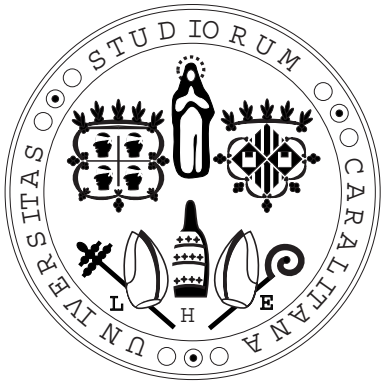
UNIVERSITÀ DEGLI STUDI DI CAGLIARI
FACOLTÀ DI INGEGNERIA
Dottorato di Ricerca in
Ingegneria Elettronica e Informatica
(Anno Accademico 2007-2008 - XX ciclo)

Design and optimization techniques for VLSI Network on Chip architectures

A novel complete design flow for application specific NoCs



PhD thesis by:
Paolo Meloni

Contents

Introduction **ix**

1 Context and Motivation **15**
1.1 Introduction 15
1.2 SoC paradigm 15
1.3 MPSoC paradigm 16
1.4 Challenges of Deep Sub-Micron Technologies 17
1.4.1 Complexity of the interconnect hierarchy 18
1.4.2 Interconnect delay 18
1.4.3 Energy consumption 19
1.4.4 Interconnect reliability 20
1.4.5 Process variations 20
1.4.6 Complexity of interconnect modeling 20
1.5 Interconnect architecture optimization: motivation 21
1.5.1 Network-on-chip communication architectures 22
1.6 Network-on-chip architectures features and classification . . . 23
1.6.1 Network design constraints 24
1.7 Network-on-chip architecture design space 26
1.7.1 Fundamental network parameters 26
1.7.2 Network Design Decisions 31
1.8 State of the Art NoC Architectures 33
1.8.1 AMBA Shared Bus 36
1.8.2 AMBA Multi-Layer 37
1.8.3 ×pipes NoC 37

2 Comparative analysis of NoCs and Traditional Interconnects **41**
2.1 Related Work 42
2.2 The Fabrics Under Test 43
2.3 The Test Applications 45
2.4 Reference Characterization Flow 48

2.4.1 Fabric Simulation 48
2.4.2 Fabric Synthesis 49
2.5 Performance comparison results 53
2.5.1 Interconnect Performance 53
2.5.2 Interconnect Area, Frequency of Operation and Bandwidth 56
2.5.3 Interconnect Power and Energy 58
2.5.4 Split Analysis of Area and Power Contributions 61

3 Designing Application-Specific Networks on Chips 69
3.1 Introduction 69
3.2 Design Flow 71
3.3 Input Models 75
3.4 Design Algorithms 76
3.5 Experiments and Case Studies 83
3.5.1 Layout-level Comparisons 83
3.5.2 Experiments on SoC Benchmarks 85

4 Area and Power Modeling for Networks-on-Chip components 91
4.1 The ×pipes Switch Architecture 94
4.2 Proposed Modeling Methodology 94
4.2.1 Parameters of Interest 96
4.2.2 Area and Power Models 98
4.2.3 Choice of a Relevant Training Set 103
4.2.4 Fitting Model Coefficients 104
4.3 Experimental Results 106
4.3.1 Experiments with Netlist-Based Models and a Netlist-Level Test Set 107
4.3.2 Test Case: a Complete NoC Topology 108
4.3.3 Experiments with Netlist-Based Models and a Layout-Level Test Set 108
4.3.4 Experiments with Layout-Based Models and a Layout-Level Test Set 111
4.3.5 Experiments with a Parabolic Model for the Dependency on the Target Synthesis Frequency 113

5 Routing Aware Switch Hardware Customization 117
5.1 Introduction 117
5.2 Reference design flow 118
5.3 Routing aware hardware optimization 119
5.3.1 Hardware-Level Customization Support 121

5.3.2 Software-Level Customization Support 121
5.4 Customization method effectiveness evaluation 122
5.4.1 Experiments on the Multimedia benchmark 122
5.4.2 Experiments on SoC benchmarks 124

6 65 nm NoC Design **129**
6.1 NoC Design Flow 130
6.1.1 Flow Back-End 131
6.1.2 Post-Layout Analysis 135
6.2 Wire Design in 65 nm Technologies 135
6.2.1 Link Delay and Link Power 135
6.3 Experimental Results 136
6.3.1 Technology Scaling from 90 to 65 nm 136
6.3.2 Topology design 140
6.3.3 High Bandwidth Application 140
6.3.4 Effect of Link Pipelining 142
6.3.5 Low Bandwidth Application 143

7 Conclusions **149**

List of figures **155**

List of tables **157**

Introduction

As steady progress is being made in the miniaturization of chip features, embedded systems are quickly evolving towards complex devices, including a large set of computation engines, dedicated accelerators, input/output controllers and multiple memory buffers. *MultiProcessor System-on-Chip (MPSoC)* is a commonly used term to describe the resulting outcome. However, this feature- and performance-oriented evolution is not devoid of significant challenges, including mastering the increasing design complexity and minimizing power consumption. Moreover, miniaturization itself is bringing its own set of design issues at the physical level, originated primarily by an increasing ratio of wire *vs.* logic propagation delay. One of the most critical areas of MPSoC design is the interconnect subsystem, due to architectural and physical scalability concerns. The former is due to the performance pressure associated with several system cores that simultaneously demand communication resources, and subsequently relates to the need for providing adequate bandwidth and latency. The latter is due to the intrinsic issues due to the design of wires that must span across the whole chip area, namely, propagation delay, noise and rosstalk. Traditional shared bus interconnects are relatively easy to design, but do not scale well. Thus, evolutions have been conceived both from the protocol (*e.g.* outstanding transactions with out-of-order delivery) and the topology (*e.g.* bridges, crossbars) points of view. Nevertheless, scalability is still suboptimal, as protocol improvements still hit a bandwidth limit due to the available physical resources, and topological extensions require the use of bridges (*i.e.* multiple buses or "spaghetti-like" design) or large area overheads in routing structures (*i.e.* using crossbars). *Networks-on-Chip (NoCs)* have been suggested as a promising solution to the scalability problem. By bringing packet-based communication paradigms to the on-chip domain, NoCs address many of the issues of interconnect fabric design. Wire lengths can be controlled by matching network topology with physical constraints and bandwidth can be boosted by increasing the number of links and switches. This thesis focuses on the design and the optimization of Network on Chip architectures paying special attention to the aspects re-

lated to the relationship between the system level design decisions and the back-end implementation variables. In particular, the core of the presented research work is a complete flow for application-specific NoC design. The proposed flow helps the NoC designer to perform all the design actions required, from the target application task graph to the placement&routing steps. The activities concerning this research have been performed in collaboration with three external partners:

- Università di Bologna
- Stanford University
- Ecole Polytechnique Fédérale de Lausanne

As will be explained more in detail in the thesis, the proposed flow aims to tailor the optimal NoC structure for a given application. The flow consists of a front-end part, devoted to the synthesis of the optimal NoC configuration based on the application communication requirements, of a back-end part, in charge of the implementation (down to the layout level) of the configuration chosen by the front-end, and of some side steps needed to produce the information received in input by the previously mentioned two parts and to close the gaps between them. In the first chapter, the motivation bringing to approach to this research field is dissussed. In this aim, an overview of the landscape of todays multicore systems implemented on a single chip is included, to explain why, from a functional point of view, modern computing devices would require a new interconnection architecture paradigm. Moreover, the problems related to the physical aspects, such as scalability, design feasibility and reliability, that emerge with the technology shrinking and pose the need for a more scalable and structured communication infrastructure, are discussed. Once this context is illustrated, some exemples of state of the art interconnection systems are presented. A brief summary about most efficient and diffused bus-bases systems and about the best known on-chip networks is reported. ×pipes , an in-house developed Network on Chip architecture, conceived in 2002 and so far defined in a completely reconfigurable and synthesizable RTL component library, is depicted more in detail, being the reference architecture for the whole research work and for the whole presented design flow. In the second chapter, to assess the usefulness of NoC architectures, and to figure out the advantages related to them, a thorough comparison between bus-based systems (simple and complex) and on-chip network is presented. A great number of design variables affect the result of such a comparison. However, to make the analysis meaningful, there are aspects which cannot be ignored. As mentioned above, it is not possible to just

simulate each architecture to assess its effectiveness. Even if performance alone is taken into account, the clock frequency variable can skew results. Further, since interconnects by definition span across significant portions of the die, wiring congestion and wire propagation delays are very difficult to estimate in advance. Therefore, the overhead of crossbars and NoCs can only be fully understood after having mapped the architecture onto a chip layout. A shared-bus, a crossbar and a NoC designs were brought to the chip layout level in order to highlight the respective strengths and weaknesses in terms of performance, area and power, keeping an eye on future scalability. Inside this chapter, the back-end part of the proposed NoC design flow is introduced and illustrated in detail. Moreover, it is used to take the test platforms down to a placed&routed layout, in order to get frequency, area and power figures. A 0.13 μm library was used for this study. In the third chapter, the whole flow for application-specific NoC design is presented. A preliminary description of the reasons pushing to the approach to application-driven design is given, then the different steps of the proposed flow are commented in detail. In particular, the front-end part related to the high-level topology synthesis is illustrated as reference and how its interfacing with the back-end part of the flow was allowed is explained. Moreover, the chapter includes the comment related to a set of experiments proving the usefulness of the proposed approach and the consistence of the used design methods. In the fourth chapter, one of the most important side steps included in the flow is presented. This step is represented by the construction of detailed area and power models, needed by the front-end of the flow to evaluate the impact of each decision taken with respect to the chosen network configuration. The models are derived by using information and experimental results coming from the back-end part of the flow, and are thus mandatory to close the gap between the configuration synthesis and the actual results after implementation. A complete methodology extendible to ×pipes as well as to other NoC architectures is presented and tested to asses accuracy and effort required. In the fifth chapter, an original design implementation technique, aimed to reduce the hardware overhead introduced by NoC components is briefly explained. Basically, according to this technique the internal hardware of the ×pipes NoC router is specifically reduced to the minimum configuration needed to suit the traffic required by a given application. The proposed technique is completely integrated in the ×pipes design flow. Finally, in the sixth chapter, the NoC design is investigated referring to Deep Sub Micron techology. Athough the scalability and predictability improvements, derived by the new communication paradigm, could already be assessed by evaluating the experiments commented in Chapter 1, further experiments and implementation trials have been performed referring to smaller technologies,

namely 90 *nm* and 65 *nm*, to assess actual NoC feasibility and efficiency in challenging technology nodes. Moreover the evaluation of real impact of wiring capacitance, was tackled thoroughly shedding light on a major important point in NoC research. Updates to the proposed design flow were found out to be mandatory in such a contest, allowing to define some very useful guidelines for the NoC designer.

Chapter 1

Context and Motivation

1.1 Introduction

Modern trends in electronics and challenges related to the physical features of future technology processes, bring the need for a scalable and efficient interconnect structure, to support the interaction between the units inside a chip. SoC and MPSoC paradigms, every day more widely diffused, pose strict requirements in terms of functional scalability and reusability of the modules. Moreover, manufacturing with deep sub-micron (DSM) technologies, the designer can not keep up with physical scalability and reliability using classic interconnect structures. So, to introduce the thesis, in this chapter, an overview of the concepts related to modern design paradigms [10] and a brief summary of the physical problems exposed by the technology scaling are reported.

1.2 SoC paradigm

By the end of the decade, according to the International Technology Roadmap for Semiconductors [1], it will be possible to realize inside a single silicon die up to 4 billion transistors smaller than 50-nm, operating below one volt and running at 10 GHz. On the other hand, today's market pushes the hardware designers asking for the possibility to run very complex applications, showing the needs for computing devices providing every day higher performances. Network processors used in high-performance routers, handsheld devices merging telephony and multimedia capabilities, digital televisions and set-top boxes, video game stations rendering gaming action in real time, sensor networks, servers, are typical modern devices needed to support performance-hungry applications. A System-on-chip is the result of the inte-

gration of complete complex systems on a single chip. Systems-on-chip use the huge integration capabilities provided by modern technology nodes, mixing in the chip digital, analog, mixed-signal and radio-frequency functions, to find a solution to the challenging design problems in the telecommunications, multimedia and consumer electronics domains. The typical SoC hardware implementation is based on a platform obtained by the composition of a set of independent tiles. A tile is defined as an independent subsystem of the System on Chip architecture that can accomplish a high-level function. It can combine storage, computation and communication interfaces with the system environment. It can be a processing unit (including its cache memory), like a General Purpose Processor (GPP) or a Digital Signal Processor (DSP), a Coarse Grain Architecture (CGA), an Application Specific Integrated Circuit (ASIC), a RAM blocks or an input/output interface. Tiles are also known as Intellectual Property cores (IP cores), typically when designed by another party and later integrated into a SoC platform. Major IP vendors like ARM license their IP cores to platform integrators. The IP cores can be provided in soft form (soft IPs), i.e. their description in a HDL language or a standard-cell netlist is delivered, or in hard form (hard IPs), meaning that a layout level black box is released to be integrated inside the SoC flooplan. Hard IPs can have various sizes and aspect ratios. The possibility to easily integrate inside a platform third party macros is mandatory to take profit about the advantages coming from the improved computing potential, since time-to-market pressure is every day stronger and a complete full-custom design flow of the whole platform is unfeasible.

1.3 MPSoC paradigm

Most current SoCs are Multi-Processors SoCs (MPSoCs). They contain multiple instruction-set processors (CPUs), and are seen as the only way to meet future system feature set, design cost, power, and performance requirements. State of the art MPSoCs are processor arrays, instantiating multiple parallel general-purpose processors (GPPs) and multiple application-specific processors (ASPs). They allow to exploit the intrinsic parallelism that is usually present in modern application domains, outperforming in this cases single high-end general purpose processor. Moreover, integrating different processors inside a chip, allows the cores to work at a lower frequency and provide balance between performance and power consumption showing much lower power consumption per core. It is also possible for the designer to specify for each core a different specialized instruction set, bringing the system to a better efficiency and to higher performances. A very useful possibility of-

fered by the MPSoC paradigm is related to accelerating the design process. An MPSoC platform can be easily upgraded or customized for different customer's needs relying on the reusability of the tile or on the integration of third party IPs. This allows to avoid the optimization process for a full-custom designed high end processor, that usually requires extremely long design cycles. MPSoC platform based design flow reduces the gap between the design productivity and the potential offered by future silicon technologies and thus permit to better underly to the time-to-market tremendous pressure. Typical examples of high performance MPSoC are:

- the Cell processor platform designed by Sony, Toshiba and IBM, that is based on one general purpose processor (IBM Power PC) and 8 graphical co-processors (see figure 1.4).The platform will be used for the Playstation 3 game console, high definition television sets and computer servers.
- Niagara 2 platform designed by Sun, including 8-core (look for figure)
- Barcelona designed by AMD, including 4 cores (look for figure)
- DSP PC205 by picoChip, including 1 GPP core and 248 individual ASPs (look for figure)
- Cisco Silicon Packet Processor, including 188 programmable 32-bit RISC processors executing 47 billion instructions per second (BIPS) (look for figure)
- Intel Network Processor, including 1 GPP Core and 16 ASPs (look for figure)

Probably the most advanced research project in the field of high performances MPSoC is Intel's TeraFlop chip including 80 GPPs, interconnected by a mesh-like network-on-chip.

1.4 Challenges of Deep Sub-Micron Technologies

Transistor size is constantly reduced in every transition to a new technology node. The minimum feature size, that is the minimal distance separating two wires, can be considered as a common unit for measuring transistor size relative to a given technology. As it became smaller than 90 nm, microelectronics has entered a new era of design challenges called the Deep Sub-Micron

era, mainly characterized by the fact that communication will become more critical than computation. Interconnect will become the dominating factor determining speed, noise and power. Compared to transistors, interconnect has evolved very slowly since CMOS technology has been introduced. In the past thirty years, the wire delay has been reduced by a factor 60 while during the same time for transistors, a factor 1000 has been reached. While, so far, interconnect has never been a critical issue as transistors were dominating the delay [8], in the coming years, interconnect will have to dramatically improve if microelectronics industry wants to keep up following Moore's law [11].

Deep Sub-Micron technologies come up with many challenges mainly affecting the global wire interconnect:

- increasing complexity of the wiring hierarchy
- increasing interconnect delay
- increasing energy consumption
- decreasing interconnect reliability
- increasing process variations
- increasing complexity of the interconnect modeling

1.4.1 Complexity of the interconnect hierarchy

The number of metallization levels costantly grows in future DSM designs as interconnect is becoming more and more heterogeneous. The hierarchy complexity must bu managed coherently on the physical plane, considering the distinction between local, intermediate and global interconnect, and on the logical plane, using different solutions to tackle local on-chip communication-architecture, targeting intra-tile communications, or global on-chip communication architecture, targeting inter-tile communications.

1.4.2 Interconnect delay

As technology scales down, local and intermediate wires become shorter in average, leading, together with the introduction of new materials, to a dramatic improvement in local and intermediate wire delay. However, many phenomena are deteriorating wire RC delay improvement, affecting the wire resistance (skin effect, effective resistivity, inelastic scattering at the boundaries, process variations...) or the wire capacitance (cross-talk, fringing capacitance,...). Transistor gates become smaller, leading to lower transistor

energy consumption and delay. The ratio between gate delay and local wire delay remains about the same. While local wire lengths scale with the technology, global do not. The length of the longest global wires remains about the same as technology scales or could even increase as silicon dies could become larger. So, the relative contribution of the global interconnect to the power consumption and delay increases considerably (see figure 1.1 [1]).

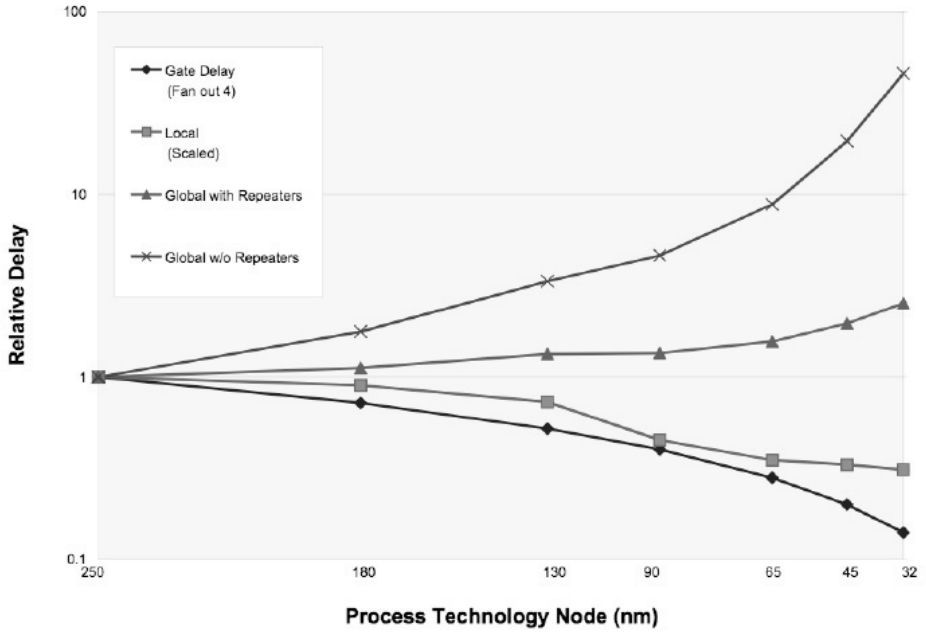


Figure 1.1: Comparison between gate delay and local/global interconnect delay:local wires and gate delay are scaling down while the relative contribution of the global wire delay is increasing with technology

1.4.3 Energy consumption

Energy consumption is a critical issue for hand-held devices as battery life is very limited and heat dissipation systems are already reaching their limits. The power consumption of a CMOS circuit has two components: dynamic and static. Dynamic power consumption occurs when a gate is switching from one state to another. Static power consumption is due to the existence of small leakage currents that flow through transistors in cut-off mode. As the industry is moving to Deep Sub Micron technologies, leakage power consumption of CMOS circuits starts to can not be neglected [9]. Leakage current of one transistors is indeed negligible in absolute but as transistors density

is reaching billions of transistors per chip, sub-threshold leakage becomes a major problem, also taking into account its temperature dependence.

1.4.4 Interconnect reliability

In Deep Sub-Micron technologies, an interconnect wire cannot be considered as an ideal transmission medium anymore. Signal reliability is affected by inter-wire noise, due especially to the capacitive crosstalk that arises from the reduction of the insulator thickness between wires and the increasing of their aspect ratio, the mutual coupling capacitance increases considerably. Moreover, noise margin is decreased due to power supply and threshold voltage scaling. Crosstalk implies delay degradation and unpredictability. On the other hand wire reliability is deeply affected by electromigration, that occurs when the metal ions in a wire are repeatedly impacted by electrons and thus transported. It causes open-circuit in the wire or short circuit between wires.

1.4.5 Process variations

As minimum feature size becomes smaller than 100nm, systematic or random process differences can arise between dies (inter-die variations) or within one die (intra-die variations). While systematic process variations can be handled by designers, random process variations remain a critical research issue at present time. Process variations may lead to manufacturing yield problems or to time-dependent variations that only appear during the chip operating time and they can either affect the functionality of the system component or only its characteristics. Examples of causes of variations are:

- Variant dopant concentration: this implies voltage threshold variations;
- Limited resolution of lithography: this causes transistor length variations resulting in leakage current variation;
- Line edge roughness problems: edges of the wires are more and more difficult to control as technology scales;
- EDA tool inaccuracies: the technological models integrated in the EDA tools have limited accuracy which introduces variability on system characteristics;

1.4.6 Complexity of interconnect modeling

To evaluate delay and power consumption, interconnect used to be modeled as simple RC lumped circuits. However, in Deep Sub-Micron technologies

and especially for long lines like global wires, line resistance will not be negligible anymore compared to driver resistance. Thus, modeling the interconnect will become more complex as models will have to take into account the distributive character of the interconnect. Inductance contributions also have to be taken into account for high performance designs. For very long high performance interconnect (clocked at 1GHz), only transmission line models give accurate results.

1.5 Interconnect architecture optimization: motivation

Interconnect is becoming critical in SoC designs due to both system-level and technological constraints. Major factors thus motivate the need for an optimized communication architecture:

- the increasing density of on-chip components which makes past interconnect solutions inefficient
- the need for communication architecture reusability to cope with platform-based design methodology
- the need to tackle problems imposed by DSM technologies effects

SoC design is now moving from computation to communication centric [2], [7]. Communication and data-access are indeed becoming the two most critical issues for future SoC designs. Traditionally, SoC designers have used either ad-hoc wiring or simple shared bus architectures as the global communication architectures. Those solutions were well adapted for small SoC platforms for which interconnect was far less important than computing architectures. However, as the system complexity grows and the wiring delay surpasses the gate delay, those communication architecture, under the load of multiple high-speed processing elements, rapidly become a bottleneck. Designers have been successfully tackling the issue by incrementally improving the bus paradigm. Approaches range from more advanced protocols [17], which keep the shared bus topology but add sophisticated features, like the support for multiple outstanding transactions, to new topological concepts [4, 14, 16], up to the extreme concept of a full crossbar interconnect. These alternatives exhibit varying levels of complexity, performance and reusability. Still, the scalability challenge is not over. A crossbar-based design guarantees maximum bandwidth, but large crossbars run into *spaghetti wiring* issues, which hinder the achievable frequency of operation and pose severe physical design

problems. Further, large crossbars are expensive to implement. Therefore, for big designs, hierarchical approaches are typically taken; IP cores are glued together by a mixture of buses, crossbars, bridges, decoders and adapters. This choice keeps hardware cost at an acceptable level, but is difficult to design, validate, maintain and extend; and most importantly, it does not really provide a comprehensive answer to the scalability dilemma. In search of a proven solution to scalability worries, researchers turned to wide area networks to get inspiration.

