Series 18, 2006, London, United Kingdom.

[25] D.Helms, G.Ehmen, W.Nebel
"Analysis and modeling of subthreshold leakage of RT-components under PTV and state variation",
Proceedings of the 2006 international symposium on Low power electronics and design, Tegernsee, Bavaria, Germany, Pages: 220 - 225, 2006.

[26] H.Su, F.Liu, A.Devgan, E.Acar, S.Nassif
"Full chip leakage estimation considering power supply and temperature variations",
Proceedings of the 2003 international symposium on Low power electronics and design, Seoul, Korea. Pages: 78 - 83, 2003

[27] T.Kuroda et al.
"A 0.9V 150MHz 10mW $4mm2$ 2-D discrete cosine transform core processor with variable-threshold-voltage scheme",
Digest of technical papers of IEEE international solid-state circuits conference, vol 166, 1996.

[28] Laurie Kelly, Piguet Piguet, Christian (EDT) Piguet
"Low-Power Electronics Design",
chapter V: Embedded software
CRC Press,Series: Computer Engineering Series, Volume: 1, 2005,CSEM, Neuchatel, Switzerland.

[29] S.Mutoh, T.Douseki, Y.Matsuya, T.Aoki, S.Shigematsu, J.Yamada
"1-V power supply high-speed digital circuit technology with multitreshld CMOS",
IEEE journal of Solid-state circuits, 1995, Vol.30(8), pp 847-854.

[30] J.T.Kao, A.P.Chandrakasan
"Dual-thresholtvoltage technique for low-power digital circuits",
IEEE journal of Solid-state circuits, 2000, Vol.35, pp. 1009-1018.

[31] S.Bobba, I.N.Hajj
"Maximun leakage power estimation for CMOS circuits",
Proceedings of Alessandro Volta memorial international workshop on Low-power design, Como, Italy, 1999, pp 116-124.

[32] A.Bogliolo, R.Corgnati, E.Macii, M.Poncino
"Parameterized RTL power models for combinational soft macros ",
IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Date: Dec 2001, Volume: 9, Issue: 6, On page(s): 880-887.

[33] Kim, N.S. Austin, T. Baauw, D. Mudge, T. Flautner, K. Hu, J.S. Irwin, M.J. Kandemir, M. Narayanan, V.
"Leakage current: Moore's law meets static power"
IEEE Computer, Volume: 36, Issue: 12, On page(s): 68- 75, 2003.

# Chapter 10

# Conclusion and Future work

## 10.1 Conclusions

Emerging consumer applications demand high performance, low power, low cost and high reliability from the current and the upcoming generations of embedded devices. MPSoCs, i.e. highly integrated, often heterogeneous chips, have a leading role in fulfilling these demands. However, their complexity is becoming so challenging that new techniques and mechanisms to design their on-chip interconnection are greatly needed.

NoCs have enjoyed increasing acceptance as the most scalable answer to these shortcomings. However, as a technology of recent inception, the consensus on their full maturity has not been reached yet. This dissertation proves fully that NoCs are a viable, efficient, scalable, predictable, flexible answer to the on-chip interconnection woes. We have shown a complete design flow that is capable of generating placed&routed instances of a NoC. Our proposed design flow is conceived to optimize design time. Thanks to a detailed modeling activity, by taking into account aspects such as the resulting chip floorplan, and with careful analysis of on-chip wiring, we dramatically increase the design closure chances and thus the need for project re-spins. Our NoC instances have been verified as fully functional. We have also shown some implementations in 65nm technology driving the designer to remove the interconnect bottleneck, and outlined some properties and optimization of these networks, and we also concretely extend the domain of applicability of NoCs, by presenting techniques for increased system fault tolerance and power awareness.

This dissertation provides a broad range of contributions to the scientific community:

- A detailed guideline to a complete design space exploration of network-on-chip architectures, in order to point-out the trade-offs associated with the design of each individual network building block and with the design of network topology overall.
- A complete, top-to-bottom design flow for the implementation of NoCs. This set of tools empowers designers to generate optimized networks for regular topologies given predefined software as well as technology constraints. The design flow includes architectural simulation and physical implementation.

- A set of techniques enabling the use of NoCs in the presence of challenging sub-65nm technology nodes, such as leakage power dissipation, containment of process variations and of their effects.

## 10.2 Future Work

Despite the large body of research devoted to NoCs in the last years, many open challenges remain. We identify several major areas for upcoming research:

- There is a consensus on the system-level need of integrating blocks at different operating frequencies, which is also often called a GLOBALLY ASYNCHRONOUS , LOCALLY SYNCHRONOUS (GALS) paradigm. However, as of today, the dilemma of which NoC alternative to choose in this context has not been fully cleared: synchronous choices (interspersed with a minimum amount of clock converters) have been claiming rapid development times and minimum overhead, while the alternatives claim, for example, superior robustness to process variance. Furthermore, to the best of our knowledge, none of the contending approaches has yet comprehensively tackled the issue of dynamic frequency (and possibly also voltage) scaling; some open questions in this research area include how to best devise mechanisms for NoC partitioning, how to best control the operating parameters of each NoC partition statically (e.g. with CAD tooling for heterogeneous NoCs) or at runtime, etc..

- NoCs for 3D chips are an emerging trend. Moreover, since the manufacturing technology is not fully mature, many crucial design parameters remain unclear(attainable density, yield and speed of vertical interconnects). Depending on these parameters, many different NoC architectures and topologies can be envisioned. Some of the key upcoming challenges in this field include the potential need for pervasive fault tolerance, the design of a new generation of CAD tools for NoC topology exploration, and the issue of clock domain synchronization across stack layers.

- NoC reconfiguration is also a broad topic. While several approaches have been published to reconfiguring NoCs, our impression is that many links are missing before fully reconfigurable, hazard- and deadlock-free, low-resource-overhead NoCs can be available. Furthermore, even then, the larger problem of efficiently deciding how to reconfigure NoCs at runtime, based on mutating application demands, will be very challenging.

## 10.3 Scientific Publication

The work presented in this thesis has led to several publications on the proceedings of international conferences and scientific journals, as listed Hereafter. The list includes a brief description of the contributions made on each paper and it also includes other research work not related to this thesis that the author has also focused on during the development of his PhD thesis.

### 10.3.1 Proceedings

**Title:** Capturing the interaction of the communication, memory and I/O subsystems in memory-centric industrial MPSoC platforms
**Authors:** **Simone Medardoni**, Martino Ruggiero, Davide Bertozzi, Luca Benini,Giovanni Strano, Carlo Pistritto.
**Proceedings:** conference on Design, automation and test in Europe
**Location:** Nice, France.
**Pages:** 660 - 665
**Year of Publication:** 2007
**Description:** Industrial MPSoC platforms exhibit increasing communication needs while not yet reverting to revolutionary solutions such as networks-on-chip. On one hand, the limited scalability of shared busses is being overcome by means of multi-layer communication architectures, which are stressing the role of bridges as key contributors to system performance. On the other hand, technology limitations, data footprint and cost constraints lead to platform instantiations with only few on-chip memory devices and with a global performance bottleneck: the memory controller for access to the off-chip SDRAM memory. The complex interaction among system components and the dependency of macroscopic performance metrics on fine-grain architectural features stress the importance of highly accurate modelling and analysis tools. This paper takes its steps from an extensive modelling effort of a complete industrial MPSoC platform for consumer electronics, including the off-chip memory sub-system. Based on this, relevant design issues concerning the communication, memory and I/O architecture and their interaction are addressed, resulting in guidelines for designers of industry-relevant MPSoCs.

**Title:** Power-Optimal RTL Arithmetic Unit Soft-Macro Selection Strategy for Leakage-Sensitive Technologies
**Authors:** **Simone Medardoni**, Davide Bertozzi, Enrico Macii.
**Proceedings:** international symposium on Low power electronics and design
**Location:** Portland, OR, USA.
**Pages:** 159 - 164
**Year of Publication:** 2007
**Description:** With the advent of nanoscale technologies, developing power efficient ASICs increasingly requires consideration of static power. An effective approach to make RTL synthesis algorithms and tools leakage-aware consists of the smart inference of RTL macros based on design constraints and optimization directives. This involves exploring the new trade-offs spanned by the design of RTL functional units, as an effect of the features of nanoscale technologies and of the power optimizations performed by commercial synthesis tools. This work explores these new trade-offs and proves that making RTL macro selection strategies aware of them results in power savings as high as 43%.