1.5.1 Network-on-chip communication architectures

Networks-on-Chip (NoCs) are the target of the design and optimization techinques proposed in this thesis. NoCs were proposed as a scalable solutions to the problems related to the utilization of shared structures in complex SoCs [5, 6]. NoCs (Fig. 1.2) are the on-chip transposition of the packet-switched paradigm; as such, they feature some known properties, but a mostly unexplored set of design tradeoffs. For example, in wide area networks, switches can leverage a whole dedicated chip and a large amount of buffer memory. Within a chip, a switch has to fit within fractions of mm^2 and its power consumption should be some milliwatts at most. The most effective way to implement a switch while staying within those bounds has not been clearly determined yet. Similarly, wide area networks can afford processing latencies of hundreds of milliseconds without problems, while in a NoC an excessive latency overhead for packetization and multi-hop routing could severely affect area, power and latency metrics.

Provided that designers can comply with such tight constraints, NoCs feature many compelling properties. Should system requirements increase, more bandwidth can be easily added to a network. Physical design is made easier by the possibility of optimizing wire utilization and enforcing strict wire segmentation. Designing complex interconnection systems becomes a more streamlined activity, since a single homogeneous architecture must be deployed and validated. At the same time, very heterogeneous IP cores (in terms of data width, operating frequency, transaction types) can be plugged to a NoC, given the proper network adapters. The NoC itself can be shaped into arbitrary topologies to optimally match the communication needs of the cores; this is key to simultaneously satisfying application requirements and cost constraints. Generally speaking, NoCs feature more degrees of freedom than alternative architectures, which is a definite advantage in the highly heterogeneous SoC market. As can be seen, each design alternative for the interconnect fabric is very suitable for a subset of the SoC application space, but is limited in some other respect. Buses are cheap to manufacture, but

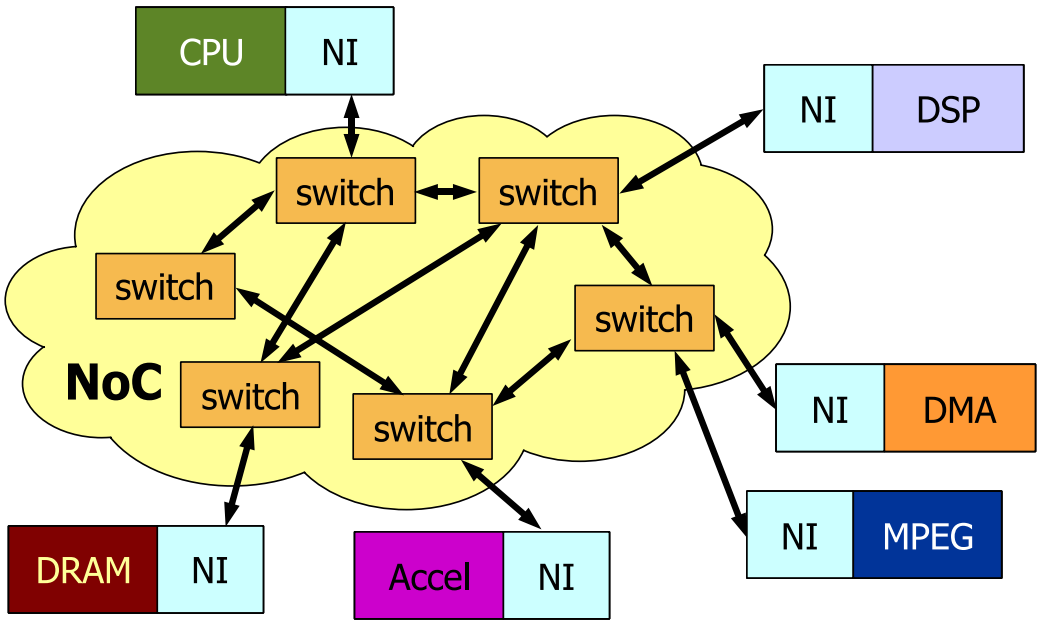


Figure 1.2: A general view of a NoC

have a clear performance scalability issue. Crossbars have the highest possible performance for medium-sized designs, however cost and wire routing issues also prevent scalability beyond a certain threshold. Hierarchical buses and mixed topologies struggle to provide the best tradeoff among performance and overheads in big designs, but also encounter a stumbling block in terms of complexity of design and validation. NoCs offer virtually unlimited scalability and a high potential for customization, but their implementation overhead has to be fully assessed first. The activities presented in this thesis aim to validate this claim with exhaustive design experiments.

1.6 Network-on-chip architectures features and classification

The efficiency of an interconnect architecture is evaluated taking into account its capability of supporting the system feasibility and communication requirements using some corresponding metrics.

A communication process is an information (data) transfer from an initiator unit (or masters) and one target unit (or slaves). The transfer can be "pushed" if the sending unit is an initiator or "pulled" when it is required by a target. The "protocol" specifies how data should be transfered from senders

to receivers. Some control information must also be provided to initiate a transfer, including for exemple an identifier of the destination. The "communication architecture" is devoted to implements the communication process. Every communication architecture, and thus every NoC, basically consists in:

- links: A "link" is a logical connection between two (or more) communication architecture components, is composed out of one or more physical communication channels supporting the signal carrying the information. Typically, on-chip networks communication channels, are physically implemented as global wires in current CMOS technologies.

- switching elements: A switching element is a component that connects its input ports with its output ports in a flexible manner, performing or not a connection between two links connected to it. Physically, while it is typically a multiplexer in a bus or a switching fabric in a crossbar based design, it consists in the switch of router in a Network-on-Chip.

- buffers: inside the communication architecture some storage resources are used to assure loss less communication and reduce cogestion.

Moreover, communication architectures in general, but especially NoCs, can require a *Network Interface*, that is a module devoted to perform a protocol conversion, from the protocol supported by the external environment, to the one featured by the communication infrastructure. In case of packet switched NoCs, the Network Interface (NI) is also in charge of packeting operations.

1.6.1 Network design constraints

Communication architecture design is performed taking into account several requirements posed by the system-level constraints:

- Connectivity requirements:
  For a given application, the traffic pattern exposes the communication requirements between the different tasks. Any application can be described as a Control and Data Flow Graph (CDFG). The nodes of the graph represent the tasks, the edges represent the control and data dependencies. A control dependency implies that a task has to be completed before the next one can start. A data dependency represents data transfers between tasks. The application mapping can thus be viewed as the mapping of the CDFG on an architecture.

- Performance requirements:
  can be evaluated using several metrics: the *end-to-end network latency*, defined as the delay between the injection of a data in the network at the source and its reception at the destination, that is only related to the network feaures; the *end-to-end communication architecture latency*, defined as the delay between the time the initiator starts its message transmission and the time the target receives the message, that includes the network interfaces latency; the *set-up time*, defined as the delay between the time at which control information is sent and the time at which data can be injected in the network; *the throughput (or aggregated bandwidth)*, defined as the maximum amount of information per unit of time that the network can physically handle; *the bisection bandwidth*, defined as the sum of the bandwidths corresponding to the links intercepted by a cut which spans the least possible amount of edges (i.e. a min-cut) and partitions the network in two equal size parts.

- Performance guarantees:
  Communication architecture performances are generally not predictable due to the contention when network resources can be shared among different users. However, real-time applications may have to meet strict temporal deadlines or require a minimum amount of bandwidth to operate properly. Multimedia applications generally contains many real-time data streams of different latency and bandwidth requirements.

- Scalability:
  The scalability of a network is its aptitude to scale efficiently performance parameters and physical parameters in function of the number of communicating devices. In particular, the network aggregated bandwidth should increase proportionally when additional communicating devices are added to the network. The bandwidth per node can thus be preserved when additional nodes are connected.

- Physical constraints:
  Physical constraints include wiring length, area overhead introduced by the communication architecture and energy and power consumption. Energy consumption should be limited to increase the battery lifetime, while power consumption should be minimized to restrain the chip temperature because advanced packaging and cooling systems are unaffordable for cheap embedded systems.

- Reliability and yield:

Besides providing performance guarantees during normal cases, a communication architecture will also have to provide mechanisms to tolerate exceptional circumstances that can alter network behavior. Basic metrics to mesure reliability are:

- Availability: measure of the probability that the network will remain operational
- Probability of failure on demand: measure of the probability of the network to behave in an unexpected way for a given request
- Rate of failure occurrence: measure of the network failure frequency on a given duration
- Mean Time To Failure (MTTF): measure of the mean time during which the network remains operational before a failure occurs.

A global network design cost function can then be built based on those soft and hard constraints. The design of the network will thus result from a trade-off between the satisfaction of those different parameters.

1.7 Network-on-chip architecture design space

The Network-on-chip communication architecture design space is extremely vast. Many network parameters can be set to tailor an optimal on-chip architecture. This section gives a brief summary of the principal network decisions.

1.7.1 Fundamental network parameters

Topology

The network topology defines the logical connections between switching elements and between the switching elements and the nodes. It can be represented by a graph G(C,N), the graph edges C being the connections between nodes and the vertices N being the nodes and switching elements of the network. The network topology is one of the most critical decision in on-chip communication architecture. It has a considerable impact on the network design costs. Topologies can be classified according to their features:

- direct-indirect
  In direct networks, each switching device (e.g. router, switch) is connected to at least one communicating device (e.g. IP core). In indirect networks, a switching device is not necessarily connected to a

communicating device. It can be connected to other switching devices. Switching devices are connected with each other by point-to-point links. Hybrid networks is a mix of those two types of network.

- orthogonality
  Most direct networks have an orthogonal topology. It means that nodes can be arranged in an orthogonal n-dimension space and that every link can be arranged in such a way that it provides a displacement in a single dimension. Strictly orthogonal topologies is a subset of orthogonal topologies for which every node has at least one link crossing each dimension.

- diameter
  The diameter of a network topology is the maximum number of edges between a pair of vertices in the topology. The diameter can be linear, logarithmic or fixed line lengtht, depending on how diameter progresses increasing the number of nodes interconnected in the network.

- regularity
  Topologies are said to be regular when all the vertices of the graph have the same number of edges connected to each vertex.

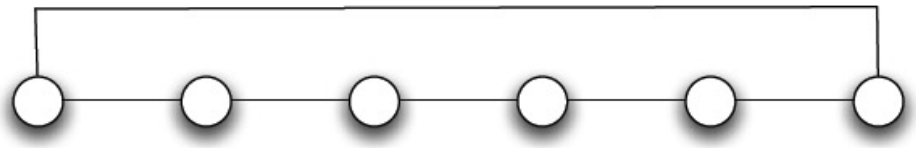
- general purpose-domain specific General purpose topologies are not optimized for a specific pattern. Examples are typically the mesh topology and k-ary n-cube networks in general. Traffic pattern specific topologies examples exploit traffic pattern knowledge, facilitate communication locality. An example of regular topology that is not general purpose is given by the fat-tree topology (see figure 1.3. This topology is regular but it exploits locality of traffic. On the other hand, it performs very badly with general purpose traffic as the root nodes will become a bottleneck.

Examples of topologies are shown in figure 1.3

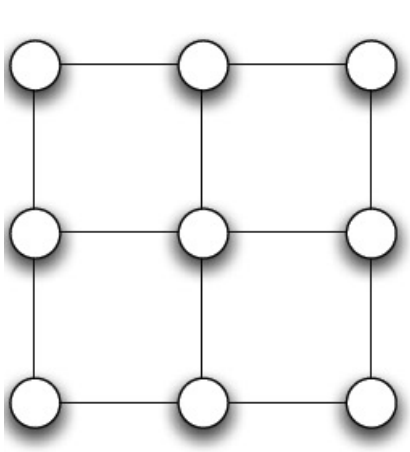
Routing Technique

The routing technique decision consists in defining and implementing a strategy to choose one path from source to destination nodes when the topology offer multiple paths. In shared-buses, no routing technique is needed, since source and destination directly share the same medium. For hierarchical buses and bridged buses, there is only one path is available from source to destination. It is only needed to specify which bridges have to be switched. Routing algorithms can be classified according to several criteria:

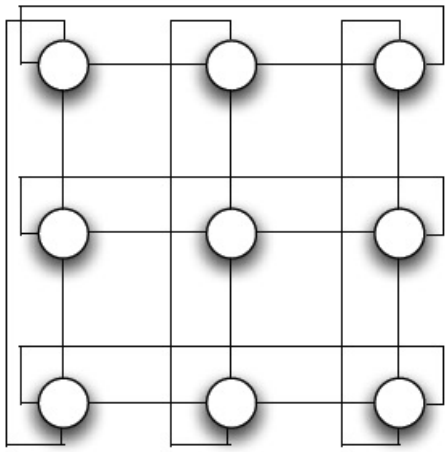
(a) Shared-Bus



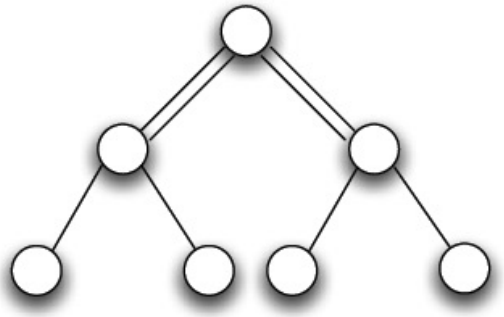
(b) Ring



(c) 2D Mesh (3-ary 2-cube)



(d) 2D Folded Torus



(e) Fat Tree

Figure 1.3: Examples of common network topologies

- Number of destinations
  If a message can be routed to multiple destinations at the same time, the routing technique will be classified as multicast (or broadcast for all possible destinations). If only one destination can be reached, it is said to be unicast. This choice mainly depends on the application requirements.

- Routing decision time
  Routing decisions can be performed exclusively at run-time, exclusively at design-time or partly at design-time, partly at run-time.

- Routing decision locality
  Locality of the routing decision: central/distributed/hybrid

- Locality of the routing technique implementation
  The routing decisions can be performed by a central control unit (centralized routing) running one or several routine or along the network (distributed routing). Hybrid schemes are also possible. The particular case in which a centralized routing implementation is located at the source network interface is commonly referred to as source routing.

- Locality of the network status information
  Information on the network status can be obtained at local ,global or semi-global level, or can not be taken into account at all (oblivious routing).

- Routing adaptivity
  The routing strategy can be deterministic (fast and well adapted to be implemented in hardware), meaning that the routing decision is invariant for a given source and destination, or it can be adaptive, providing different routes depending on the network load (offering better performance and fault-tolerance).

- Number of paths
  The routing algorithm can either consider all the paths offered by the topology from source to destination (complete routing algorithm) or only a sub-set of available paths (partial routing algorithm).

- Parallelization
  The routing algorithm can be implemented by one sequential process or by multiple parallel processes executed on different processing cores.

- Implementation
  The routing decisions can be implemented in hardware(faster), software(more flexible) or in a combination of both.

Routing granularity

The routing granularity decision consists in defining at which data granularity the routing decision will be performed: circuit-based routing or packetbased routing. For a *circuit-based routing* technique, the routing decision is performed once for all at the establishment of the circuit. All the data will then transit through the same path offering predictable Quality of Service to that connection, once an extra-delay required at the circuit set-up time is payed. For a packet-based routing technique, the routing decision can be performed at the granularity of a packet. A packet is typically composed of a header containing routing informations and a payload which contains the data. Packet-size is an important parameter for the network. In the packet-based routing technique there is no circuit set-up delay and bandwidth utilization is closer to the optimum, no guarantees on available bandwidth and latency can be provided.

Flow Control Mechanism

The flow control defines the policy used to adjust the data flow between communicating devices. Data sending, generally, requires multiple transmissions since links bit-width is limited. The flow control can be performed at several levels and it defines the granularity at which data are sent through communication devices. Flow/congestion control strategies handles the transmission of data from a communication device input port to an output port. The communication device can be a switch or a bus.

Switching Technique

When applied to networks, different combinations of paramenters such flow control granularity, or buffering placement corresponds to well-known switching techniques such as:

- real circuit switching
  Real circuit switching consists of reserving a path across the network from source to destination, typically sending an header over the network from the source node. The header reserves the channels along the path. When the probe arrives at the destination, a message is sent back to confirm that the circuit has been established and proper data sending

can start. This switching technique is well suited for frequent and long messages between two nodes.

- store and forward
  In SAF the messages is split in smaller independently routed chunks called packets. This scheme does not require full path pre-establishment overhead. Each packet is entirely stored in a buffer and only then can be forwarded to the next router.

- virtual cut through
  VCT reduces the latency of SAF switching scheme. The switching granularity is smaller than a packet. Elements of the packet called flits can already be sent even if the packet is not fully arrived. If the packet's output port is busy, the packet is stored in a buffer. Buffer space is the same as for SAF.

- wormhole
  In wormhole, the input buffer space is typically dimensioned to the size of a couple of flits. When a packet accesses a router output port used by another packet, the packet remains distributed over all the previous router buffers along the path.

- mad postman
  The mad postman switching technique is comparable to the wormhole switching technique. The difference is that the flow granularity is set at the bit-level instead of the flit-level to reduce latency overhead.

- switched virtual circuit
  In Switched Virtual Circuit an header is sent first. When it reaches the destination, an acknowledgment is sent back to the source The switched virtual circuit technique is appropriate for frequent and long messages.

The switching technique specify the way a message is forwarded on the network and considerably affects the network delay.

1.7.2 Network Design Decisions

On-chip communication architectures offer a vast design space. Designing a complete network thus appears as a very complex and difficult problem. It consists in performing decisions on the value of each parameter, to optimize a cost function. Real design decisions are generally neither completely independent nor fully dependent of each others. The on-chip communication architecture characteristics can be split following the OSI stack model in

which networks can be viewed at different abstraction layers. Upper layers decisions will impose constraints on the lower layers values of decisions to optimize a given cost function. As mentioned the seven layers of the OSI network model can be identified in an on-chip communication architecture: application, presentation, transport, network, link and physical layers.
The *Application layer* offers high-level communication services to the Intellectual Property (IP) cores. Application layer decisions are related to the definition of the inter-process communication interface and primitives (interfaces can be custom or compliant to widely diffused standard like Open Core Protocol) and to the supporting of a determined level of Quality of Service (QoS).
The *Presentation layer* provides data formatting services to the application layer (e.g. encryption).
The *Session layer* is managing the establishment and coordination of the end-to-end connections.
The presentation layer and the session layer are often merged with the neighbour layers in network protocol implementations.
The *Transport layer* offers services like end-to-end communication services to the IP cores. With respect to the transport layer, a decision must be taken between two modes of communication: the connection-oriented or connectionless modes. In the connection-oriented mode, after negotiation, source and destination agree on a set of QoS parameters. The end-to-end connection will act as if cores were directly interconnected to each other, and it must be established prior to the actual data transfer and released once it is no more exploited. In the connection-less mode the transport layer then simply forward data to the network layer after packetization. Moreover while defining the transport layer a decision must be taken on the flow and congestion control strategies used to prevent contention in the network either by adapting the source data flow to the network capacity or by adapting the flow of data transiting in the network. The most popular techniques are credit-based flow control and thresholdbased control. The credit-based flow control technique consists in permitting the incoming data to enter the network only when it acquires one flow control credit, that is released when the data arrive at the destination. The network load is limited by a fized number of credits allowed through the network. The threshold-based flow control is based on the regulation of data injection in the network depending on the network congestion status.
The *Network layer* describes how data are sent from source to destination node. It includes decisions about network topology, routing techniques and routing granularity.
The *Link layer* describes how data are sent between the communication de-

vices constituting the on-chip architecture. Decision taken at this layer are related to the encoding techniques, the switching techniques, the amount and the location of the buffering resources in the network, the arbitration techinques used to handle multiple access to a shared resource, the flow control granularity.
The *Physical layer* deals with the physical implementation of the network on the chip. It includes the circuit and process technologies that are used and the network physical lay-out. Synchronicity is also addressed in this layer as well as low-swing drivers and other interconnect options.

1.8 State of the Art NoC Architectures

This paragraph provides a big picture of the state of the art in NoC propositions, as currently found in the available literature, summarized in figure 1.4.

(NA or shadowed boxes means that data are not available, whereas GT means guaranteed throughput)

The features presented for each considered Data can be divided into three groups:

- features related to network and switch structure, presented in the four first columns;
- features related to performance data, in the following three columns;
- features related to prototyping and/or silicon implementation data, in the last column.

A basic common choice is the use of packet switching (with the exception of aSOC), where the definition of the route is fixed during the hardware instantiation time.

Two connected concepts, network topology and routing strategy, are the subject of the first column. The predominant network topology is the 2D mesh, because this choice implies three advantages:

- facilitated implementation using current IC planar technologies;
- simplicity of the XY routing strategy; and finally
- network scalability.

NoC	Topology / Routing	Flit Size	Buffering	IP-switch Interface	Switch Area	Estimated Peak Performance	QoS Support	Implementation
SPIN - 2000	Fat-tree / Deterministic and adaptive	32 bits data + 4 bits control	Input queue + 2 shared output queue	VCI	0.24 mm^2 CMOS 0.13μm	2 Gbits/s per switch		ASIC layout 4.6 mm^2 CMOS 0.13μm
aSOC - 2000	2D mesh (scalable) / Determined by application	32 bits	None		50,000 transistors		Circuit-switching (no wormhole)	ASIC layout CMOS 0.35μm
Dally - 2001	Folded 2D torus / XY source	256 bits data + 38 bits control	Input queue		0.59 mm^2 CMOS 0.1μm (6.6 % of a tile)	4 Gbits/s per wire	GT - virtual channels	No
Nostrum – 2001	2D mesh (scalable) / Hot potato	128 bits data + 10 bits control	Input and output queues		0.01 mm^2 CMOS 65nm			
Sgroi - 2001	2D mesh / NA	18 bits data + 2 bits control		OCP				
Octagon-2001	Chordal ring / Distributed and adaptive	Variable data + 3 bits control				40 Gbits/s	Circuit-switching	No
Marescaux - 2002	2D torus (scalable) / XY blocking, hop-based, deterministic	16 bits data + 3 bits control	Input queue	Custom	611 slices VirtexII (6.58% area overhead XC2V6000)	320Mbits/s per virtual channel at 40 MHz	2 virtual channels (to avoid deadlock)	FPGA VirtexII / VirtexII Pro
Bartic – 2003	Arbitrary (parameterizable links) / Deterministic, virtual-cut-through	Variable data + 2 bits control / link	Output queue	Custom	552 slices + 5 BRAMs VirtexII Pro for 5 bidirectional links router	800Mbits/s per channel for 16-bit flits at 50 MHz	Injection rate control, congestion control	FPGA VirtexII Pro
Æthereal - 2002	2D mesh / Source	32 bits	Input queue	DTL (Philips proprietary standard)	0.26 mm^2 CMOS 0.12μm	80Gbits/s per switch	Circuit-switching	ASIC layout
Eclipse – 2002	2D sparse hierarchical mesh / NA	68 bits	Output queue					No
Proteo – 2002	Bi-directional ring / NA	Variable control and data sizes	Input and output queues	VCI				ASIC layout CMOS 0.18μm
SOCIN – 2002	2D mesh (scalable) / XY source	n bits data + 4 bits control (parameterizable)	Input queue (parameterizable)	VCI	420 LCs APEX FPGAs (Estimated, for n=8, bufferless)	1 Gbits/s per switch at 25 MHz	No	No
SoCBus – 2002	2D mesh / XY adaptive	16 bits data + 3 bits control	Single position input and output buffers	Custom		2.4 Gbits/s per link	Circuit-switching	No
QNOC - 2003	2D mesh regular or irregular / XY	16 bits data + 10 bits control (parameterizable)	Input queue (parameterizable) + Output queue (single position)	Custom	0.02 mm^2 CMOS 90nm (Estimated)	80 Gbits/s per switch for 16-bit flits at 1GHz	GT - virtual channels, (4 different traffic)	No
T-SoC – 2003	Fat-tree / Adaptive	38 bits maximum	Input and output queues	Custom/ OCP	27000 to 36000 two input NAND gates		GT - 4 virtual channels	
Xpipes – 2002	Arbitrary (design time) / Source static (street sign)	32, 64 or 128 bits	Virtual output queue	OCP	0.33 mm^2 CMOS 100nm (Estimated)	64 Gbits/s per switch for 32-bit flits at 500MHz	No	No
Hermes – 2003	2D mesh (scalable) / XY	8 bits data + 2 bits control (parameterizable)	Input queue (parameterizable)	OCP	555 LUTs 278 slices VirtexII	500 Mbits/s per switch at 25 MHz	No	FPGA VirtexII

Figure 1.4: State of the art in NoCs overview

Another approach is to use the 2D torus topology, to reduce network diameter (Marescaux-2002). The folded 2D torus (Dally-2001) reduces also the cost in wiring. A common problem of mesh and torus topologies is the associated network latency. To overcome this problem the following architectures have been proposed:

- SPIN that employ a fat-tree topology;
- T-SoC that employ a fat-tree topology; and finally
- Octagon that proposes the use of a chordal ring topology.