**Title:** Control- and Data-path decoupling in the design of a NoC switch: area, power and performance implications
**Authors:** **Simone Medardoni**, Davide Bertozzi, Luca Benini, Enrico Macii.
**Proceedings:** System-on-Chip, 2007 International Symposium

**Location:** Tampere, Finland.
**Pages:** 1-4
**Year of Publication:** 2007
**Description:** Networks on chip are emerging as a disruptive technology to tackle the problem of scalable on-chip communication. An intensive research effort is being devoted to customizing generic network building blocks for specific design objectives such as low-latency or low-power. In this work, we identify in control and datapath decoupling inside a switch architecture an effective means of achieving the needed flexibility, while taking into account the switching, buffering and flow control implications of each design point. We deploy a 65 nm low-power technology library to explore the performance-power trade-off in the design of a NoC switch with area awareness, while leveraging placement-aware logic synthesis tools to deal with the predictability challenges posed by nanoscale designs.

**Title:** Variation tolerant NoC design by means of self-calibrating links
**Authors:** **Simone Medardoni**, Marcello Lajolo, Davide Bertozzi.
**Proceedings:** conference on Design, automation and test in Europe
**Location:** Munich, Germany.
**Pages:** 1402-1407
**Year of Publication:** 2008
**Description:** We present the implementation and analysis of a variation tolerant version of a switch-to-switch link in a NoC. The goal is to tolerate the effects of process variations on NoC architectures using self-correcting links that automatically detect delay variations and compensate them. The correction is applied without increasing the switch-to-switch latency by substituting the output flip-flops of the sending switch with a self-correcting flip-flop followed by an adaptive voltage swing selector. Higher delay variations will result in a smaller slack in the switch-to-switch path, but the adaptive voltage swing selector could mitigate its impact on the NoC communication by increasing the voltage swing on the link, thus allowing a compensation of the delay variation. As a result, it is possible to tolerate delay variations at the cost of additional power consumption.

**Title:** Network Interface Sharing Techniques for Area Optimized NoC Architectures
**Authors:** Alberto Ferrante, **Simone Medardoni**, Davide Bertozzi.
**Proceedings:** 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools
**Location:** Parma, Italy
**Pages:** 10-17
**Year of Publication:** 2008
**Description:** Abstract-Regular multi-core processors are appearing in the embedded system market as high performance software programmable solutions. The use of regular interconnect fabrics for them allows fast design time, ease of routing, predictability of electrical parameters and good scalability. k-ary n-mesh topologies are candidate solutions for these systems, borrowed from the domain of off-chip interconnection networks. However, the on-chip

integration has to deal with unique challenges at different levels of abstraction. From a technology viewpoint, interconnect reverse scaling causes critical paths to go across global links. Poor interconnect performance might also impact IP core speed depending on the synchronization mechanism at the interface. Finally, this might also conflict with the requirements that communication libraries employed in the MPSoC domain pose on the underlying interconnect fabric. This paper provides a comprehensive overview of these topics, by characterizing physical feasibility of representative k-ary n-mesh topologies and by providing silicon-aware system-level performance figures.

**Title:** Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework
**Authors:** Francisco Gilabert, **Simone Medardoni**, Davide Bertozzi, Luca Benini, Maria Engracia Gomez, Pedro Lopez and Jose Duato.
**Proceedings:** Second ACM/IEEE International Symposium on Networks-on-Chip
**Location:** Newcastle University, UK.
**Pages:** 107-116.
**Year of Publication:** 2008
**Description:** Networks-on-chip (NoCs) address the challenge to provide scalable communication bandwidth to tiled architectures in a power-efficient fashion. The 2-D mesh is currently the most popular regular topology used for on-chip networks in tile-based architectures, because it perfectly matches the 2-D silicon surface and is easy to implement. However, a number of limitations have been proved in the open literature, especially for long distance traffic. Two relevant variants of 2-D meshes are explored in this paper: high-dimensional and concentrated topologies. The novelty of our exploration framework includes the use of fast and accurate transaction level simulation to provide constraints to the physical synthesis flow, which is integrated with standard industrial toolchains for accurate physical implementation. Interestingly, this work illustrates how effectively the compared topologies can handle synchronization-intensive traffic patterns and accounts for chip I/O interfaces.