All of them lead to a smaller network diameter, with consequent latency reduction. Concerning routing strategies, there is a clear lack of published information; this means that this is still an open field.

The second important quantitative parameter of NoC switches is flit size. It is possible to classify approaches in two groups the first one focused on future SoC technologies and the other concerned on the existing limitations. The first group uses wide switching channels (150 to 300 wires), without significantly affecting the overall SoC area. This can be achieved by using a future 60nm technology for building 22mm x 22mm chip with a 10 x 10 NoC to connect 100 2mm x 2mm IPs. The second group uses flit size ranging mostly from 8 to 64 bits, a data width similar to current processor architectures. The next parameter considered is the switch buffering strategy. Most NoCs employ input queue buffers. Since input queuing implies a single queue per input, this leads to lower area overhead but, at the same time, presents problems of head-of-line blocking. To overcome this problem, output queuing can be used, the counterbalance of this improvement is a greater buffering cost, since the total number of queues is increases. An intermediate solution is to use virtual output queuing associated with time-multiplexed virtual channels ($\times$pipes).

Another important parameter, related to the buffering field, is the FIFO size, which implies the need to solve the compromise among of the amount of network contention, packet latency and switch area overhead. Bigger queues lead to small network contention, higher packet latency, and bigger switches; other way round for the smaller ones.

The last structural parameter is the characteristic of the IP-switch interface. The use of standard communication interfaces for the on-chip environment is an evolving trend in industry and academia, to allow re-use of cores. VCI and OCP standards are used by several of the NoC proposals presented in figure 1.4.

The fifth column collects results concerning the size of the switch. It is interesting to observe that the two approaches targeted to ASICs (SPIN-2000 and Æthereal-2002), both with a 32-bit flit size, have similar dimensions, around 0.25mm2 for similar technologies. FPGA prototyped systems produced results ranging from 555 LUTs (Hermes-2003) to 1222 LUTS (611 slices) (Marescaux-2002). This difference comes from the fact that Marescaux-2002 employs virtual channels, while Hermes-2003 does not, leading to smaller switch area. Switch size, flit size and switch port cardinality are fundamental values to allow estimating the area overhead and the expected peak performance for on-chip communication.

In the following, two communication architectures, a bus based and a NoC, are presented more in detail, being the reference for the discussions reported in following chapters.

1.8.1 AMBA Shared Bus

The Advanced Microcontroller Bus Architecture (AMBA) 2.0 [4] interconnect is a well-established fabric architecture for SoC designs, thanks to its efficiency despite the moderate silicon footprint. Therefore, we choose it as a reference against which to compare the ×pipes NoC. The AMBA specification dictates three different architectures with varying levels of complexity and performance; in this paper, we will refer to Advanced High-performance Bus (AMBA AHB), the fastest of them.

AMBA traditionally leverages upon a shared bus topology, and its communication protocol is kept simple to minimize area overhead (a single ongoing transaction at a time, no posted writes, *etc.*). An AHB bus is composed of several IP blocks (AHB masters and slaves), of one central arbiter to manage resource access, and of some interconnection resources. A minimal amount of flip-flops is required in the architecture, which is typically bufferless. The bus resources are owned by a single master at a time; if the targeted slave is forced to insert wait states before responding, no other transaction can be initiated, neither by the current bus owner nor by any other master. As a result, the utilization of bus bandwidth, which is already limited, might be poor.

AMBA AHB uses non-pipelined paths for communication among all masters and all slaves. Therefore, a key performance assumption is that the propagation delay of the interconnect wires will be short. If that is the case, communication will incur the minimum possible latency. However, new technology nodes are leading to faster and faster logic, potentially resulting in faster clock periods, while wire propagation delays are proportionally increasing. If the whole fabric is constrained to slow operation by wire delay, this

factor represents a limit to maximum operating frequency.

1.8.2 AMBA Multi-Layer

Due to increasing congestion and bandwidth demands in modern SoCs, a crossbar component was added to the AMBA toolkit, resulting in "Multi-Layer" (ML) AMBA designs. To keep existing AMBA interfaces unchanged, the device is purely combinational and completely transparent to AMBA masters and slaves. The crossbar component behaves as a slave towards multiple AMBA AHB buses ("layers"), and forwards requests to real AMBA slaves (*e.g.* memories). Multiple layers can simultaneously access the crossbar, provided they do not conflict on arbitration for the same transaction target.

Given its high-bandwidth topological nature and its combinational response times, this crossbar is clearly a best-case reference block from an architectural point of view. However, from the perspective of a layout engineer, wire routing constraints in such a block are very demanding. Therefore, an assessment of the achievable clock frequency and area is clearly essential.

1.8.3 ×pipes NoC

The ×pipes NoC [3, 15] is an example of a highly flexible library of component blocks (Fig. 1.5). The NoC is instantiated by deploying a set of components in an arbitrary topology and by configuring them. The ×pipes library contains three main components: switches, Network Interfaces (NIs) and links.

The backbone of the NoC is composed of switches, whose main function is to route packets from sources to destinations. Arbitrary switch connectivity is possible, allowing for implementation of any topology. Switches provide buffering resources to lower congestion and improve performance; in ×pipes , output buffering is chosen, *i.e.* FIFOs are present on each output port. Switches also handle flow control [13] issues (we use the ACK/NACK protocol in this paper), and resolve conflicts among packets when they overlap in requesting access to the same physical links.

An NI is needed to connect each IP core to the NoC. NIs convert transaction requests/responses into packets and vice versa. Packets are then split into a sequence of flits before transmission, to decrease the physical wire parallelism requirements. In ×pipes , two separate NIs are defined, an *initiator* and a *target* one, respectively associated to system masters and system slaves. A master/slave device will require an NI of each type to be attached to it. The interface among IP cores and NIs is point-to-point as defined by the OCP 2.0 [12] specification, guaranteeing maximum reusability. NI Look-Up

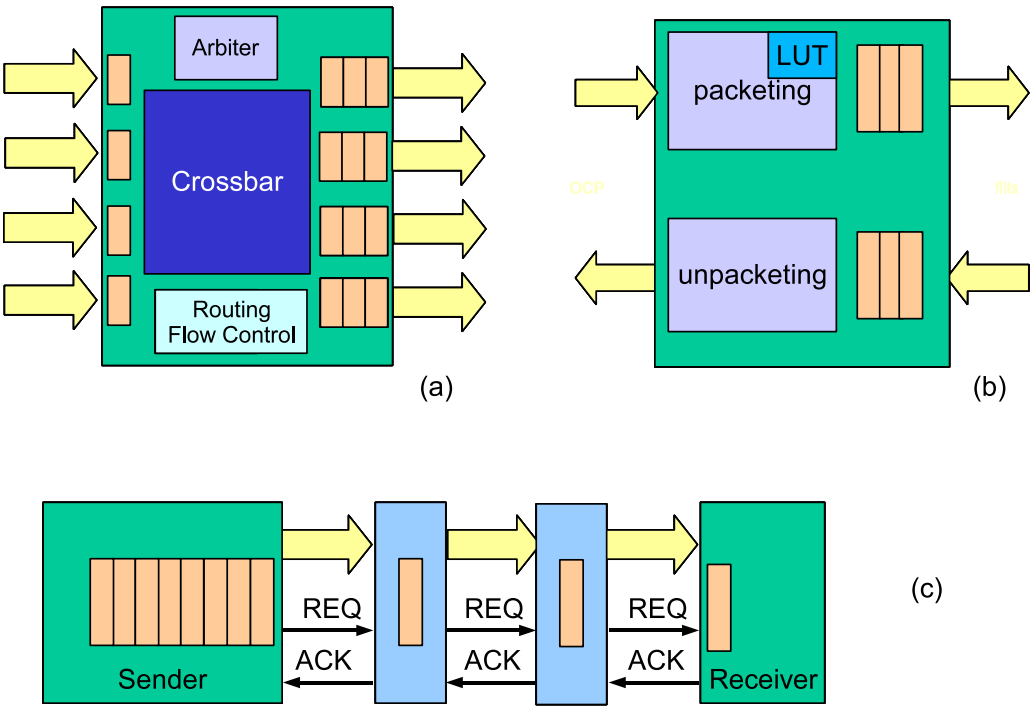


Figure 1.5: ×pipes building blocks: (a) switch, (b) NI, (c) link

Tables (LUTs) specify the path that packets will follow in the network to reach their destination (source routing). Two different clock signals can be attached to NIs: one to drive the NI front-end (OCP interface), the other to drive the NI back-end (×pipes interface). The ×pipes clock frequency must be an integer multiple of the OCP one. This arrangement allows the NoC to run at a fast clock even though some or all of the attached IP cores are slower, which is crucial to keep transaction latency low. Since each IP core can run at a different divider of the ×pipes frequency, mixed-clock platforms are possible.

Inter-block links are a critical component of NoCs, given the technology trends for global wires. The problem of signal propagation delay is, or will soon become, critical. For this reason, ×pipes supports link pipelining [13], *i.e.* the interleaving of logical buffers along links. Proper flow control protocols are implemented in link transmitters and receivers (NIs and switches) to make the link latency transparent to the surrounding logic. Therefore, the overall platform can run at a fast clock frequency, without the longest wires being a global speed limiter. Only the links which are too long for single-cycle propagation will need to pay a repeater latency penalty.

Bibliography

[1] International technology roadmap for semiconductors (ITRS). Technical report, International Technology Roadmap for Semiconductors.

[2] Tapani Ahonen et al. A brunch from the coffee table – case study in NoC platform design. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, pages 425–453. Kluwer Academic Publishers, 2004.

[3] Federico Angiolini, Paolo Meloni, Davide Bertozzi, Luca Benini, Salvatore Carta, and Luigi Raffo. Networks on chips: A synthesis perspective. In *Proceedings of the 2005 ParCo Conference (to be published)*, 2005.

[4] ARM Ltd. The Advanced Microcontroller Bus Architecture (AMBA) homepage. www.arm.com/products/solutions/AMBAHomePage.html.

[5] Luca Benini and Giovanni De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70 – 78, January 2002.

[6] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, pages 684–689, June 2001.

[7] Jorg Henkel, Wayne Wolf, and Srimat Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. *vlsid*, 00:845, 2004.

[8] M. Horowitz, R. Ho, and K. Mai. The future of wires, 1999.

[9] Ali Keshavarzi, Kaushik Roy, and Charles F. Hawkins. Intrinsic leakage in deep submicron cmos ics-measurement-based test solutions. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(6):717–723, 2000.

[10] Anthony Leroy. *Optimizing the on-chip communication architecture of low power Systems-on-Chip in Deep Sub-Micron technology.* PhD thesis, Universite Libre de Bruxelles Faculte des Sciences Appliquees, 2007.

[11] James D. Meindl. Interconnect opportunities for gigascale integration. *IEEE Micro*, 23(3):28–35, 2003.

[12] Open Core Protocol Specification, Release 2.0. www.ocpip.org, 2003.

[13] Antonio Pullini, Federico Angiolini, Davide Bertozzi, and Luca Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design (SBCCI)*, pages 224–229, 2005.

[14] Sonics Inc. SonicsMX. www.sonicsinc.com/sonics/products/smx/.

[15] Stergios Stergiou, Federico Angiolini, Salvatore Carta, Luigi Raffo, Davide Bertozzi, and Giovanni De Micheli. ×pipes Lite: A synthesis oriented design library for networks on chips. In *Proceedings of the 2005 Design, Automation and Test in Europe Conference (DATE)*, pages 1188–1193. IEEE, 2005.

[16] STMicroelectronics. The STBus interconnect. www.st.com.

[17] Drew Wingard. Micronetwork-based integration for SoCs. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 673–677. ACM, June 2001.

Chapter 2

Comparative analysis of NoCs and Traditional Interconnects

As already mentioned, the ever shrinking lithographic technologies available to chip designers enable performance and functionality breakthroughs, but they bring new hard problems. For example, Multi-Processor Systems-on-Chip (MPSoCs) featuring several processing elements can be conceived, but efficiently interconnecting them while keeping the design complexity manageable is a challenge. In this first part of the thesis, different alternative approaches to the on-chip interconnect designs are compared. In particular, three different kinds of architecture are analyzed: traditional buses, easy to deploy an to understand, but not capable of providing enough bandwidth for modern complex systems; packet-switching Networks-on-Chip (NoCs), usually claimed to be more scalable with respect to the busses, but more complex to design and to tailor; hybrid architectures interleaving bandwidth-rich components (*e.g.* crossbars) within the preexisting fabrics. In this chapter the results of this analysis pointing out the strengths and weaknesses of these alternative approaches are presented. In particular, a thorough analysis based upon actual chip floorplans after the interconnection place&route stages was performed, to cover a big lack in NoC research field, answering to difficult questions associated to chip layout problems. Since interconnects, by their own nature, span across significant portions of the die, wiring congestion and wire propagation delays are very difficult to estimate in advance. Therefore, the overhead of crossbars and NoCs can only be fully understood after having laid out the architecture onto a chip floorplan. The placement&routing step is instrumental to a synthesis flow that properly investigates wiring loads, to evaluate the delay impact of long-range wiring resources and to assess the real usefulness of segmented architectures. The delay estimation provided by synthesis tools is verified against post-placement figures. Performance, area and

power results are discussed, while keeping an eye on the scalability prospects in future technology nodes. Similarly, the distribution of the clock signal to the cells in the design is a crucial point. It is well known that the clock tree can represent a significant percentage of the power budget [13, 15, 28], while another significant fraction is due to sequential logic. Therefore, the clock distribution tree and the clock gating techniques are prime candidates for evaluation. To take into account as many key effects as possible, a back-end flow which takes the test platforms down to a placed&routed layout was used. From the designs, obtained by means of this flow, frequency, area and power figures were derived, referring to a 0.13 μm library.

2.1 Related Work

Large numbers of communication fabrics are described in previous literature. ARM AMBA [5], including the latest AXI [6] packages, STMicroelectronics STBus [24] and Sonics MicroNetworks [30] are examples of buses and of attempts to overcome the most classical shared bus architecture by various means (for example, multiple STBus channels can be deployed, leading to crossbars).

NoCs have been suggested as a scalable communication fabric [8, 14]. Research has focused on multiple design levels. From the architectural point of view, a complete scheme is presented for example in [16], while specific topics are tackled in several works: flow control protocols [20], Quality of Service (QoS) provisions [11, 29], asynchronous implementations [10]. A CAD tool for NoC instantiation and optimization can be found for example in [9].

The synthesis flow of NoCs has been explored by several groups. Layouts are presented in [2, 21], a test chip is shown in [17], and an FPGA target is provided for [32]. Synthesis and layout results for the ×pipes library of component blocks that we will leverage upon are shown in [3, 23].

As previously mentioned, one key topic which has not yet been extensively covered is studying how NoCs compare to more traditional interconnects. In [31], an analytical methodology is illustrated to compare NoCs of arbitrary topology (a shared bus and a crossbar are provided as examples) also taking into account area, frequency and power metrics. However, some assumptions of this work (such as the relative cost of wiring *vs.* logic) do not seem to be fully confirmed when considering actual fabrics, as Section 2.5 will show. In [22], a synthesis-aware flow is presented to characterize the Hermes NoC; PI Bus is used as a benchmark for performance metrics, but not for area and power analysis. Further, PI Bus is not representative of current, widely used

high-performance interconnects.

2.2 The Fabrics Under Test

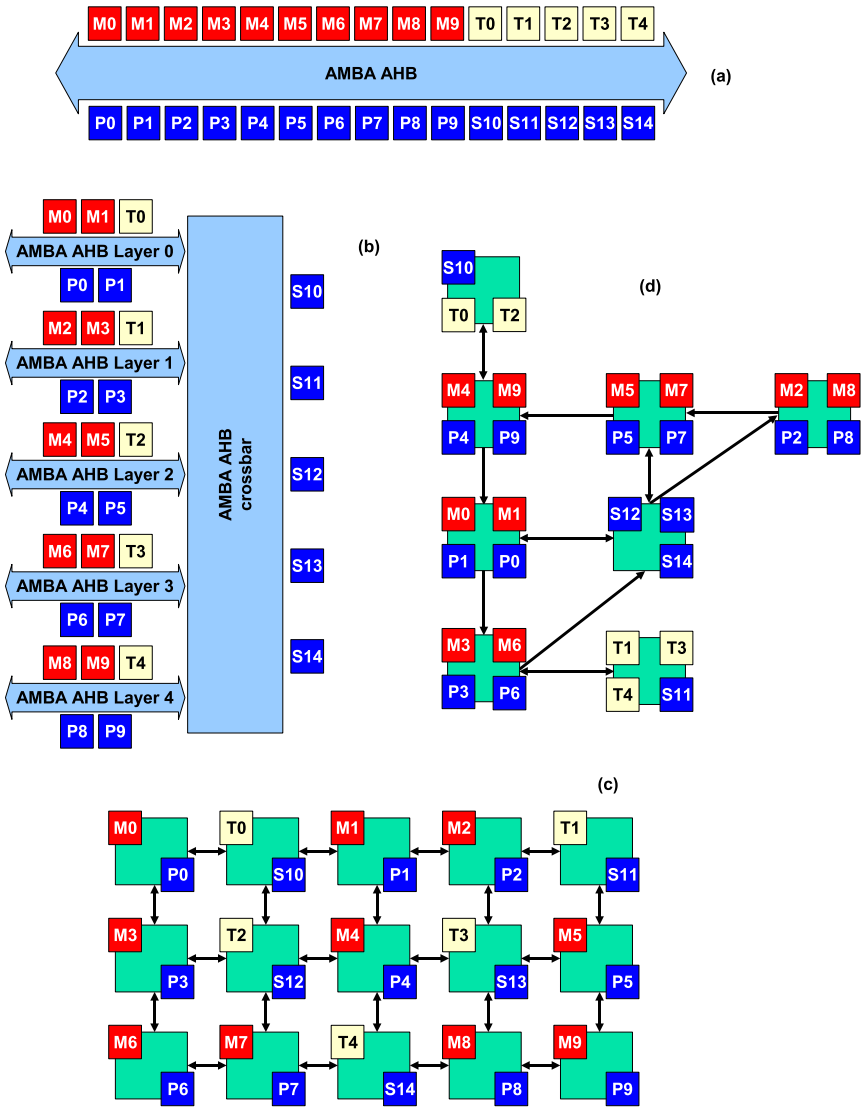


Figure 2.1: The platforms under test. (a) shared bus AMBA AHB; (b) ML AMBA AHB; (c) ×pipes mesh; (d) ×pipes custom topology. **M**: ARM7 masters; **T**: traffic generators; **P**: privately accessed slaves; **S**: shared slaves

It must be noticed that choosing a test environment to compare such different architectures as a NoC and a bus/crossbar is a difficult task, since

communication fabrics can be heavily tuned to optimally fit a target benchmark - but the resulting figures would not be representative of real-world performance under a different test load. For this reason, it is not meaningful, for the sake of the proposed benchmarking, to take into account deeply optimized evaluation platforms, specifically tailored for one target application. Thus it is more useful to present relatively regular mappings which should be suitable for multiple applications. Anyway, evaluating the impact on the comparison results obtained by means of an application-specific optimization, is an interesting speculation that is suitable to be added to the analysis. In this aim, a NoC irregular topology, optimized to better match the target application, according to criteria of low area occupation and low power consumption, was included in the comparison. The ×pipes NoC module library, as mentioned, is taken as reference architecture. Summarizing, four test platforms were conceived, namely:

- an AMBA AHB shared bus;
- an AMBA AHB ML system containing a crossbar element;
- a ×pipes mesh
- the mentioned ×pipes custom topology (Fig. 2.1).

All fabrics allow for attaching up to 30 IP cores, of which 15 masters and 15 slaves (typically memory banks). This amount is justified considering that, already at the 0.13 μm lithography node, simple processor elements and 32 kB memory banks can be expected to require just about 1 mm^2 of die area. In fact, it is logical to expect that this amount of IP cores may well be surpassed in some of the next-generation MPSoCs.

The ML AMBA topology is not a full crossbar. A full 15x15 crossbar would be prohibitively expensive in terms of area and wiring. In fact, the IP library used to synthesize this fabric (see Section 2.4) only allows instantiation of up to 8x8 components. The ML AMBA test fabric contains a middle sized 5x5 crossbar. For both the ML and shared bus AMBA designs, the canonical data width of 32 bits is chosen, since it represents the best match for ARM7 cores. For the ×pipes NoCs, non-pipelined links are instantiated, in the assumption, confirmed by experimental results, that the nature of the topologies should provide enough wire segmentation to guarantee single-cycle propagation on all links, at least in 0.13 μm technology. The NoC mesh is configured with two different flit sizes, namely 21 and 38 bits, to explore the dependency of area, power consumption and performance on this parameter. These numbers are chosen taking into account the length of each possible

packet type and trying to optimize the resulting flit decomposition. The OCP pinout is configured with 32 bit data ports. The custom NoC topology is configured with 21-bit flits to compare it against the mesh. For all the experiments, the NoC components (switches and NIs) are always configured with FIFO buffers having a depth of three flits, since from previous design space explorations performed on the ×pipes architecture, this value proved to be a good tradeoff between performance and area/power cost.

2.3 The Test Applications

The performance of the interconnects are tested under two main scenarios:

- a multimedia processing application
- a Data Encryption Standard (DES) encryption algorithm.

Both applications are parallelized to be suitable for multiprocessor computation. The task graphs for both can be seen in Fig. 2.2. As can be noticed, the multimedia application is fundamentally a pipeline of computation tasks; the encryption application features a producer task (to split an incoming data stream into chunks of data), a consumer task (to reassemble the outputs) and a set of "worker" tasks which operate in parallel to perform the actual encryption. In the testing, every task is mapped onto a single processor. Testing results related to both applications derive from several run iterations during which the processes were feeded them with a stream of input data. Moreover the experiments are evaluated by capturing performance statistics only during the execution of the application kernel, *i.e.* skipping the boot stage and properly handling initialization or shutdown periods where some of the tasks may be running while some others may be idle. This guarantees proper handling of cache-related effects.

The multimedia application was implemented as a standalone program, which can directly run on ARM CPUs, while the encryption algorithm is an example of a software running on top of the RTEMS [19] operating system.

In both benchmarks, communication between nodes is handled by means of a shared memory buffer, while synchronization is achieved via polling of hardware semaphores. The shared memory and the hardware semaphore device are labeled S12 and S13 in Fig. 2.1. To avoid the shared memory to become a huge system bottleneck, processors are assigned private memory buffers (P0 to P9 in Fig. 2.1), on which they can operate without the need of using synchronization primitives. These private buffers are cacheable, while the shared components are not, to avoid coherency issues. Therefore, the inter-processor communication paradigm is as follows:

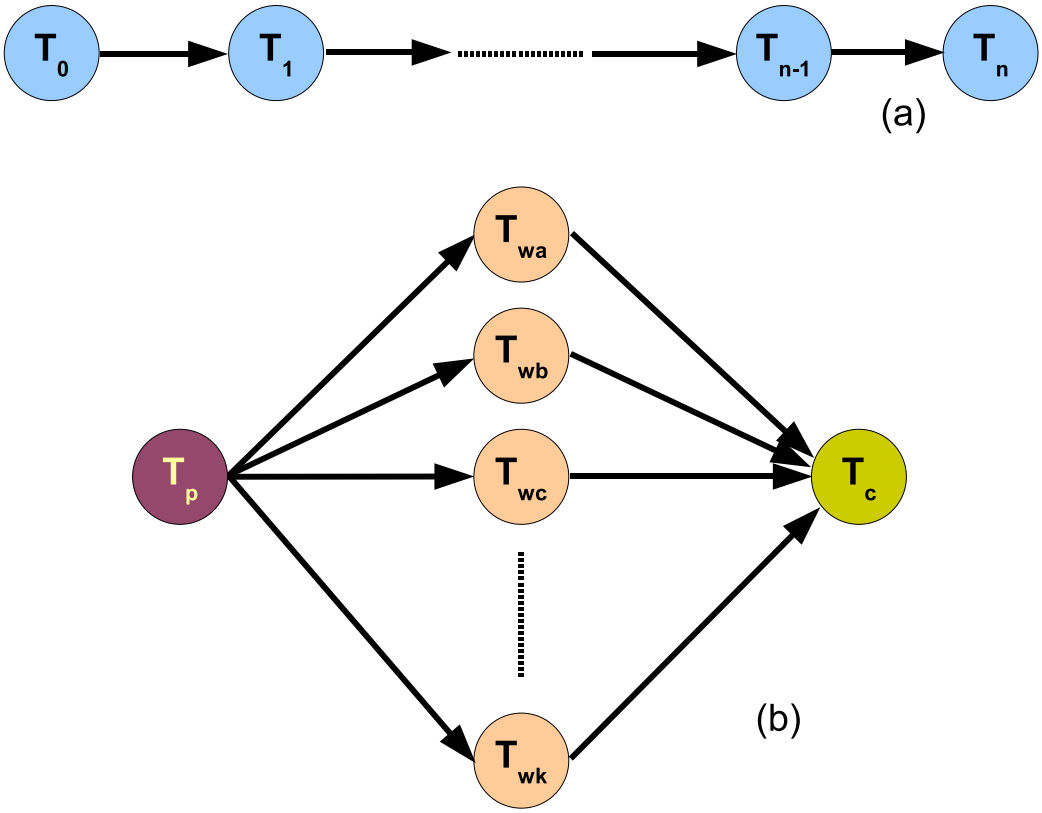


Figure 2.2: Task graphs for the two applications under test: (a) multimedia processing application, (b) LMS filtering application

- Producers prepare a data set in their private memory space. In the meanwhile, consumers operate on the previous chunk of data in their private memory space.
- When ready, producers copy the new data set to shared memory. This may need semaphore polling if the shared memory buffer is still busy with the previous transaction. As soon as data is copied to shared memory, the producer begins preparing the new message.
- When ready, consumers acquire the new data set from shared memory. This may need semaphore polling if the shared memory buffer does not contain new data yet. As soon as data is copied from shared memory, the consumer begins computation on it.

This communication paradigm is just one of almost endless alternative possibilities. The choice of this particular paradigm rely on the assumption that, since it features both distributed (private memories) and centralized

(single shared memory and semaphore device) elements, it represents a fair comparison ground for such diverse communication fabric topologies such as a shared bus and a NoC. It may be assumed that an approach based on a fully shared memory subsystem would improve the relative performance of a bus-based fabric, while a message passing paradigm would be more suitable for distributed architectures such as NoCs. This analysis is beyond the scope of the reported investigation, but is surely an interesting field for future research work. To verify whether the computation/communication ratio of the applications is a critical factor, two variants of the multimedia application benchmark were implemented, with different degrees of computational requirements: the low-computation variant is performing roughly eight times less mathematical operations. Please note, however, that while the ratio of computation to *explicit inter-processor* communication can be easily tuned in this way, the ratio of computation to *overall* communication requirements depends on several additional factors. For example, unless an ideal cache is available, changing the computation patterns also *implicitly* results in different bandwidth demands (for cache refills and write-backs). This will be further discussed in Section 2.5.

In the following, for the sake of brevity, the benchmarks will be called **multi-high**, **multi-low** (high-computation and low-computation variants of the multimedia benchmark, respectively) and **des**.

Since real-life MPSoCs are not likely to only feature general purpose processing cores, traffic generators were also deployed to model additional hardware IP blocks which may be present in the platform. While this choice is not in any way supposed to model on-chip coprocessors in a general fashion, it adds extra realism. Therefore, two different types of traffic generation patterns were include: DSP-like (streams of accesses to a memory bank) and I/O controller-like (a rotating pattern of accesses towards neighbouring devices). DSP-like traffic generators are each programmed to fetch 128 or 256 bits of data from one of the shared memory banks or devices (indicated with S in Fig. 2.1), compute for 10 clock cycles, and repeat. I/O controller-like traffic generators are instead programmed to query three shared devices in a rotating pattern, by reading 256 bits from each. It must be noticed that, since the aim of the analysis is comparing the performance of a packet-based interconnect *vs.* a traditional interconnect under equal conditions, the generators have to be programmed to issue functional traffic (such as data transactions); a lower-level approach with injection of packets in the NoC would make the comparison with AMBA very unintuitive.

2.4 Reference Characterization Flow

In the present section we introduce for the first time in the thesis a complete simulation and design flow for ×pipes . The mentioned flow takes as input the architecture definition and, on a side, brings to the evaluation of the application communication requirements and to the verification of the system functionalities, on the other allows to obtain the layout masks for tape out, going through the synthesis and the place& route steps. The flow includes the use of home-made (when needed) and commercial tools, and its definition required avery big effort due to the unusual field of application of the standard implementation routines performed by the involved software. The ×pipes design flow is taken as reference in the rest of the thesis, since the implementation of sample NoC designs at layout level is key for the sake of every research topic presented in this thesis. For the sake of the argument discussed in this chapter, the tool was used to properly characterize the platforms mentioned in Section 2.2. Thus, since the implementation of AMBA platforms was also needed, and since it requires some different operation to be performed, a small subsection explaining AMBA synthesis procedure is also included. Simulation and synthesis flows are described more in detail in the following.

2.4.1 Fabric Simulation

The platforms under test were rendered into MPARM (Fig. 2.3), a SystemC cycle-accurate virtual platform [18]. MPARM allows for accurate injection of functional traffic patterns as generated by real IP cores during a benchmark run. Further, it provides facilities for debugging, statistics collection and tracing. This virtual platform allows us to perform the tasks outlined at the top right of Fig. **??** and Fig. **??**.

10 ARM7 processors (M0 to M9 in Fig. 2.1), with caches, and 5 custom traffic generators (T0 to T4), four of which programmed to execute DSP-like traffic and one of which programmed to perform I/O controller-like traffic, were plugged to the master ports of the platforms. The functional multimedia benchmarks described in Section 2.3 were run on the general purpose cores. System slaves (memories) are either accessible by a single master ("private") or subject to contention due to requests by multiple masters ("shared"). This distinction is key, since private memories exhibit optimal latency only if located next to their master (*i.e.* on the same AHB layer or attached to the same ×pipes switch). The placement of shared slaves must comply with functional constraints: for example, in a ML AMBA topology, shared slaves must be put beyond the crossbar component, otherwise only local masters

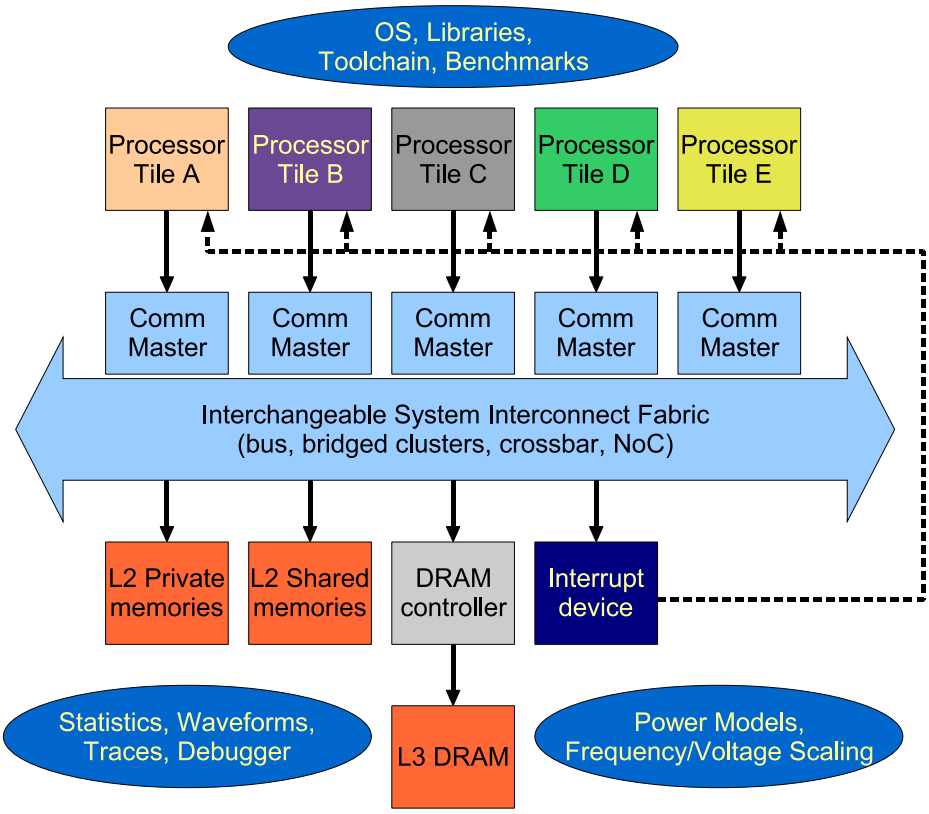


Figure 2.3: The MPARM SystemC virtual platform

will be able to access them.

2.4.2 Fabric Synthesis

The flow to getting a chip layout for the interconnection fabrics encompasses multiple steps. A single 0.13 μm, power characterized, technology library is used.

×pipes Synthesis

The first synthesis step for the ×pipes interconnect (Fig. 2.4, top left) is based on the previously mentioned custom CAD tool, called ×pipes Compiler, by means of which the designer can instantiate and configure ×pipes library components to fit application needs. The user must provide a specification file defining the topology connectivity and routing tables.

Moreover, touching some configuration headers, the architecture modules can be parameterized: the flit width can be set to arbitrary values, switches

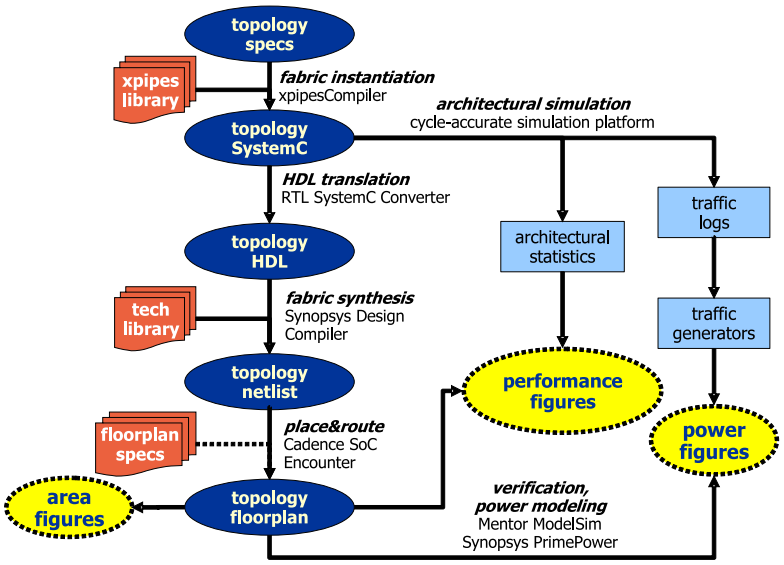


Figure 2.4: The synthesis flow for our test fabrics: ×pipes

can be customized in terms of I/O ports, buffering resources and arbitration policies, while NI degrees of freedom are buffering resources and clock dividers. The ×pipes Compiler tool processes these specifications to generate a SystemC instance, which is suitable both for architectural simulation and for synthesis. Second step of the flow is a SystemC-to-HDL automatic translation. The outputs include the HDL code for traffic generators, for the NoC building blocks, and for the top level instantiation layer. Following step is the logical synthesis of the NoC building blocks. The synthesis is performed with Synopsys Design Compiler [26]. The synthesis step receives in input the HDL description of the building blocks and produces in output a standard-cell netlist. The synthesizer is also fed with a set of constraints expressed in a Synopsys Design Constraint (SDC) format. This permits also to use the same set of constraints to feed both the synthesizer and the place and route tool, since both support the same general format. Inside the SDC constraint file, the specified constraints defined a target for the optimization processes related to the timing features of the netlist, to the area occupation and to its power consumption. In particular, realistic constraints obtained by performing some preliminary synthesis iterations were defined for the critical path and for the maximum input and output delays. On the other hand, since area and power optimization processes have lower priority with respect to the timing, maximum optimization effort is allowed for both this tasks. In this way, the user prevents the synthesis results to be damaged by an aggressive timing optimization aimed to reach unrealistic targets, while keeping area

an power results as low as possible. The output of the synthesis process is a verilog netlist instantiating a set of cells from the library, and a Standard Delay Format (SDF) file providing information about the delay due to the cells and to the interconnects, estimated for each path in the netlist. This SDF file is needed to perform the post-synthesis simulations. Full custom logic blocks would lead to better performance, but logical synthesis process avoids is a better candidate to illustrate the results achievable with a completely synthetic, and therefore maximally flexible, approach.

During synthesis, Design Compiler was instructed to save power when buffers are inactive by applying clock gating to NoC blocks. The gating logic can be instantiated only for sequential cells which feature an *input enable* pin; in ×pipes , such cells are 100% of the flip-flops along the NI datapath and 75% or more of those along the switch datapath. If the target technology library features dedicated clock gating cells, they may be used; in the case of the library that were taken as reference for the sake of this analysis, such devices are not available, therefore Design Compiler implemented the gating circuit by means of generic cells. This incurs a small penalty in operating frequency and power consumption, that could be avoided with a more complete technology library. Once that all NoC blocks have been mapped onto a netlist, a third step (last one that deserves to be included in the synthesis phase) is quickly performed on the top level instantiation code to link the complete topology. A `dont_touch` synthesis directive is issued on the sub-blocks, to prevent the tool to perform additional optimization trials on the linked design, so basically this step is mainly used to attach components to each other.

AMBA Synthesis

The logic synthesis phase is different for the AMBA fabrics with respect to the NoC architecture. AMBA synthesis (Fig. 2.5, top left) is performed by using the Synopsys CoreAssembler [25] tool to instantiate the IP cores included in the Synopsys AMBA DesignWare libraries, therefore composing the needed topologies. During this phase, design parameters (such as data lane width) and constraints can be defined. The final result is a low-level HDL netlist composed of technology library standard cells representing the interconnect fabric, with AMBA AHB masters and slaves instantiated as black boxes.

Placement&Routing

The final place&route phase is performed with Cadence SoC Encounter suite [12] (Fig. 2.5 and Fig. 2.4, bottom left) is common to all fabrics, . First, a manual floorplanning phase is performed, where the hard black

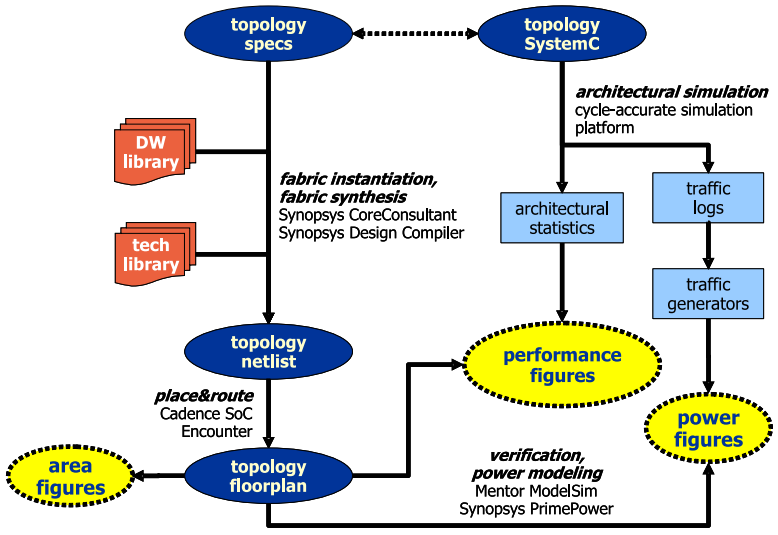


Figure 2.5: The synthesis flow for our test fabrics: AMBA

boxes are placed on the floorplan by the designer. *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 and finally the wire routing steps are performed. From preliminary experiments arised that letting the tool handle the complete topology simultaneously, without hierarchical tiling, enables a more thorough optimization. The tool was allowed to use over-the-cell routing, *i.e.* to route wires on top of the black boxes representing IP cores, out of the eight metal layers that the reference technology library allows, only the top three were conservatively used for routing; the bottom five remain free for local wires of IP cores. During the place&route phase, a clock tree aimed to reduce the clock skew across the register pins inside the chip was synthesized. Some constraints for the clock tree were specified inside a .ctstch file. The synthesis was performed automatically by the tool according with the mentioned constraints. As will be explained more in detail in chapter 3, for the sake of the ×pipes custom topology implementation, a small variant is introduced. In this case SoC Encounter is feeded with an automatically-generated floorplan specification file. This file contains information about the layout fences, and is built by an in-house CAD tool based on ×pipes area models. Such an approach [7] lets the designer skip the tedious activity of manually placing blocks on the floorplan, and iteratively improving the result by means of trial-and-error tighter packing.

Post-Layout Analysis

Post-layout verification and power estimation (Fig. ?? and Fig. ??, bottom right) is achieved as follows. First, the HDL netlist representing the final placed&routed topology is simulated by injecting functional traffic through the OCP ports of the NIs. This simulation is aimed both at verifying functionality of the placed fabric and at collecting a switching activity report. At this point, accurate wire capacitance and resistance information, as backannotated from the placed&routed layout, is combined with the switching activity report using Synopsys PrimePower [27]. The output is a layout-aware power/energy estimation of the simulation.

2.5 Performance comparison results

2.5.1 Interconnect Performance

First, a cycle-accurate architectural simulation of the alternative topologies under the load of three benchmarks was performed, as described in Section 2.3 and Section 2.4.1, to assess their performance. The plots in Fig. 2.6 summarize the global results. The vertical axis represents the relative benchmark execution time, taking as the baseline the execution on the multilayer AMBA AHB topology. Execution times are computed by first running an architectural simulation, which provides results in terms of clock cycles, and then by backannotating the clock period as resulting from the synthesis flow, as discussed in Section 2.4. Frequency results will be discussed in more detail in Section 2.5.2, but it is worth to anticipate that it was possible to achieve 370 MHz for the AMBA topologies and 793 MHz for the NoC topologies. Realistically, ARM7 cores should be able to run up to a frequency of 400-500 MHz in 0.13 μm lithography. Since the ML AMBA topology is capable of achieving 370 MHz at most, and the overhead for the usage of dual clock synchronization FIFOs would not be justified in this case, the system can be considered fully synchronous at 370 MHz. For the NoC, the dual clock support offered by ×pipes was exploited to run the cores at 396.5 MHz and the NoC at its maximum frequency of 793 MHz. The 7% frequency boost given to the cores is small and does not represent an unfair advantage for the NoCs; in fact, it is a byproduct of the high clock frequency achievable by NoCs, a feature that must be exploited as much as possible by NoC designers. The tests was repeated with three different cache sizes for the ARM7 processors; smaller caches translate into more cache misses and more congestion on the fabric.

In all benchmarks, the shared bus is completely saturated and execution

times are unreasonable - about four times larger than with the other interconnects. In **multi-high** and **multi-low**, the 21-bit NoC mesh exhibits a small but noticeable advantage with respect to the ML AMBA fabric of about 5% to 15%, with the largest gain being achieved in high-congestion (small cache) setups. The 38-bit NoC extends the gap to the 10% to 20% range. Even the custom NoC topology, which features much less bandwidth than the meshes, typically performs 10% better than ML AMBA. The figures represent overall benchmark execution time, therefore such improvements are remarkable. Finally, the **des** benchmark is strongly more bandwidth-intensive than **multi-high** and **multi-low**, and therefore the gap among AMBA and the NoC interconnects widens to the 20 to 35% range.

The results do not show a large difference between the **multi-high** and **multi-low** applications; in both cases, the gap between AMBA and NoC topologies is similar. At first sight, the benchmark with more communication and less computation demands (**multi-low**) should give a larger advantage to bandwidth-rich interconnects, such as the NoCs. The results can be better understood by noticing two things.

The first is that, despite a difference of a factor of eight in the performed computation, and all else being equal, this 8X difference is only translating into a gap of about 20% in the ratio of computation to *actual* communication. This happens because communication bandwidth is required not only for explicit data transfers, but also implicitly to fetch computation inputs (cache misses) and to store computation outputs (cache write-backs or write-throughs). For example, on the 21-bit NoC mesh with 1 kB caches, we observe that in the **multi-low** case, computation is typically 50 to 60% of the application kernel's simulation time, while in **multi-high** the fraction of computation time ranges between 70% to 85% - obviously larger, but less so than what could be expected.

The second key to understand the behaviour of **multi-high** *vs.* **multi-low**, and more in general the reasons for the performance advantage of NoCs, is the difference between bandwidth and latency.

One important factor that contributes to the NoC speed results is certainly their peak bandwidth. Given the clock frequencies above, the overall bandwidth of the fabrics can be calculated. The NoC meshes have 44 links, for an aggregate bandwidth of about 87 GB/s (21-bit mesh) or even 158 GB/s (38-bit flits). The custom topology, which is specifically optimized, only features 14 links and therefore has around 28 GB/s of bandwidth. The ML AMBA topology can have at most five pending transactions at a time; considering two sets of 32-bit data wires (AMBA features dual data channels for reads and writes), 32-bit address wires and a dozen used control wires, the available bandwidth can be computed to be around 24 GB/s.

Therefore, the NoC meshes feature about 3.5 to 6.5 times more peak bandwidth than the ML AMBA topology. This seems to explain the performance gap. However, the NoC custom topology still outperforms ML AMBA with just 28 GB/s of peak bandwidth - a 15% margin.