**Title:** Assessing Fat-Tree Topologies for Regular Network-on-Chip Design under Nanoscale Technology Constraints
**Authors:** D. Ludovici, F. Gilabert, **S. Medardoni**, C. G. Requena, M. E. Gómez, P. López, D. Bertozzi, G. N. Gaydadjiev.
**Proceedings:** conference on Design, automation and test in Europe
**Location:** Nice, France.
**Year of Publication:** 2009
**Description:** Most of past evaluations of fat-trees for on-chip interconnection networks rely on oversimplifying or even irrealistic architecture and traffic pattern assumptions, and very few layout analyses are available to relieve practical feasibility concerns in nanoscale technologies. This work aims at providing an in-depth assessment of physical synthesis efficiency of fat-trees and at extrapolating silicon-aware performance figures to back-annotate in the system-level performance analysis. A 2D mesh is used as a reference

architecture for comparison, and a 65 nm technology is targeted by our study. Finally, in an attempt to mitigate the implementation cost of k-ary n-tree topologies, we also review an alternative unidirectional multi-stage interconnection network which is able to simplify the fat-tree architecture and to minimally impact performance, resulting in a more power-aware fat-tree implementation.

**Title:** Designing Regular Network-on-Chip Topologies under Technology, Architecture and Software Constraints
**Authors:** F.Gilabert, D.Ludovici, **S.Medardoni**, D.Bertozzi, L.Benini, G.N.Gaydadjiev.
**Proceedings:** International Workshop on Multi-Core Computing Systems
**Location:** Fukuoka, Japan
**Year of Publication:** 2009
**Description:** Abstract-Regular multi-core processors are appearing in the embedded system market as high performance software programmable solutions. The use of regular interconnect fabrics for them allows fast design time, ease of routing, predictability of electrical parameters and good scalability. k-ary n-mesh topologies are candidate solutions for these systems, borrowed from the domain of off-chip interconnection networks. However, the on-chip integration has to deal with unique challenges at different levels of abstraction. From a technology viewpoint, interconnect reverse scaling causes critical paths to go across global links. Poor interconnect performance might also impact IP core speed depending on the synchronization mechanism at the interface. Finally, this might also conflict with the requirements that communication libraries employed in the MPSoC domain pose on the underlying interconnect fabric. This paper provides a comprehensive overview of these topics, by characterizing physical feasibility of representative k-ary n-mesh topologies and by providing silicon-aware system-level performance figures.

### 10.3.2 Magazine

**Title:** An Efficient Implementation of Distributed Routing Algorithms for NoCs
**Authors:** S.Rodrigo, **S.Medardoni**, J.Flich, D.Bertozzi, J.Duato.
**Magazine:** to appear in IET Computers and Digital Techniques Journal.
**Description:** Chip multiprocessors (CMPs) are gaining momentum in the high-performance computing domain. Networks-on-chip (NoCs) are key components of CMP architectures, in that they have to deal with the communication scalability challenge while meeting tight power, area and latency constraints. 2D mesh topologies are usually preferred by designers of general purpose NoCs. However, manufacturing faults may break their regularity. Moreover, resource management frameworks may require the segmentation of the network into irregular regions. Under these conditions, efficient routing becomes a challenge. Although the use of routing tables at switches is flexible, it does not scale in terms of latency and area due to its memory requirements. LBDR (Logic-Based Distributed Routing) is proposed as a new routing method that removes the need for routing tables at all. LBDR enables the implementation of many rout-

ing algorithms on most of the practical topologies we may find in the near future in a multi-core system. From an initial topology and routing algorithm, a set of three bits per switch/output port is computed. Evaluation results show that, by using a small logic, LBDR mimics the performance of routing algorithms when implemented with routing tables, both in regular and irregular topologies. This paper also explores LBDR implementation in a real NoC switch, proving its smooth integration in the architecture and its negligible hardware and performance overhead.

### 10.3.3 Poster

**Title:** Performance, area and power breakdown analysis for NoC switches in 65nm technology
**Authors:** **S.Medardoni**, D.Bertozzi.
**Proceedings:** Acaces.
**Year of Publication:** 2007

**Title:** Assessing the Implementation Trade-offs of Logic-Based Distributed Routing for NoCs
**Authors:** S.Rodrigo, **S.Medardoni**, J.Flich, J.Duato, D.Bertozzi.
**Proceedings:** Acaces.
**Year of Publication:** 2008

### 10.3.4 Patent

**Title:** Variation Tolerant Network on Chip (NoC) system with self-calibrating links.
**Authors:** **Simone Medardoni**, Marcello Lajolo.

# List of Tables

# List of Figures