This is due to the fact that bandwidth is only an indirect clue of performance; the real metric to assess the speed of an interconnect is the latency from request to completion of a transaction. In Fig. 2.7, we depict the average latencies for various types of transactions in **multi-high** (other benchmarks show similar trends) as seen by the ARM7 cores. For single reads, the packeting overhead of the NoC is clear; the ML AMBA topology is about twice as fast in returning responses. Indeed, it is possible to analytically and empirically observe that the ML AMBA design can internally complete a single read in 5 CPU clock cycles in the best case (with one-wait-state memories), while 10 CPU clock cycles are needed for the NoC in the 38-bit configuration with a 2X clock multiplier. The same does not hold for burst transfers, where the packeting overhead is only paid once, and congestion becomes the key limiter: even though the burst traffic in our case is mostly composed of short 4-beat cache refills (the traffic generators inject a smaller amount of 8-beat reads), the NoCs come out faster, with a margin of 10 to 20% (the 38-bit topology performing even better at 20 to 25%, due to lower congestion). This result strongly suggests that NoCs should try to take advantage of stream transfers. The single write latency figure is also interesting; in this case, the NoC shows, on average, less than half the latency of the ML AMBA scheme. This figure is the result of the support for posted writes in the OCP protocol, which is exploited by ×pipes . ×pipes allows streams of writes to be issued in a posted fashion, without any delays except those possibly introduced by eventual buffer saturation somewhere in the network. In contrast, the AMBA protocol forces a master to wait for the response to the previous write request and for rearbitration before issuing a new write; therefore, write streams experience continuous hiccups. This phenomenon could be bypassed by interposing data FIFOs, but this kind of optimizations is beyond the scope of this paper.

The overwhelming bandwidth advantage of the NoC meshes is a hint to overdesign, and explains our choice of presenting a custom NoC topology that is specifically tailored for the benchmarks under scrutiny. The purpose is not to contrast this topology against the AMBA fabric, which would be unfair, but to show how significant the savings that derive from custom mapping can be. The custom topology is much less bandwidth-rich than the meshes, noticeably trimming power consumption while not affecting fabric speed nearly as much.

Coming back to the analysis of benchmark execution times, as shown

above, our results show similar performance gains of NoCs *vs.* AMBA in **multi-low** and **multi-high**, despite the fact that **multi-low** spends about 20% more time in communication than **multi-high**, and should therefore exhibit larger speedups. On the other hand, **des** requires heavy communication resources and indeed strongly benefits from NoCs. Given the discussion above, the mix of transaction types can clearly explain the results. For example, relatively to the **multi-high** benchmark when run on the 21-bit mesh with 4 kB caches, one can observe 26% of single reads, 1% of burst reads (very few cache misses) and 73% of single writes. In the same setup, **multi-low** exhibits 46%, 2% and 52%. So, while **multi-low** spends more time in communication, its communication mix is less favourable to the NoCs than that of **multi-high**, producing similar overall results. **des** not only demands a lot from the interconnect, but it is also a good match to NoC architectures; due to a much larger data set and code segment, in the same setup, the ratios are respectively 16% (few single reads), 32% (many refills) and 52%.

The mix of transactions also depends on other parameters such as cache sizes or cache locality of the application; larger caches decrease cache refills (burst reads), but do not affect inter-processor communication, which is not cacheable. For example, **multi-high** run on 256 B caches generates as much as 33% of burst reads for cache refills, compared to 1%, as seen above, in a 4 kB setup. Further, many other factors can contribute to performance results, including functional bottlenecks (one slave is heavily accessed and slows down the whole system), localized congestion (the topology suffers from overload at some location), traffic spikes over time (resources are normally underutilized, but communication spikes occur and when they occur they are poorly handled). Overall, the results show a noticeable performance lead of NoCs over a wide range of transaction patterns.

2.5.2 Interconnect Area, Frequency of Operation and Bandwidth

Screenshots of the layouts for ML AMBA AHB and for the NoC topologies are shown in Fig. 2.8. Here and elsewhere in this Section, the shared bus configuration is omitted because, as shown (Section 2.5.1), the performance of the fabric is so bad as to make any comparison pointless.

The first analysis reported is the one related to the area occupation of the topologies under test. As a premise, it must be stated that the present study mostly focuses on performance and power. Thus, only few iterations of a specific optimization step in the synthesis flow were performed to minimize area requirements. In fact, a placement step to derive fabric floorplans was

performed, but this is only done to get a realistic estimation of capacitive overheads due to long wires. To put more effort in the step of tightly compacting the design, which would be needed in an industrial product but is unneeded for our characterization, was avoided for the sake of this analysis, and can be considered to point out the need for an efficient automatic NoC flooplanner. This argument will be discussed more in detail in the following chapters, while considering the benefits derived by a tool aimed to the application-specific NoC design. For these reasons, the overall floorplan areas which can be inferred from Fig. 2.8 are not competely meaningful, except as a way to get information about wire lengths. In fact, since the floorplans have not been intensively iteratively tightened, floorplan areas only represent the bounds manually set for the placement&routing tools.

Still, the cell area metric, which only takes into account the area occupation of logic cells in the design, is a useful indication about the expected silicon overhead of alternative fabrics. After placement&routing, and including the clock tree buffers, the cell area for the AMBA ML topology is 0.52 mm^2. For the ×pipes meshes, which feature 15 switches and 30 NIs, area is 1.7 mm^2 (21-bit design) to 2.1 mm^2 (38-bit design), while the custom topology comprises fewer switches and is therefore a bit less demanding at about 1.5 mm^2. While these results show a large relative overhead for the NoCs *vs.* AMBA, the overhead is in fact small when compared to the area requirements of the IP cores.

Max Frequency	**ML AMBA**	**×pipes mesh (21-bit)**	**×pipes mesh (38-bit)**	**×pipes custom (21-bit)**
After synthesis	480 MHz	847 MHz	847 MHz	847 MHz
After place&route	370 MHz	793 MHz	793 MHz	793 MHz
Frequency drop	22.9%	6.4%	6.4%	6.4%

Table 2.1: Pre- and post-placement achievable frequencies

The maximum frequency results for the fabrics are as reported in Table 2.1. The ML AMBA fabric reaches at most 480 MHz before the placement stage. After placement and clock tree insertion, the actual achievable frequency decreases sharply to 370 MHz (-22.9%). This drop means that, compared to the design netlist, some unexpected capacitive loads arise in the final floorplan due to routing constraints. An explanation can be found in the purely combinational nature of the fabric, which implies long wire propagation times and compounds the delay of the crossbar block.

As can be seen, the ×pipes topologies all achieve much higher clock frequencies. Even after placement&routing, the critical path is not on the

NoC links, which confirms that the wire segmentation is highly effective. A byproduct is wire load predictability; in fact, as the table shows, the NoC fabrics suffer a minimal timing penalty of 6.4% after taking into account actual capacitive loads. These results suggest better scalability of the NoC architecture to future technology nodes.

It is meaningful to underline the effect that clock gating and clock tree deployment have on the design of a complex architecture. With respect to some initial results [4] on cross-benchmarking, where these elements were not accounted for, it is for example possible to notice that the maximum frequency achievable by NoCs drops by almost 100 MHz (885 MHz *vs.* 793 MHz). This is easily explained; signals need to travel from flip-flop to flip-flop within a time budget of one clock period, but the clock management logic adds delay and skew, both of which cut into the available timing window. This result, while certainly not novel, is further highlighting the importance of a complete modeling and synthesis flow spanning up to the lowest levels of abstraction.

2.5.3 Interconnect Power and Energy

To attempt a power evaluation, in a first phase, the activity during functional system simulations was monitored and all source-target transaction pairs were logged. Then the flow outlined in Section 2.4.2 was set up by injecting traffic from master ports towards each of the targets which are accessed in the functional simulation. We only analyze average power and energy consumption figures under specific workloads. In chapter *fix reference to chapter about power modeling* methods to build flexible yet accurate power models for ×pipes architecture are presented.

From preliminary analysis, sequential logic resulted to represent by far the largest fraction of power consumption in the ×pipes NoC, with flip-flops burning as much as 80% of the global power requirements (still excluding the clock tree contribution, which is also major). This is contrary to some previous assumptions, where switching activity on global wires is expected to be the dominant contribution to power dissipation (an example is in [1]). The balance is expected to eventually shift in smaller technologies, but at the 0.13 μm node sequential elements seem to be the prime candidates for tuning. This observation leads to optimize the NoC by means of several strategies related to buffering elements. So, first, the implementation of clock gating achieved about 30% power savings. Second, the buffering resources are kept to a minimum across the NoC, by sizing NI and switch buffers to hold only three flits at a time. Third, the flit width degree of freedom was explored, proving its usefulness: moving from 38-bit to 21-bit flits reduces buffer size

almost in half, cutting power by a significant amount (see below). The aspects related to the optimal flit width are discussed more in chapter *Fix the reference with the chapter about cache performances* The power results obtained for the topologies at their maximum operating frequency are reported in Table 2.2, while energy results are reported in Table 2.3. ×pipes figures are for designs with clock tree and clock gating, while in the case of AMBA, only insert a clock tree is present; given the low amount of sequential logic that AMBA contains (see Table 2.2), clock gating would offer negligible benefits and unnecessarily add design flow complexity and frequency penalties.

It is interesting to note that, with the used 0.13 μm technology library, leakage power is present but almost negligible (less than 0.1%). This is also expected to change in smaller technologies.

Power Consumption	**ML AMBA**	**×pipes mesh (21-bit)**	**×pipes mesh (38-bit)**	**×pipes custom (21-bit)**
Global power	75 mW	376 mW	473 mW	347 mW
Sequential cell power	15 mW	145 mW	187 mW	116 mW
Sequential power ratio	20.5%	38.6%	39.5%	33.4%

Table 2.2: Power consumption of the fabrics

Energy Consumption	**ML AMBA**	**×pipes mesh (21-bit)**	**×pipes mesh (38-bit)**	**×pipes custom (21-bit)**
Energy per injectable data	3.13 mJ/GB	4.32 mJ/GB	2.99 mJ/GB	12.39 mJ/GB
Energy per benchmark run (fabric only)	0.075 mJ	0.338 mJ	0.402 mJ	0.312 mJ
Energy per benchmark run (1W system)	1.08 mJ	1.30 mJ	1.31 mJ	1.28 mJ
Energy per benchmark run (5W system)	5.08 mJ	5.17 mJ	4.96 mJ	5.14 mJ

Table 2.3: Energy consumption of the fabrics

The ML AMBA crossbar is clearly the winner in terms of pure power consumption. The power consumption of the NoC meshes is 5.6 to 7.5 times higher. Keeping the flit width of the NoC mesh low is however helpful, as power savings of 25% can be noticed, with a much lower impact on overall performance (see Section 2.5.1). Thanks to clock gating, the fraction of power consumption due to sequential logic drops significantly, from the initial value

of around 80% [4] obtained in preliminary analysisto around 35%. This drop is due to the compound effect of clock gating (which cuts the sequential power requirements by 30%) and of the clock tree insertion, which is implemented by means of combinational cells, thus lowering the relative fraction of sequential power. The custom NoC topology, thanks both to its lower switch count, is able to shave about 8% off the power of the 21-bit mesh. When considering the power density of the interconnects, AMBA has an advantage of roughly a factor of two; this can be attributed to the difference in clock speeds.

In terms of energy, the advantage of ML AMBA is less clear. When considering the ratio among power consumption and available bandwidth (mW over GB/s, or mJ over GB of injectable data), ML AMBA and the NoC meshes look much closer. However, this figure is a bit misleading; using all of the available bandwidth, the meshes would indeed consume much more energy. Further, the custom NoC, which is designed around providing bandwidth only where necessary, but utilizing it efficiently, is unreasonably penalized by this analysis. Therefore, it may be useful to define a more meaningful metric: power over benchmark execution time, *i.e.* the energy required to complete a benchmark. Given the performance figures shown by the experiments, an execution time advantage as shown by **multi-high** or **multi-low** is conservatively assumed; in a **des**-like scenario, of course, the results of NoCs would look better. Thus, an execution time gain of 10% for the 21-bit NoCs (mesh and custom) against the ML AMBA fabric, and of 15% in the case of the 38-bit NoC mesh was set. Calculating the energy consumption over an execution time which is 1 *ms* for the baseline ML AMBA case, it was possible to obtain the results reported in the second row of Table 2.3. To derive an even more useful metric, however, the energy consumption of the whole system should be taken into account. To this effect, the power consumption of other parts of the system must be modeled. This is a very difficult task, as it greatly depends on the specific components at hand. It is correct to very conservatively assume a power consumption of just 1 W at 370 MHz for all of the 15 cores, caches and memory blocks. Further, a 370 MHz working frequency for the cores in the ML AMBA case and a 396.5 MHz frequency for the NoCs (Section 2.5.1) can be aasumed. The overall power consumption of the systems would therefore be 1.075 W for ML AMBA, 1.449 W for the 21-bit NoC, and so on. The corresponding energy is reported in the third row of Table 2.3; the three NoCs are giving almost identical results, about 20% worse than ML AMBA. With system components requiring 5 W, however, the NoCs become strongly competitive, as the table shows; the 21-bit NoCs get almost on par with ML AMBA, while the 38-bit topology actually offsets its higher power requirements with its performance results, coming out as the most energy-efficient by a slight

margin.

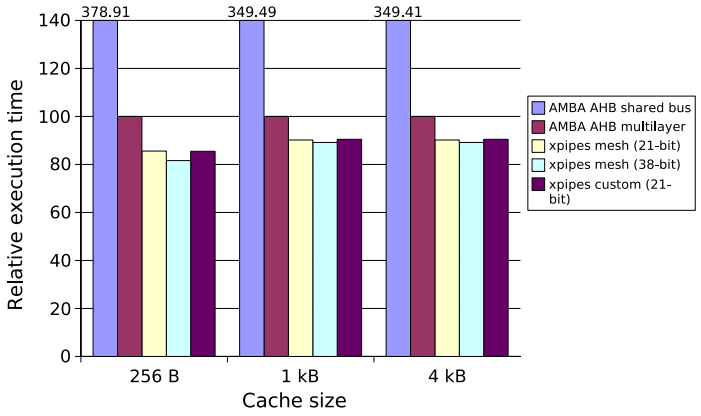
2.5.4 Split Analysis of Area and Power Contributions

In Fig. 2.9, a split report of area occupation and power consumption for the three NoCs is presented. In terms of area (Fig. 2.9(a)- 2.9(c)), at first glance, three main contributions can be noticed: switches, initiator NIs and target NIs. However, the ratios between them can shift in a relevant fashion. To understand the figures, it must be noticed that the mesh topologies feature 15 instances of each type of component, while the custom NoC has 15 of each type of NIs but only 8 switches. The absolute contribution of the NIs to the NoC area remains roughly constant across the topologies; NI area is found to be dominated by the OCP front-end, which remains unchanged regardless of the topologies. The main differences are therefore due to switch area. Taking the 21-bit mesh as a baseline, switches require 38.5% of the NoC area. In the 38-bit mesh (where switches are larger due to the increased flit width) the percentage rises to 47.7%, while in the 21-bit custom topology (where there are fewer switches) it falls to 27.0%.

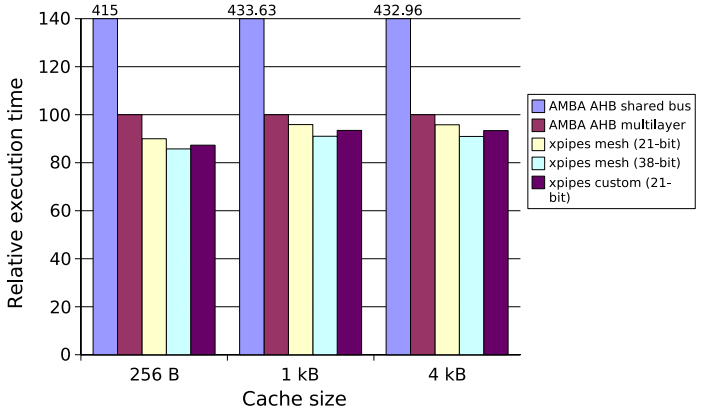
In terms of power (Fig. 2.9(d)- 2.9(f)), the major contribution, as expected, is due to the clock distribution network. Two clocks are actually being distributed, a fast one for the network and a slower one for the OCP front end of the NIs. The two clock trees together burn 40% or more of the overall power. Several interesting trends can be observed in this split analysis. For example, since the absolute power consumption of the NIs remains more or less constant across the topologies, their relative consumption is determined by the other components. The ×pipes clock tree, *i.e.* the fast one, has a consumption which is very directly correlated to the amount of flip-flops it must drive; therefore, it takes the smallest fraction of the power budget in the 21-bit custom topology (where there are fewer switches to clock), a larger one in the 21-bit mesh, and the largest one in the 38-bit mesh. Switches themselves exhibit a more complex trend. They already burn a significant amount of power in the 21-bit mesh, and this budget increases even more in the 38-bit mesh and in the custom topology. The reasons are different; in the former case, there are simply more gates (38-bit switches have datapaths which are almost twice as wide), while in the latter, the amount of gates is actually lower (8 switches instead of 15) but traffic is still efficiently processed, which points to a much higher switching activity.

Finally, it is interesting to note that, in all cases, when comparing the switches to the NIs, switches take a larger fraction of the power budget than they do of the area budget. For example, in the most extreme case, the custom topology, the switches require less area than either the initiator or

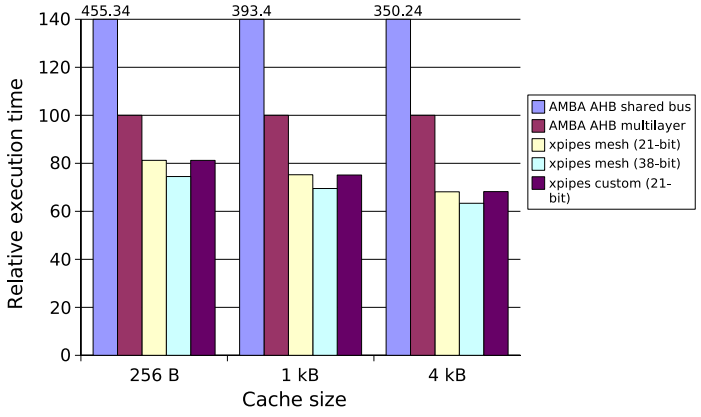
target NIs, but burn as much power as both types of NIs together. We mostly attribute this fact to NI front ends working at the OCP clock frequency, *i.e.* at half the frequency of the rest of the network (Section 2.5.1), while switches uniformly run at the higher frequency. Further, switches experience a higher average activity level, since for example a single incoming packet forces all output ports of a switch to evaluate a new request - even if a single output port will eventually let flits through.



(a)

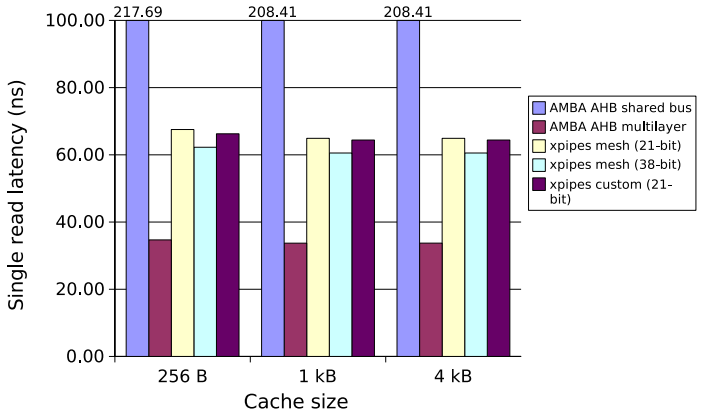


(b)

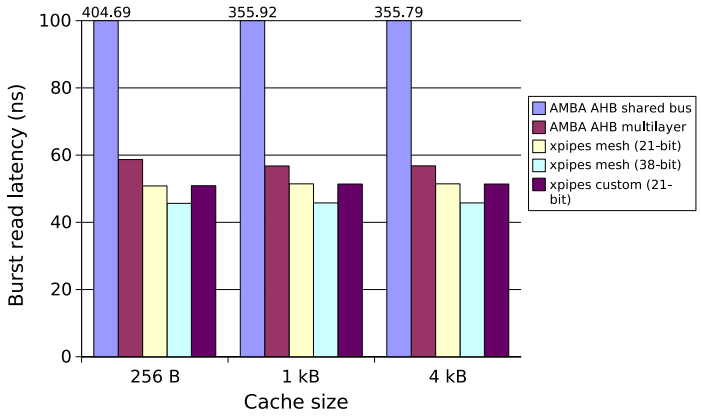


(c)

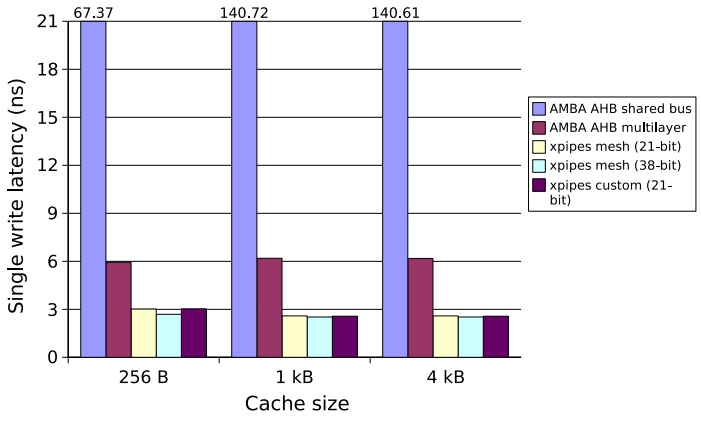
Figure 2.6: Relative execution times for (a) **multi-high**, (b) **multi-low**, (c) **des** with varying cache sizes. The ML AMBA AHB execution time is the baseline at 100. AMBA AHB shared bus results lay beyond the upper limit of the Y axis scale



(a)



(b)



(c)

Figure 2.7: Latency of (a) single read, (b) burst read, (c) single write transfers on the interconnects. AMBA AHB shared bus results lay beyond the upper limit of the Y axis scale

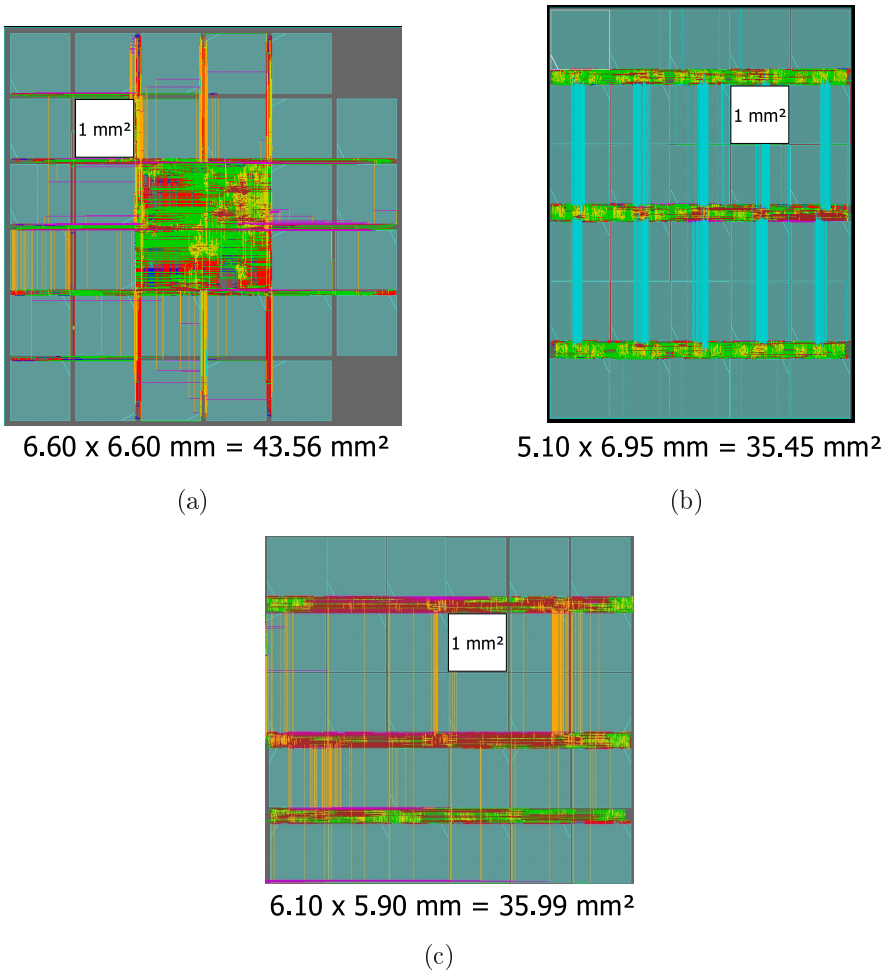


Figure 2.8: Layouts of our test fabrics: (a) ML AMBA, (b) ×pipes meshes, (c) ×pipes custom

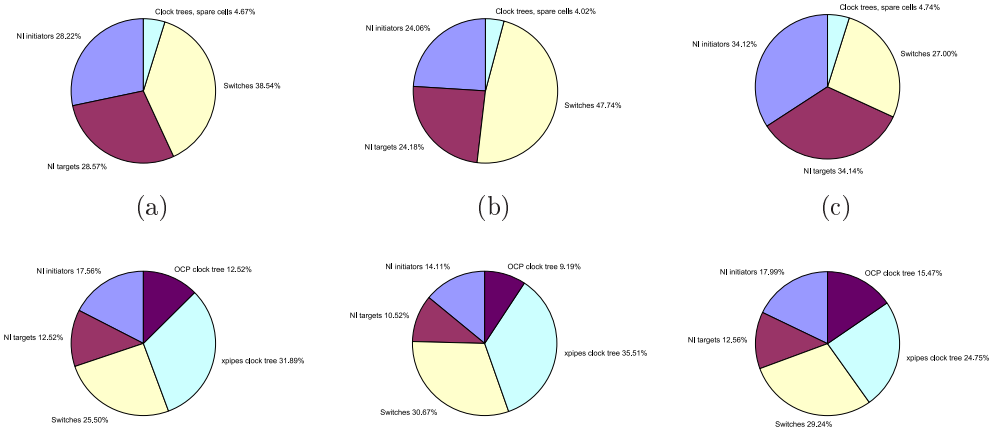


Figure 2.9: Split report for a ×pipes topology: (a-c) area of the 21-bit mesh, the 38-bit mesh and the 21-bit custom NoC; (d-f) power consumption of the three topologies

Bibliography

[1] Tapani Ahonen, David A. Sigüenza-Tortosa, Hong Bin, and Jari Nurmi. Topology optimization for application-specific networks-on-chip. In *Proceedings of the 6th International Workshop on System Level Interconnect Prediction (SLIP04)*, pages 53–60, 2004.

[2] Adrijean Andriahantenaina and Alain Greiner. Micro-network for SoC: Implementation of a 32-port SPIN network. In *The Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1128– 1129. IEEE, 2003.

[3] Federico Angiolini, Paolo Meloni, Davide Bertozzi, Luca Benini, Salvatore Carta, and Luigi Raffo. Networks on chips: A synthesis perspective. In *Proceedings of the 2005 ParCo Conference (to be published)*, 2005.

[4] Federico Angiolini, Paolo Meloni, Salvatore Carta, Luca Benini, and Luigi Raffo. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference and Exhibition*, pages 124–129, 2006.

[5] ARM Ltd. The Advanced Microcontroller Bus Architecture (AMBA) homepage. www.arm.com/products/solutions/AMBAHomePage.html.

[6] ARM Ltd. PrimeXsys platforms. www.arm.com.

[7] Luca Benini. Application specific NoC design. In *Proceedings of the 2006 Design, Automation and Test in Europe Conference (DATE)*, pages 491–495, 2006.

[8] Luca Benini and Giovanni De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70 – 78, January 2002.

[9] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna R. Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16, Issue 2:113–129, February 2005.

[10] Tobias Bjerregaard and Jens Sparsø. Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 34–43, 2005.

[11] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. QNoC: QoS architecture and design process for network on chip. In *Journal of Systems Architecture*. Elsevier, 2004.

[12] Cadence Design Systems Inc. SoC Encounter. www.cadence.com.

[13] R.Y. Chen, N. Vijaykrishnan, and Mary Jane Irwin. Clock power issues in System-on-a-Chip designs. In *Proceedings of the 1999 IEEE Workshop on Very Large Scale Integration (VLSI)*, pages 48–53, 1999.

[14] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, pages 684–689, June 2001.

[15] Monica Donno, Alessandro Ivaldi, Luca Benini, and Enrico Macii. Clock-tree power optimization based on RTL clock-gating. In *Proceedings of the Design Automation Conference (DAC)*, pages 622–627, 2003.

[16] Faraydon Karim, Anh Nguyen, Sujit Dey, and Ramesh Rao. On-chip communication architecture for OC-768 network processors. In *Proceedings of the Design Automation Conference (DAC)*, pages 678 – 683, 2001.

[17] Kangmin Lee, Se-Joong Lee, Sung-Eun Kim, Hye-Mi Choi, Donghyun Kim, Sunyoung Kim, Min-Wuk Lee, and Hoi-Jun Yoo. A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform. In *Digest of Technical Papers of the 2004 IEEE International Solid-State Circuits Conference (ISSC)*, pages 152–518. IEEE Computer Society, 2004.

[18] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, and Roberto Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference (DATE)*. IEEE, 2004.

[19] Real-Time Operating System for Multiprocessor Systems (RTEMS). www.rtems.com.

[20] Antonio Pullini, Federico Angiolini, Davide Bertozzi, and Luca Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design (SBCCI)*, pages 224–229, 2005.

[21] Andrei Radulescu, John Dielissen, Kees Goossens, Edwin Rijpkema, and Paul Wielage. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference (DATE)*. IEEE, 2004.

[22] Gilles Sassatelli, Séverine Riso, Lionel Torres, Michel Robert, and Fernando Moraes. Packet-switching network-on-chip features exploration and characterization. In *Proceedings of the International Conference on Very Large Scale Integration System-on-Chip (VLSI-SoC 2005)*, pages 403–408, 2005.

[23] Stergios Stergiou, Federico Angiolini, Salvatore Carta, Luigi Raffo, Davide Bertozzi, and Giovanni De Micheli. ×pipes Lite: A synthesis oriented design library for networks on chips. In *Proceedings of the 2005 Design, Automation and Test in Europe Conference (DATE)*, pages 1188–1193. IEEE, 2005.

[24] STMicroelectronics. The STBus interconnect. www.st.com.

[25] Synopsys Inc. coreTools. www.synopsys.com.

[26] Synopsys Inc. Design Compiler. www.synopsys.com.

[27] Synopsys Inc. PrimePower. www.synopsys.com.

[28] M.D. Taylor, W. Lee, S.P. Amarasinghe, and A. Agarwal. Scalar operand networks. *IEEE Transactions on Parallel and Distributed Systems*, 16:145–162, 2005.

[29] Daniel Wiklund and Dake Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS03)*. IEEE, 2003.

[30] Drew Wingard. Micronetwork-based integration for SoCs. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 673–677. ACM, June 2001.

[31] Jiang Xu, Wayne Wolf, Joerg Henkel, and Srimat Chakradhar. A methodology for design, modeling, and analysis of Networks-on-Chip. In *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1778–1781. IEEE Computer Society, 2005.

[32] Cesar Albenes Zeferino and Altamiro Amadeu Susin. SoCIN: A parametric and scalable network-on-chip. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI03)*, pages 34–43, 2003.

Chapter 3

Designing Application-Specific Networks on Chips

For the use of NoCs to be feasible in today's industrial designs, a custom-tailored, application-specific NoC that satisfies the design objectives and constraints of the targeted application domain is required.

3.1 Introduction

Taking into account the comparative analysis reported in the previous chapter, a major reason for the delay for NoCs to appear in industrial designs can be individuated in the fact that, even though they provide large throughput and flexibility, the power consumption and latency of the design is quite high, when compared to today's bus based systems. Thus a huge research field is open, aiming to optimize the NoC hardware costs while keeping the benefits in performances shown by preliminary evaluations. The optimization challenges encountered in the design of on-chip networks for SoCs are quite different from the design of macro-networks such as the interconnection networks used in parallel processing and the Internet. Some major differences are:

- The communication between the various cores can be statically analyzed for many SoCs and the NoC can be tailored for the particular application behavior
- The design objectives and constraints are different. As most SoCs are used in mobile and hand-held devices, having a network with minimum power consumption becomes an important design objective. Many SoCs also need to respond in real-time for certain inputs, for which

the NoC has to support different criticality levels for the different traffic streams.

- The design process should also consider VLSI issues, such as the structure and wiring complexity of the resulting interconnect.

A very important phase in NoC design, having a deep impact on the area occupation, latency and power consumption shown by the designed NoC, is the synthesis of the topology (*i.e.* the definition of the structure of the network and the setting of various design parameters , such as frequency of operation or link-width). It must be noticed that standard topologies (mesh, torus, etc.) providing uniform and regular connectivity between the cores are required for on-chip systems where the traffic characteristics of the system cannot be predicted statically, as in chip-multiprocessors. However, for most SoCs the system is designed with static (or semi-static) mapping of tasks to processors and hardware cores and hence the communication traffic characteristics of the SoC can be obtained statically. This is true from SoC designs that are small to state-of-the art SoCs, such as, the Philips Nexperia platform [1], ST Nomadik [2], TI OMAP [3], etc. In this cases, very frequent indeed, it would be very useful to design the best topology that is tailor-made for a specific application and satisfies the communication constraints of the design. Being customly designed for a given use-case, the NoC designed according to this approach would be able to save area, power and latency allocating the hardware resources needed to provide fast communication only where needed. Another motivation that brings the SoC designers to the use of a NoC is the fact that the interconnect structure and wiring complexity can be well controlled. When the interconnect is structured, the number of timing violations that occur during the physical design (floorplanning and wire routing) phase is minimum. Such design predictability is critical for today's SoCs for achieving timing closure. It leads to faster design cycle, reduction in the number of design re-spins and faster time-to-market. As the wire delay as a fraction of gate delay is increasing with each technological generation, having shorter wires is even more important for future SoCs. An application-specific NoC with structured wiring, which satisfies the design objectives and constraints is important to have feasible NoC designs.

A large body of research works exists in synthesizing and generating bus-based systems [9]- [14]. A floorplan-aware point-to-point link design and bus design methodologies are presented in [15] and [14]. Anyway, while some of the design issues in the NoCs are similar to bus based systems (such as link-width sizing), a large number of issues such as finding the number of required switches, sizing the switches, finding routes for packets, etc. are new in NoCs. Methods to collect and analyze traffic information that can

be fed as input to the bus and NoC design processes have been presented in [12] and [13]. Mappings of cores onto standard NoC topologies have been explored in [16]- [19]. In [17], [19] a floorplanner is used during the mapping process to get area and wire-length estimates. Unlike the method presented here, these works only select topologies from a library of standard topologies. In [18], a unified approach to mapping, routing and resource reservation has been presented. However, the work does not explore topology design process. The NoC design process for supporting multiple applications has been presented in [20]. This research complements what was previously stated about application-specific NoC and its methods can be applied here to support multiple applications as well. Important research in macro-networks has considered the topology generation problem [21]. As the traffic patterns on these networks are difficult to predict, most approaches are tree-based (like spanning or Steiner trees) and only ensure connectivity with node degree constraints [21]. Hence, these techniques cannot be directly extended to address the NoC synthesis problem. Application-specific custom topology design has been explored in [22]- [25]. The works from [22], [23] do not consider the floorplanning information during the topology design process. In [24], a physical planner is used during topology design to reduce power consumption on wires. However, the work does not consider the area and power consumption of the NoC modules in the design, thus, taking into account the split analysis reported in the previous chapter, does not include in the optimization process the biggest part of the interconnect power consumption. Also, the number and size of network partitions are manually fed. Several works exist on automatically generating the *Register Transfer Level (RTL)* code of a designed topology for simulation and synthesis (*e.g.* [28]). These works again complement application-driven NoC design, as the input to them is a synthesized topology. Building area, power models for on-chip networks has been addressed in [29]- [32].

3.2 Design Flow

In this thesis, as mentioned, some design techniques and optimizations supporting application-aware NoC design are presented. In following chapters this methods are presented more in detail. Anyway, all of them are thought as parts of a novel streamlined design methodology for NoC topology synthesis developed in collaboration with the research partners. The developed methodology should bring the designer from the system high-level specification to the back-end physical design (in particular this final part of the design flow is completely integrated with the state-of-the commercial tools

and was already discussed in section 2.4). The presented tool flow automates the entire NoC design process, including:

- performance and power consumption aware topology synthesis (integrated with NoC architectural parameter setting and compliant with accurate switch area, power models and link power models that are obtained from layouts of the components),
- routing and path computation (providing deadlock-free network operation is provided without special hardware mechanisms),
- RTL code generation and layout generation; thereby bridging an important gap in the design of application-specific NoCs.

The developed NoCs design flow is presented in Figure 3.1.

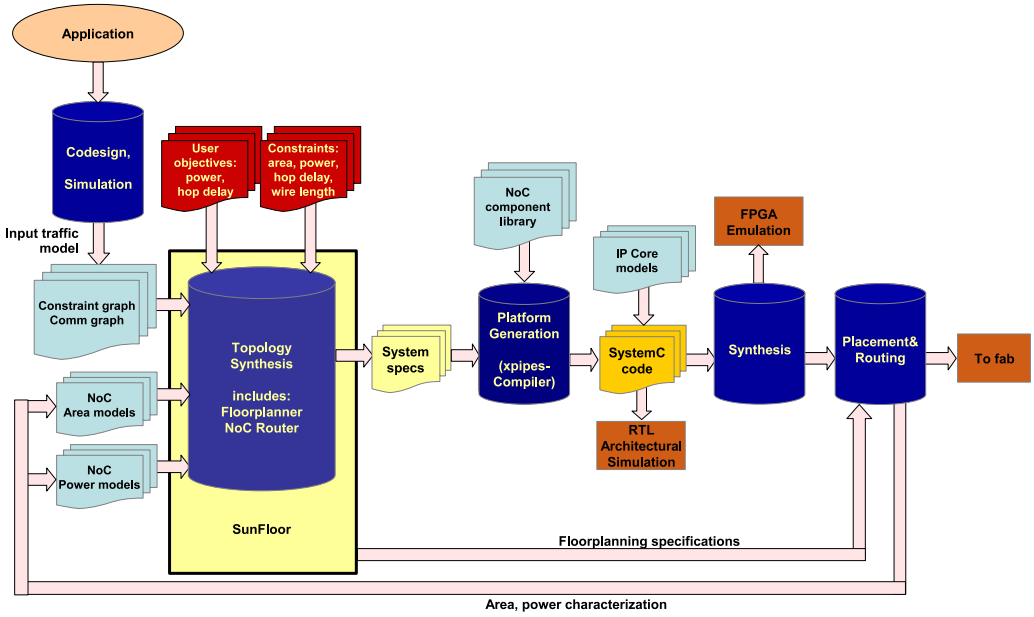


Figure 3.1: NoC Design Flow

- In the first phase, the user specifies the objectives and constraints that should be satisfied by the NoC. The application traffic characteristics, size of the cores, and the area and power models for the network components are also obtained.
- The second phase of the flow consists basically in the use of an homemade tool named SunFloor, which is explained in detail as reference

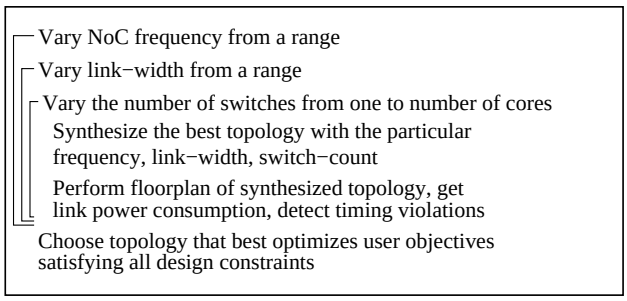


Figure 3.2: Sunfloor NoC architecture synthesis (Second phase of the design flow)

in this chapter. By means of SunFloor the NoC architecture that optimizes the user objectives and satisfies the design constraints is automatically synthesized. The topology design process supports two objective functions: minimizing network power consumption and hop-count for data transfer. The designer can optimize for one of the two objectives or a linear combination of both. The topology design process supports constraints on several parameters such as the hop-count (when the objective is power minimization), network power consumption (when the objective is hop-count minimization), design area and total wire-length. The topology synthesis process uses a floorplanner to estimate the design area and wire-lengths. The wire-length estimates from the floorplan are used to evaluate whether the designed NoC satisfies the target frequency of operation and to compute the power consumption of the wires. The different steps in this phase are presented in Figure 3.2. In the outer iterations, the key NoC architectural parameters (NoC frequency of operation and link-width) are varied in a set of suitable values. The bandwidth available on each NoC link is the product of the NoC frequency and the link-width. During the topology synthesis, the algorithm ensures that the traffic on each link is less than or equal to its available bandwidth value. The synthesis step is performed once for each set of the architectural parameters. In this step, several topologies with different number of switches are explored, starting from a topology where all the cores are connected to one switch, to one where each core is connected to a separate switch. The synthesis of each topology includes finding the size of the switches, establishing the connectivity between the switches and connectivity with the cores, and finding deadlock-free routes for the different traffic flows.
In the next step, to have an accurate estimate of the design area and wire-lengths, the floorplanning of each synthesized topology is automatically performed. The floorplanning process finds the 2D position

of the cores and network components used in the design. For this, we use Parquet, a fast and accurate floorplanner [35]. Based on the frequency point and the obtained wire-lengths, the timing violations on the wires are detected and the power consumption on the links is obtained.
In the last step, from the set of all synthesized topologies and architectural parameter design points, the topology and the architectural configuration that best optimizes the user's objectives, satisfying all the design constraints is chosen. Thus, the output of phase 2 is the best application-specific NoC topology, its frequency of operation and the width of each link in the NoC.

- The last phase of the design (right side of Figure 3.1), basically consists in the reference back-end design flow presented in section 2.4. As already mentioned, the RTL (SystemC) code of the switches, network interfaces and links for the designed topology is automatically generated, tanking as input the mentioned ×pipes library [8], [34], a library of soft macros for the network components, and the associated tool ×pipesCompiler [26] to interconnect the network elements with the cores. At this phase, a synthesizable RTL design that can also be emulated on FPGA ca be obtained. From the floorplan specification of the designed topology, the synthesis engine automatically generates the inputs for placement&routing. The placement&routing of the design is performed using SoC Encounter [37] for obtaining the layout, including the global and detailed routing of wires. The output of this phase is a complete layout of the NoC design that can be sent to a foundary.

As the flow has several steps, it is important to close the design gap across the different steps. To ensure that the designed topology will satisfy the timing constraints after place&route, the wire-lengths are evaluated for detecting timing violations early in the design process, i.e. during the topology synthesis phase itself. To bridge the gap between the initial traffic models and the actual observed traffic after simulating the designed NoC, a *mismatch* parameter is used. The parameter is read as part of the input specifications by the topology synthesis engine. The user can manually tune the parameter and re-design the NoC to suit the actual traffic characteristics. Several other options are also supported by the topology synthesis engine, such as support for cores with fixed locations in the layout (due to pin/pad constraints). Due to lack of space, here we only present the major features of the synthesis process.

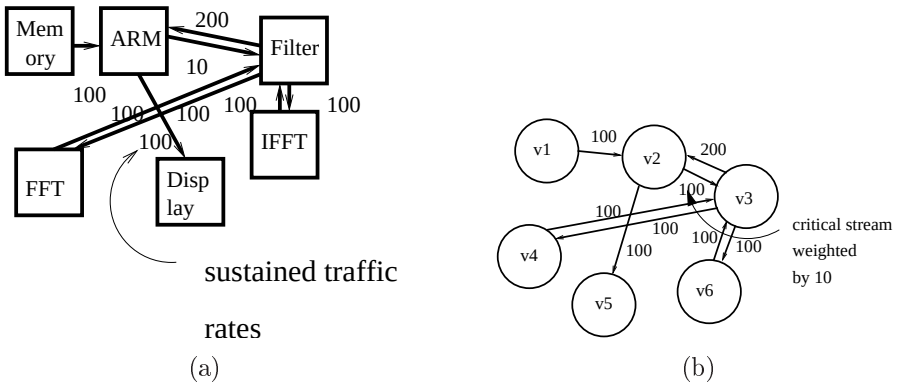


Figure 3.3: (a) Filter application (b) Core graph with sustained rates and critical streams

3.3 Input Models

The traffic characteristics of the application are represented by a graph [16], [17], [19], defined as follows:

Definition 1. *The core graph is a directed graph, $G(V, E)$ with each vertex $v_i \in V$ representing a core and the directed edge (v_i, v_j), denoted as $e_{i,j} \in E$, representing the communication between the cores v_i and v_j. The weight of the edge $e_{i,j}$, denoted by $comm_{i,j}$, represents the sustained rate of traffic flow from v_i to v_j weighted by the criticality of the communication. The set F represents the set of all traffic flows, with value of each flow, f_k, $\forall k \in 1 \cdots |F|$, representing the sustained rate of flow between the source (s_k) and destination (d_k) vertices of the flow.*

The core graph for a small filter example (Figure 3.3(a)) is shown in Figure 3.3(b). The edges of the core graph are annotated with the sustained rate of traffic flow, multiplied by the criticality level of the flow, as done in [19].

As mentioned, one of the research topics presented in this thesis, aimed to support application-specific NoC design, was the definition of accurate analytical models for the power consumption and area of the network components, based on the ×pipes architecture [8]. In particular, two different methodologies were studied and tested on the reference architecture. A detailed description of the two methodologies and the discussion about their accuracy and costs in terms of computational and manual effort is reported in chapter *fix the reference to chapter models* Briefly, to get the power estimates, the place&route of the components is performed using SoC Encounter and

Table 3.1: Component Area-Power

Component	Parameter	Analytical	Experimental
4x4 switch	area(mm^2)	0.036	0.035
	power(mW)	22.16	22.54
5x5 switch	area(mm^2)	0.048	0.047
	power(mW)	28.38	28.70
link (2mm)	power(mW)	0.57	0.57

accurate wire capacitances and resistances are obtained, as back-annotated information from the layout, with 0.13μm technology library. The switching activity in the network components is varied by injecting functional traffic. The capacitance, resistance and the switching activity report are combined to estimate power consumption using Synopsys PrimePower [38].

Power consumption on the wires is also obtained at the layout level. The analytical and experimental area, power consumption values for some components (with 900 MHz frequency, link-width of 32 bits, buffer depth of 3 in the switches) are presented in Table 3.1.

3.4 Design Algorithms

The algorithms for the topology design process are explained in this section. In the first step of Algorithm 1, a design point θ is chosen from the set of available or interesting design points ϕ for the NoC architectural parameters. In the current implementation, the synthesis engine automatically tunes two critical NoC parameters: operating frequency ($freq_\theta$) and link-width (lw_θ). As both frequency and link-width parameters can take a large set of values, considering all possible combinations of values would be infeasible to explore. The system designer has to trim down the exploration space and give the interesting design points for the parameters. The designer usually has knowledge of the range of these parameters. As an example, the designer can choose the set of possible frequencies from minimum to a maximum value, with allowed frequency step sizes. Similarly, the link data widths can be set to multiples of 2, within a range (say from 16 bits to 128 bits). Thus, it is possible to get a discrete set of design points for ϕ, as done in [14]. In all the experiments that will be presented in this chapter, 8 frequency steps and 4 link-width steps are supported , providing 32 discrete design points in the set ϕ. The rest of the topology design process (steps 3-15 in Algorithm 1) is repeated for each design point in ϕ.

As the topology synthesis and mapping problem is NP-hard [22], efficient heuristics are used to synthesize the best topology for the design. For each

design point θ, the algorithm synthesizes topologies with different numbers of switches, starting from a design where all the cores are connected through one big switch until the design point where each core is connected to only one switch. The reason for synthesizing these many topologies is that it cannot be predicted beforehand whether a design with few bigger switches would be more power efficient than a design with more smaller switches. A larger switch has more power consumption than a smaller switch to support the same traffic, due to its bigger crossbar and arbiter and due to its higher number of buffering resources. On the other hand, in a design with many smaller switches, the packets may need to travel more hops to reach the destination. Thus, the total switching activity would be higher than a design with fewer hops, which can lead to higher power consumption. The relationship between switch power consumption and switch size obviosly depends strictly on the used technology process. For the chosen switch count i, the input core graph is partitioned into i min-cut partitions (step 3). The partitioning is done in such a way that the edges of the graph that are cut between the partitions have lower weights than the edges that are within a partition (refer to Figure 3.4(a)) and the number of vertices assigned to each partition is almost the same. Thus, those traffic flows with large bandwidth requirements or higher criticality level are assigned to the same partition and hence use the same switch for communication. Hence, the power consumption and the hop-count for such flows will be smaller than for the other flows that cross the partitions. For partitioning, an efficient hierarchical graph partitioning tool, named Chaco [36], was used. At this point, the communication traffic flows within a partition have been resolved. In steps 5-9, the connections between the switches are established to support the traffic flows across the partitions. In step 5, the *Switch Cost Graph (SCG)* is generated.

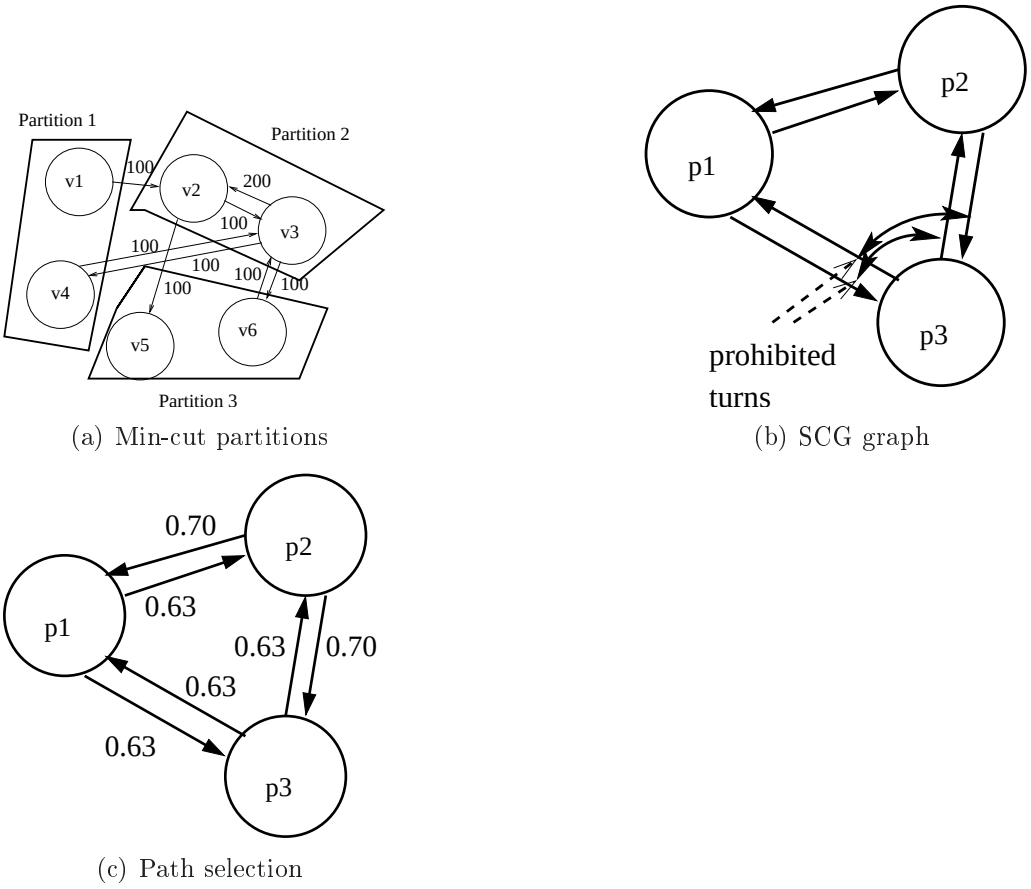
Definition 2. *The SCG is a fully connected graph with i vertices, where i is the number of partitions (or switches) in the current topology.*

SCG does not imply the actual physical connectivity between the different switches. The actual physical connectivity between the switches is established using the SCG in the *PATH_ COMPUTE* procedure, which is explained in the following paragraphs.

In NoCs, wormhole flow control [39] is usually employed to reduce switch buffering requirements and to provide low-latency communication [6], [7]. With wormhole flow control, deadlocks can happen during routing of packets due to cyclic dependencies of resources (such as buffers) [39]. The SCG is pre-processed to prohibit certain turns in order to break such cyclic dependencies. This guarantees that deadlocks will not occur when routing packets. For finding the set of turns that need to be prohibited to break cycles, the

turn prohibition algorithm presented in [33] is used, [18]. The algorithm has polynomial time complexity (very fast in practice, see Section 3.5) and guarantees that at most 1/3 of the total number of turns would be prohibited to remove cycles. The algorithm also guarantees connectivity between all nodes in the SCG after prohibiting the turns. From the algorithm, the *Prohibited Turn Set (PTS)* for the SCG is built, which represents the set of turns that are prohibited in the graph. To provide guaranteed deadlock freedom, any path for routing packets should not take these prohibited turns.

These concepts are illustrated in the following example:



(a) Min-cut partitions

(b) SCG graph

(c) Path selection

Figure 3.4: Algorithm examples

Example 3.4.1. *The min-cut partitions of the core graph of the filter example (from Figure 3.3(a)) for 3 partitions is shown in Figure 3.4(a). The SCG for the 3 partitions is shown in Figure 3.4(b). After applying the turn prohibition algorithm from [33], the set of prohibited turns is identified. In Figure 3.4(b), the prohibited turns are indicated by circular arcs in the SCG.*

For this example, both the turns around the vertex P3 are prohibited to break cycles. So no path that uses the switch P3 as an intermediate hop can be used for routing packets.

The topology synthesis process also supports freedom from another type of deadlock, known as message-level deadlock [39], by routing the traffic flows of the different message types in the design onto different physical links.

Algorithm 1 Topology Design Algorithm

1: Choose design point θ from ϕ: $freq_\theta$, lw_θ
2: **for** $i = 1$ to $|V|$ **do**
3: Find i min-cut partitions of the core graph
4: Establish a switch with N_j inputs and outputs for each partition, $\forall j \in 1 \cdots i$. N_j is the number of vertices (cores) in partition i. Check for bandwidth constraint violations.
5: Build *Switch Cost Graph (SCG)* with edge weights set to 0
6: Build *Prohibited Turn Set (PTS)* for SCG to avoid deadlocks
7: Set ρ to 0
8: Find paths for flows across the switches using function *PATH_ COMPUTE(i, SCG, ρ, PTS, θ)*
9: Evaluate the switch power consumption and average hop-count based on the selected paths
10: Repeat steps 8 and 9 by increasing ρ value in steps, until the hop-count constraints are satisfied or until ρ reaches ρ_{thresh}
11: If ρ_{thresh} reached and hop-count not satisfied, go to step 2.
12: Perform floorplan and obtain area, wire-lengths. Check for timing violations and evaluate power consumption on wires
13: If target frequency matches or exceeds $freq_\theta$, and satisfies all constraints, note the design point
14: **end for**
15: Repeat steps 2-14 for each design point available in θ
16: For the best topology and design point, generate information for ×pipesCompiler and Cadence SoC Encounter

The actual physical connections between the switches are established in step 8 of Algorithm 1 using the *PATH_ COMPUTE* procedure. The objective of the procedure is to establish physical links between the switches and to find paths for the traffic flows across the switches. Here, the procedure where the user's design objective is to minimize power consumption is presented as reference. The procedure for the other two cases (with hop-count as the objective and with linear combination of power and hop-count as objective) follow the same algorithm structure, but with different cost metrics.

Algorithm 2 PATH_COMPUTE(i, SCG, ρ, PTS, θ)

1: Initialize the set $PHY(i1, j1)$ to false and $Bw\_avail(i1, j1)$ to $freq_\theta \times lw\theta$, $\forall\ i1,\ j1 \in 1 \cdots i$
2: Initialize $switch\_size\_in(j)$ and $switch\_size\_out(j)$ to N_j, $\forall\ j \in 1 \cdots i$. Find $switching\_activity(j)$ for each switch, based on the traffic flow within the partition.
3: **for** each flow f_k, $k \in 1 \cdots |F|$ in decreasing order of f_c **do**
4: **for** i1 from 1 to i and j1 from 1 to i **do**
5: {Find the marginal cost of using link i1, j1}
6: {If physical link exists and can support the flow}
7: **if** $PHY(i1, j1)$ and $Bw\_avail(i1, j1) \geq f_c$ **then**
8: Find $cost(i1, j1)$, the marginal power consumption to re-use the existing link
9: **else**
10: {We have to open new physical link between i1, j1}
11: Find $cost(i1, j1)$, the marginal power consumption for opening and using the link. Evaluate whether switch frequency constraints are satisfied.
12: **end if**
13: **end for**
14: Assign $cost(i1, j1)$ to the edge $W(i1, j1)$ in SCG
15: Find the least cost path between the partitions in which source (s_k) and destination (d_k) of the flow are present in the SCG. Choose only those paths that have turns not prohibited by PTS
16: Update PHY, Bw_avail, $switch\_size\_in$, $switch\_size\_out$, $switching\_activity$ for chosen path
17: **end for**
18: Return the chosen paths, switch sizes, connectivity

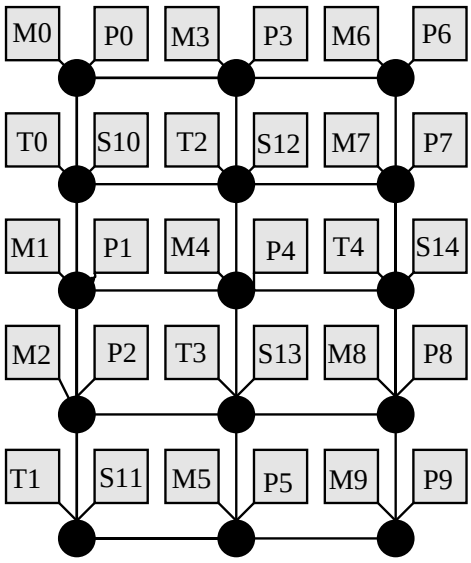
An example illustrating the working of the *PATH_ COMPUTE* procedure is presented in Example 3.4.2. In the procedure, the flows are ordered in decreasing rate requirements, so that bigger flows are assigned first. The heuristic of assigning bigger flows first has been shown to provide better results (such as lower power consumption and more easily satisfying bandwidth constraints) in several earlier works [17], [18]. For each flow in order, the amount of power that will be dissipated across each of the switches, if the traffic for the flow used that switch, is evaluated. This power dissipation value on each switch depends on the size of the switch, the amount of traffic already routed on the switch and the architectural parameter point (θ) used. It also depends on how the switch is reached (from what other switch)

and whether an already existing physical channel will be used to reach the switch or a new physical channel will have to be opened. This information is needed, because opening a new physical channel increases the switch size and hence the power consumption of this flow and of the others that are routed through the switch. These marginal power consumption values are assigned as weights on each of the edges reaching the vertex representing that switch in the SCG. This is performed in steps 8 and 11 of the procedure. When opening a new physical link, whether the switch size is small enough to satisfy the particular frequency of operation is also checked. As the switch size increases, the maximum frequency of operation it can support reduces (as the critical path inside the switch gets longer) [8]. This information is obtained from the placement&route derived models of the switches, taken as an input to the algorithms, that will be discussed more in detail in the following.

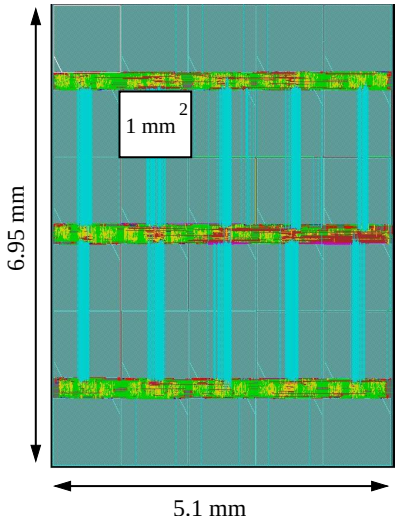
Once the weights are assigned, choosing a path for the traffic flow is equivalent to finding the least cost path in the SCG. This is done by applying Dijkstra's shortest path algorithm [40] in step 15 of the procedure. When choosing the path, only those paths that do not use the turns prohibited by PTS are considered. The size of the switches and the bandwidth values across the links in the chosen path are updated and the process is repeated for other flows.

Example 3.4.2. *For the SCG from Example 3.4.1, let us consider routing the flow of value* 100 *between the vertices* $v1$ *and* $v2$*, across the partitions* $p1$ *and* $p2$*. Initially no physical paths have been established across any of the switches. If we have to route the flow across a link between any two switches, we have to first establish the link. The cost of routing the flow across any pair of switches is obtained from step 11 of the PATH_ COMPUTE procedure. The SCG with the edges annotated with the costs is presented in Figure 3.4(c). The costs on the edges from* $p2$ *are different from the others due to the difference in initial switching activity in* $p2$ *compared to the other switches. This is because the switch* $p2$ *has to support flows between the vertices* $v2$ *and* $v3$ *within the partition. The least cost path for the flow, which is across switches* $p1$ *and* $p2$ *is chosen. Now we have actually established a physical path between these switches and this is considered when routing the other flows. Also, the size and switching activity of these switches have changed, which is noted.*

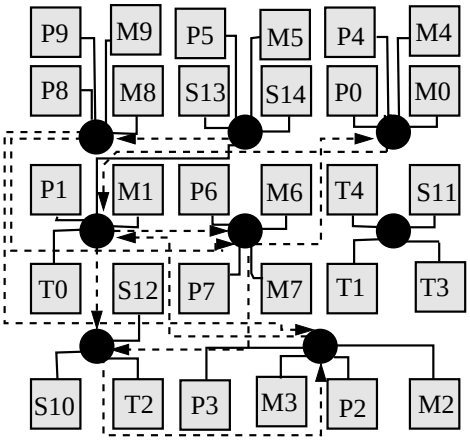
The *PATH_ COMPUTE* procedure returns the sizes of the switches, connectivity between the switches and the paths for the traffic flows. The objective function for establishing the paths is initially set to minimizing power consumption in the switches. Once the paths are established, if hop-count constraints are not satisfied, the algorithm gradually modifies the objective



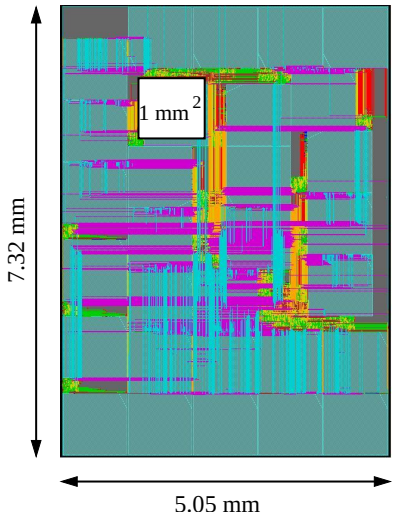
(a) Hand-designed topology



(b) Layout



(c) Automatically synthesized



(d) Layout

Figure 3.5: (a), (b) Hand-designed topology and layout. M: ARM7 processors, T: traffic generators, P, S: private and shared slaves (c), (d) Automatically synthesized topology and layout. In Figure (c), bi-directional links are solid and uni-directional links are dotted.

function to minimize the hop-count as well, using the parameter ρ (in steps 7, 10 and 11 of Algorithm 1). The upper bound for ρ, denoted by ρ_{thresh}, is set to the value of power consumption of the flow with maximum rate, when it crosses the maximum size switch in the SCG. At this value of ρ, for all traffic flows, it is beneficial to take the path with least number of switches, rather than the most power efficient path. The ρ value is varied in several steps until the hop-count constraints are satisfied or until it reaches ρ_{thresh}.

In the next step (step 12, Algorithm 1), the algorithm invokes the floorplanner to compute the design area and wire-lengths. The floorplanner minimizes a dual-objective function of area and wire-length, with equal weights assigned to both. The floorplanner used [35] also supports soft cores, fixed pin/pad locations and aspect ratio constraints for the generated design. From the obtained wire-lengths, the power consumption across the wires is calculated. Also, the length of the wires is evaluated to check any timing violations that may occur at the particular frequency ($freq_\theta$). In the end, the tool chooses the best topology (based on the user's objectives) that satisfies all the design constraints. At the last step, for the synthesized topology, the algorithm automatically generates the information required for the ×pipesCompiler tool for network instantiation and the SoC Encounter tool to perform placement&routing.

The presented NoC synthesis process scales polynomially with the number of cores in the design. The number of topologies evaluated by the methodology also depends linearly on the number of cores. Thus, the algorithms are highly scalable to a large number of cores and communication flows. The synthesis time for several different SoC benchmarks is presented in Section 6 B.

3.5 Experiments and Case Studies

3.5.1 Layout-level Comparisons

To perform a layout level comparison between manually developed and automatically synthesized designs, the NoC design used in previous chapter about layout aware crossbenchmarking of NoCs and bus systems was taken as reference, considering it a suitable example of SoC where to run multi-media benchmarks. As already depicted, the mentioned design consists of 30 cores: 10 ARM7 processors with caches, 10 private memories (a separate memory for each processor), 5 custom traffic generators, 5 shared memories and devices to support inter-processor communication. The hand-designed NoC has 15 switches connected in a 5x3 quasi-mesh network (2 cores connected

to each switch), shown in Figure 3.5(a). The design, as already explained, even if not tailored for a specific application, can be considered quite logically optimized from the point of view of the hardware resources, with the private memories being connected to the processors across a single switch and the shared memories distributed around the switches. A layout of the design (presented in Figure 3.5(b)) was performed using SoC Encounter and the mesh structure was maintained in the layout. Each of the cores has an area of 1 mm^2 [34] in the design. The entire process, from topology specification to layout generation took several weeks. The post-layout NoC could support a maximum frequency of operation of 885 MHz, which is determined by the critical path in the switch pipeline. The power consumption of the topology for a particular functional traffic pattern has been evaluated to be 368 mW. It must be noticed that the different frequency value reported with respect to the previous chapter is due to two main factors. First, being the aim of this layout a comparison between two NoC topologies, the analysis of the clock tree effect on the frequency was avoided for both the compared topologies, thus the reported value of 885 MHz is not affected by the clock skew. Second, the value reported in the previous chapter was kept very conservative for the sake of the fairness of the crossbenchmarking, thus did not include any very stressing optimization of the clock gating insertion, thus showing a higher frequency drop related to this technique. The topology synthesis process was applied with the objective of minimizing power consumption, to automatically synthesize the NoC for this application. The design constraints and the required frequency of operation were set to be the same (885 MHz) as that of the hand-designed topology. The synthesized NoC topology and the layout obtained using SoC Encounter are presented in Figures 3.5(c) and 3.5(d). The synthesized topology has fewer switches (8 switches) than the hand-designed topology. It can support the same maximum frequency of operation (885 MHz), without any timing violations on the wires. As the wire-lengths were considered during the synthesis process to estimate the frequency that could be supported, the most power efficient topology that would still meet the target frequency could be synthesized. To reach such a design point manually would require several iterations of topology design and place&route phases, which is a very time consuming process.

Layout level power consumption calculations on functional traffic show that the synthesized topology has 277 mW power consumption, which is 1.33× lower than the hand-designed topology. Given the fact that the hand-designed topology is reasonably optimized, with much of the communicating traffic (which is between the ARM cores and their private memories) traversing only one switch, these savings are achieved entirely from efficiently spreading the shared memories around the different switches. As mentioned,

the layout of the hand-designed NoC was optimized to a large extent to reduce the area of the design (by moving switches, network interfaces), performing a reasonable number of iterations due to the big effort required by this manual procedure. The layout of the synthesized topology is obtained completely automatically, and still the area of the design is close to that of the manual design (only a marginal 4.3% increase in area).

Cycle-accurate simulations of the hand-designed and the synthesized NoCs for two multimedia benchmarks were performed. The total application time for the benchmarks (including computation time) and the average packet latencies for read transactions for the topologies are presented in Figures 3.6(a) and 3.6(b). The custom topology not only matches the performance of the hand-designed topology, but provides an average of 10% reduction in total execution time and of 11.3% in packet latency.

3.5.2 Experiments on SoC Benchmarks

The topology design procedure was tested over six different SoC benchmarks:

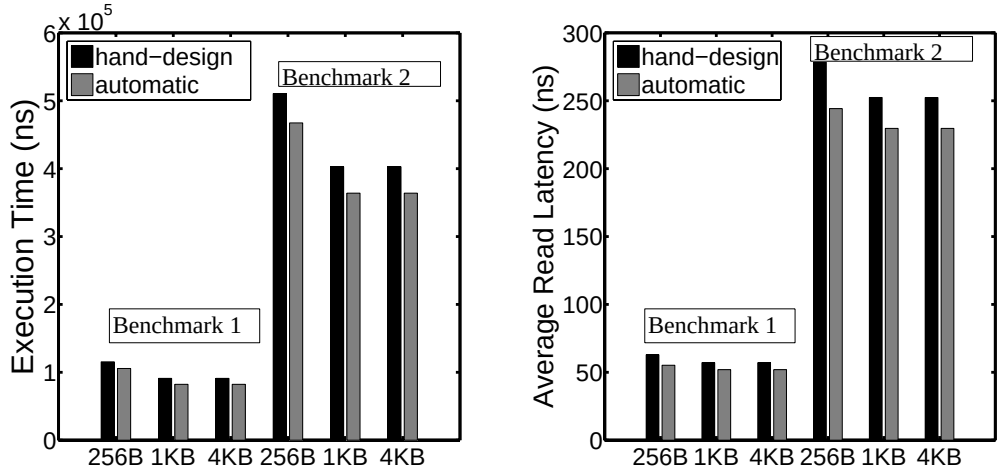
- *video processor (VPROC-42 cores),*
- *MPEG4 decoder (12 cores),*
- *Video Object Plane Decoder (VOPD-12 cores),*
- *Multi-Window Display application (MWD-12 cores),*
- *Picture-in-Picture application (PIP-8 cores)*
- *IMage Processing application (IMP-23 cores).*

Communication characteristics of some of these benchmarks are reported in [7].

For comparison, a mesh topologies for the benchmarks was also generated by modifying the design procedure to synthesize NoCs based on mesh structure. To obtain mesh topologies, we generate a design with each core connected to a single switch and restrict the switch sizes to have 5 input/output ports. A variant of the basic mesh topology was also generated: *optimized mesh (opt-mesh)*, where those ports and links that are unused by the traffic flows are removed. The core graph and the floorplan for the custom topology synthesized by the synthesis tool for one of the benchmarks (VOPD) are shown in Figure 3.7. The network power consumption (power consumption across the switches and links), average hop-count and design area results for the different benchmarks are presented in Table 3.2. Note that the average

hop-count is the same for mesh and opt-mesh, as in the opt-mesh only the unused ports and links of the mesh have been removed and the rest of the connections are maintained. The custom topology results in an average of 2.78× improvement in power consumption and 1.59× improvement in hop-count when compared to the standard mesh topologies.

The area of the designs with the different topologies is similar, thanks to efficient floorplanning of the designs. It can be seen from Figure 3.7 that only very little slack area is left in the floorplan. This is because the presented flow considers the area of the network elements during the floorplanning process, and not after the floorplanning of blocks. The total run time of the topology synthesis and architectural parameter setting process for the different benchmarks is presented in Table 3.2. Given the large problem sizes and very large solution space that is explored (8 different frequency steps, 4 different link-widths, 42 cores for VPROC and several calls to the floorplanner) and the fact that the NoC parameter setting and topology synthesis are important phases, the run-time of the engine is not large. This is mainly due to the use of hierarchical tools for partitioning and floorplanning and to the development of fast heuristics to synthesize the topology.



(a) Execution time

(b) Average read latency

Figure 3.6: Run time and latency for different cache sizes

A comparisons of synthesized topology against several other standard topologies is also presented. For mapping the cores onto the standard topologies, we use the tool from [17]. As the power libraries used for switches, links in the tool are different from the ones used in the synthesis process, the

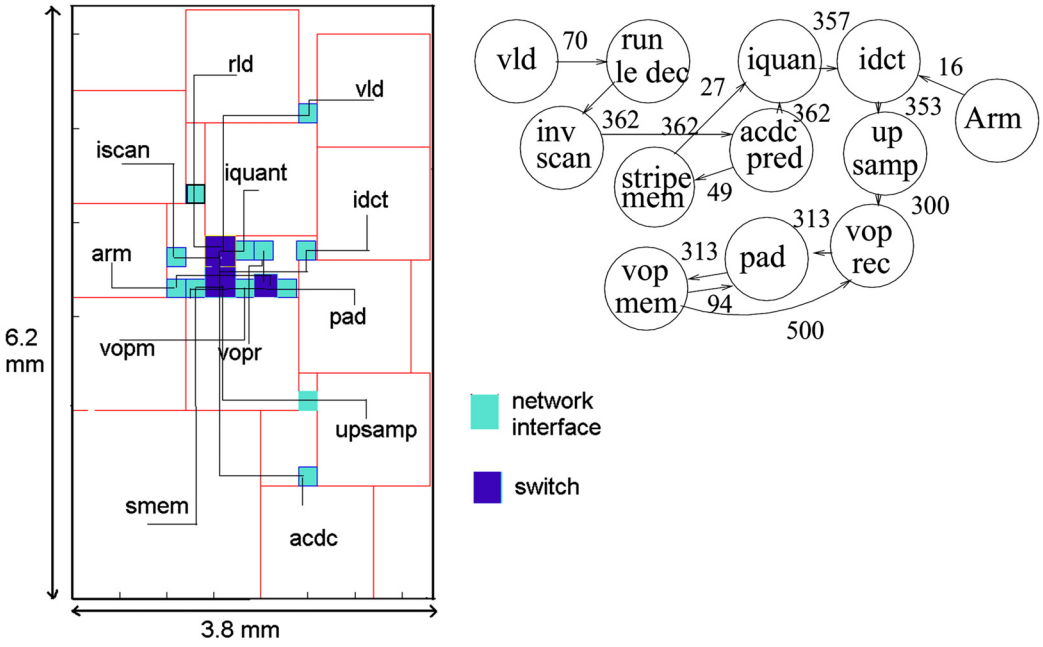


Figure 3.7: VOPD custom topology floorplan and core graph

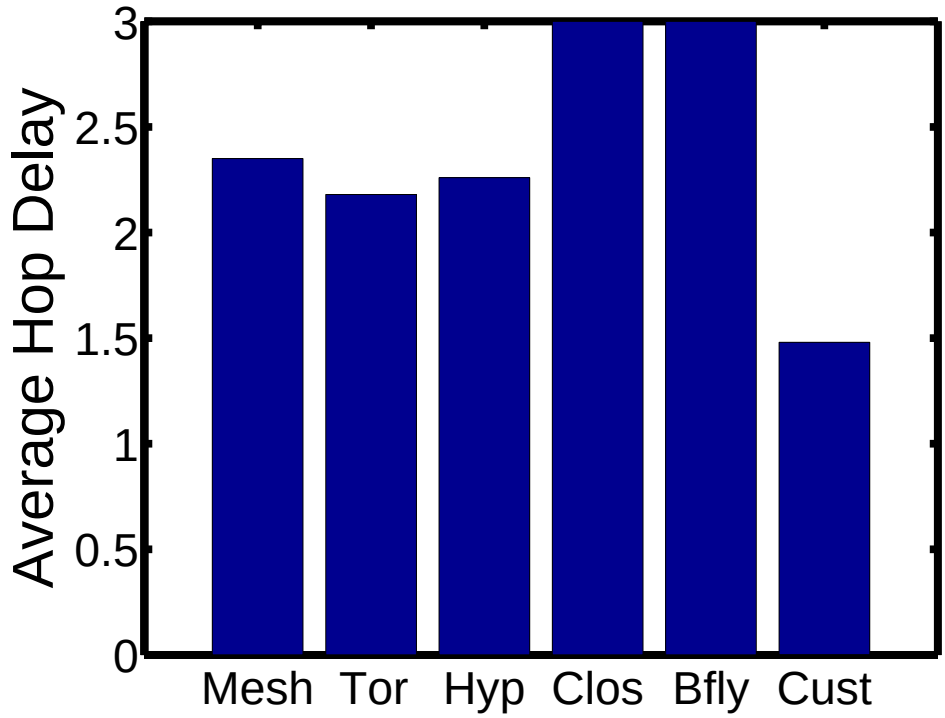


Figure 3.8: Performance comparisons

Table 3.2: Comparisons with standard topologies

Appl	Topol.	Power (mW)	Avg. Hops	Area mm^2	Time (mins)
VPROC	custom	79.64	1.67	47.68	68.45
	mesh	301.8	2.58	51.0	
	opt-mesh	136.1	2.58	50.51	
MPEG4	custom	27.24	1.5	13.49	4.04
	mesh	96.82	2.17	15	
	opt-mesh	60.97	2.17	15.01	
VOPD	custom	30.0	1.33	23.56	4.47
	mesh	95.94	2.0	23.85	
	opt-mesh	46.48	2.0	23.79	
MWD	custom	20.53	1.15	15	3.21
	mesh	90.17	2.0	13.6	
	opt-mesh	38.60	2.0	13.8	
PIP	custom	11.71	1	8.95	2.07
	mesh	59.87	2.0	9.6	
	opt-mesh	24.53	2.0	9.3	
IMP	custom	52.13	1.44	29.66	31.52
	mesh	198.9	2.11	29.4	
	opt-mesh	80.15	2.11	29.4	

topologies were optimized for performance, subject to the design constraints. The comparisons against 5 standard topologies (mesh, torus, hypercube, Clos and butterfly) for an image processing benchmark with 25 cores is presented in Figure 3.8. The custom topology synthesized by the presented synthesis method shows large performance improvements (an average of 1.73×) over the standard topologies.

As an interesting observation, it must be noticed that prohibiting certain turns to avoid deadlocks during routing had a negligible impact on the power and performance results for all of the benchmarks. This was because, even if some turns were avoided, the path computation procedure could easily find other paths with low cost, as several alternative low cost paths exist between each source and destination in the SCG (refer to Section 5).

Bibliography

[1] S. Dutta et al., “Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems”, IEEE D&T, Sep/Oct 2001, pp. 21-31.

[2] http://www.st.com

[3] http://www.ti.com.

[4] L.Benini and G.De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, pp. 70-78, Jan. 2002.

[5] D.Wingard,"MicroNetwork-Based Integration for SoCs", Proc. DAC, pp. 673-677, Jun 2001.

[6] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", DATE 2005.

[7] D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multi-Processor Systems-on-Chip", IEEE TPDS, Feb 2005.

[8] S. Stergiou et al., “×pipesLite: a Synthesis Oriented Design Library for Networks on Chips”, pp. 1188-1193, Proc. DATE 2005.

[9] J. Daveau et al., “Synthesis of system-level communication by an allocation based approach”, Proc. ISSS, pp. 150-155, Sept. 1995.

[10] M. Gasteier, M. Glesner, “Bus-based communication synthesis on system level”, ACM TODAES, vol.4, no.1, pp. 1-11, 1999.

[11] K. Ryu, V. Mooney, “Automated Bus Generation for Multiprocessor SoC Design”, Proc. DATE, pp. 282-287, March 2003.

[12] K.Lahiri et al., “Design Space Exploration for Optimizing On-Chip Communication Architectures”, IEEE TCAD, vol.23, no.6, pp. 952- 961, June 2004.

[13] S. Murali, G. De Micheli, “An Application-Specific Design Methodology for STbus Crossbar Generation”, pp. 1176-1181, Proc. DATE ’05.

[14] S. Pasricha et al., “Floorplan-aware automated synthesis of bus-based communication architectures”, Proc. DAC ’05.

[15] J. Hu et al., “System-Level Point-to-Point Communication Synthesis Using Floorplanning Information”, Proc. ASPDAC ’02.

[16] J. Hu, R. Marculescu, ’Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures’, Proc. DATE, March 2003.

[17] S. Murali, G. De Micheli, “SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs”, Proc. DAC 2004.

[18] A. Hansson et al., “A Unified Approach to Mapping and Routing on a Combined Guaranteed Service and Best-Effort Network-on-Chip Architectures”, Technical Report No: 2005/00340, Philips Research, April 2005.

[19] S. Murali et al., “Mapping and Physical Planning of Networks on Chip Architectures with Quality-of-Service Guarantees”, Proc. ASPDAC 2005.

[20] S. Murali et al., “A Methodology for Mapping Multiple Use-Cases onto Networks on Chips”, pp. 1-6, Proc. DATE, 2006.

[21] R. Ravi et al., “Approximation algorithms for degree-constrained minimum-cost network design problems”, Algorithmica, 31(1): 58-78, 2001.

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

[23] W.H.Ho, T.M.Pinkston, “A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns”, HPCA 2003, pp. 377-388, Feb 2003.

[24] T. Ahonen et al. "Topology Optimization for Application Specific Networks on Chip", Proc. SLIP 04.

[25] K. Srinivasan et al., “An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks”, Proc. ICCAD ’05.

[26] A. Jalabert et al., “×pipesCompiler: A tool for instantiating application specific networks-on-chip”, pp. 884-889, Proc. DATE 2005.

[27] D.Siguenza-Tortosa, J. Nurmi, “Proteo: A New Approach to Network-on-Chip”, in CSN 02, Sep. 2002.

[28] X.Zhu, S.Malik, “A Hierarchical Modeling Framework for On-Chip Communication Architectures”, ICCD 2002, pp. 663-671, Nov 2002.

[29] T. T. Ye et al., “Analysis of powerconsumption on switch fabrics in network routers”, Proc. DAC ’03.

[30] H-S Wang et al., “Orion: A Power-Performance Simulator for Interconnection Network”, Proc. Micro, Nov 2002.

[31] N. Banerjee et al., “A power and performance model for network-on-chip architectures”, Proc. DATE ’04.

[32] G. Palemoro, C. Silvano, “PIRATE: A Framework for Power/Performance Exploration of Network-On-Chip Architectures”, PATMOS 2004

[33] D. Starobinksi et al., “Application of network calculus to general topologies using turn-prohibition”, IEEE/ACM Transactions on Networking, Vol. 11, Issue 3, pp. 411-421, June 2003.

[34] F. Angiolini et al., “Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness”, pp. 124-129, Proc. DATE 2006.

[35] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning : Enabling Hierarchical Design", IEEE Trans. on VLSI Systems, vol 11(6), pp. 1120-1135, Dec 2003. URL: http://vlsicad.eecs.umich.edu/BK/parquet/

[36] B. Hendrickson, R. Leland, “The Chaco User’s Guide: Version 2.0”, Sandia Tech Report SAND94–2692, 1994. URL: //www.cs.sandia.gov/˜bahendr/chaco.html

[37] www.cadence.com

[38] www.synopsys.com

[39] W. J. Dally, B. Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmann , Dec 2003.

[40] T. H. Cormen et al., "Introduction to Algorithms", The MIT Press, June 1990.

Chapter 4

Area and Power Modeling for Networks-on-Chip components

As discussed in the previous chapters, to be fully competitive with traditional interconnects, NoCs need well-established CAD deployment tools to tackle the large amount of available degrees of freedom, starting from the choice of a network topology. In Chapter 3 a complete "Silicon-aware" optimization tools was outlined, able to select a NoC topology taking into account the tradeoff between performance and hardware cost, *i.e.* area and power consumption. A key requirement for the effectiveness of this tool, is the availability of accurate analytical models for power and area. Such models are unfortunately not as available and well understood as those for traditional communication fabrics. Further, simplistic models may turn out to be totally inaccurate when applied to wire-dominated architectures; this observation demands at least for a model validation step against placed and routed devices.

It is mandatory to do not underestimate the complexity of synthesis flows, which involve multiple tools, increasingly complex libraries, a large amount of heuristics and several approximations or models of the behaviour of physical on-chip devices. Experience shows that one wrong assumption may severely impact the properties of a whole design.

The increasing importance of wiring resources is deeply impacting the final performance of circuits, by introducing higher parasitic capacitances and therefore power consumption, or by forcing redesign iterations due to higher transition latencies. This scenario demands for careful assessment of the accuracy of any predictive model at the lowest available level of abstraction; if possible, designers should try to validate their assumptions on placed and routed netlists. This obviously requires a large effort and may be impractical.

In this chapter a method to devise analytical models of area occupation and power consumption of NoC switches for the reference NoC architecture

×pipes was presented, and strategies for coefficient characterization which have different tradeoffs in terms of accuracy and of modeling activity effort are proposed. The models are parameterized on several architectural, synthesis-related and traffic variables, resulting in maximum flexibility. The accuracy of the models is also assessed, checking whether they can also be applied to placed and routed NoC blocks.

It must be noted that, as with any hardware component, the hardware cost of a NoC switch depends on several kinds of parameters, including (i) architectural (*e.g.* amount of buffering), (ii) synthesis tool-related (*e.g.* target operating frequency), (iii) operating (*e.g.* traffic flows).

The proposed NoC modeling methodology which takes advantage of the designer's knowledge of the target architecture and synthesis library. It is of course impossible to devise an accurate yet fully generic model for the hardware cost, in power and area, of any given NoC. The focus is instead on how such a model can be built for a specific NoC instance; the challenges and opportunities involved in this flow are illustrated, in terms of accuracy and characterization time.

The ×pipes NoC switch can be considered an optimal case study due to its parameterizability (Section 4.1). Key properties of the presented approach include accuracy and explicit modeling on several parameters of the design, like switch cardinality, flit width, buffering, traffic and synthesis parameters. These properties make the approach suitable for fast exploration of large parts of the fabric design space, flexible and applicable in real life, for example by accounting for the behaviour of the synthesis tools when the target operating frequency approaches the limits of the design. The characterization is dependent on the target technology library, but can be easily scripted and automated. A major feature of the proposed methods is the accuracy of the modeling style against placed and routed test instances. This is an essential step given the uncertainties intrinsic in today's technology processes. Model coefficients can be made even more accurate by using a placed and routed training set for characterization, albeit at a modeling effort cost. Remaining inaccuracies can mostly be attributed to the intrinsic variations induced by synthesis tools.

The proposed modeling approach starts from an existing RTL description of the NoC components, which are then synthesized and characterized under multiple architectural configurations and traffic conditions. A mathematical formulation of the area and power models is derived from empirical evidence and from the designer's knowledge of the NoC. Eventually, the coefficients of the model are fitted to the experimental results, guaranteeing accurate results for the given architecture. Two different ways of characterizing the coefficients are presented, with varying accuracy/effort tradeoffs, and two

models to account for the dependency of synthesis results on the target synthesis frequency. The synthesis process can optionally include the placement and routing step for maximum thoroughness of the assessment.

Power models and simulators for processors and memories have been proposed in an extremely large body of research [3, 14].

Some models of NoC hardware cost have already been proposed in previous literature. Results in [12] are derived from a mix of results on template circuits and from technology trends, and are specifically aimed at wide applicability. Therefore, even though they have been used for design space exploration [11] and in association with high-level traffic injection models [5], they do not guarantee maximum accuracy within an architecture-specific CAD flow. The main advantage of these techniques is flexibility and fast deployment. They can be seen as complementary to the presented approach, especially for initial exploration when the NoC component library is not available yet.

The approach in [4], on the other hand, attempts to build a cycle-accurate power model of a target router instance. However, several major points differentiate it by the proposed approach. First, the models presented in this chapter are parametric not only on traffic-related events, but also on the architectural knobs of the design. A second difference is related to the introduction of an area model. Third, the model presented in the following can be more readily adopted within a CAD mapping flow due to its definition as a function of architectural parameters, and to its capability of providing a high-level dependence on traffic variables, instead of a cycle-by-cycle one. Fourth, the presented approach aims to be as applicable as possible in real-world conditions, including the hard-to-model peculiarities of the behaviour of synthesis tools when aiming for maximum frequency operation, and placement and routing issues. Fifth, a fast characterization mechanism is proposed, by means of which model coefficients can be quickly derived with a minimal amount of synthesis runs.

In [7], a framework for NoC exploration is presented; the framework includes a power modeling flow. The power model features very limited dependence on architectural parameters and does not seem to account for the configuration knobs of synthesis tools. No area model is provided.

In [13], a bit energy modeling flow is proposed to compare different switch fabrics in IP network routers. The approach is focused on the cost for transmitting bits from input to output ports, and while bit pattern-accurate, it is only focused on comparing router topologies against each other. The authors of [8] propose a model based on transistor count, while in [15], which is focused on FPGAs, switch cardinality is the main parameter. None of these models is meant for simultaneously accurate, parametric and fast representa-

tion of power consumption, *i.e.* suitable for design space exploration within a CAD environment.

4.1 The ×pipes Switch Architecture

As mentioned, the switch architecture taken as reference is the one defined in the ×pipes NoC component library [1, 2], due to its customizability. The ×pipes switch was already depicted in Chapter 1, but it is useful to remind in this section those features that are more closely related to the model understanding. The ×pipes switch is output buffered; FIFOs of configurable depth are instantiated at each output, while inputs feature a single register. The flit width can be arbitrarily set. The number of input and output ports is also a parameter; full connectivity is provided in the central crossbar. An arbiter is attached to each output port to handle contention issues. We test the switch with its default ACK/NACK flow control mechanism, which leverages the output buffer resources. Since ×pipes performs source routing, the switch does not include routing LUTs.

4.2 Proposed Modeling Methodology

The presented modeling activity is composed of five main phases. First, a set of parameters that are relevant to the accuracy of any model which aims at practical applicability is devised. Second, a general model formula for area and for power is defined, relying on the knowledge of the target architecture as explained in Section 4.1. Third, several configurations (*training set*) of the target switch architecture were synthesized in a 0.13 μm technology library with Design Compiler [9], and the corresponding area and power consumption is measured. The configurations are chosen so as to uniformly but sparsely cover the design space of interest, therefore allowing for an accurate yet quick construction of the model. Fourth, the obtained experimental results are used to numerically quantify the coefficients of the model. Two different ways of performing this step are presented. Fifth, the quality of the obtained models is assessed against configurations (*test set*) outside of the training set. The first four steps will be covered in Subsections 4.2.1, 4.2.2, 4.2.3, 4.2.4, while the fifth will be discussed in Section 4.3. An outline of how we handle steps 3-5 is provided in Figure 4.1.

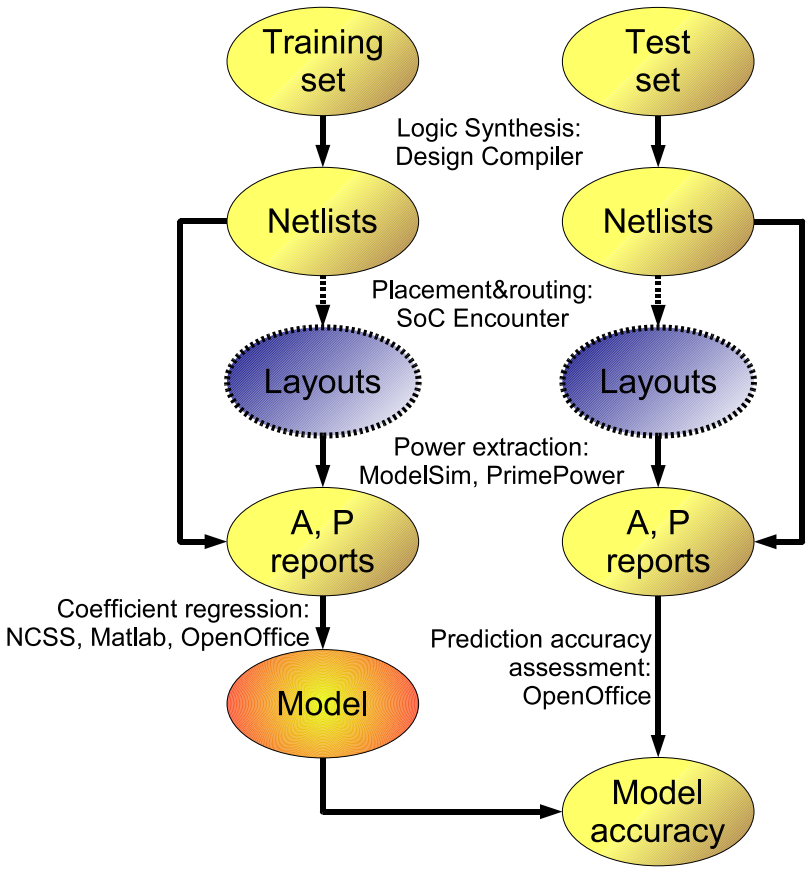


Figure 4.1: Outline of our characterization activity. The placement and routing step is optional both for the training and the test set

4.2.1 Parameters of Interest

A key phase of the approach is devising a model that matches the architecture under consideration and its properties. However, considering the architecture alone does not guarantee that the model will be applicable and accurate enough in practice. For example, synthesis tools play a primary role in defining the area and power efficiency of a component. Therefore, it is useful to summarize the parameters of interest when assembling the model.

Architectural Parameters

The main parameters are:

- Switch cardinality (number of ports). To account for rectangular switches, the amount of input ports (np_i) and output ports (np_o) are separately considered.
- Amount of buffering devoted to flow control handling and performance optimization, also called buffer depth (bd) (expressed in terms of single-flit buffering elements).
- Number of bits of the incoming and outgoing elementary data blocks, also called flit width (fw).

Implementation Flow Parameters

It is possible to tune synthesis tools, among other things, for:

- Target frequency of operation.
- Target area.
- Target power consumption.

Tuning these parameters differently in the synthesis tools yields, as expected, a widely different quality of results. For example, when performance demands are extreme, synthesis tools are forced to create netlists containing large amounts of buffers and fast gates, which are not area- and power-efficient. To mimic a typical industrial flow, where an application performance constraint must be satisfied, for the sake of the modeling activity, a certain target operating frequency (which is a parameter of the model) is imposed as the primary objective, while area and power minimization are given to the tool as secondary optimization objectives. As a result, area

and power requirements, expressed as a function of the target operating frequency, exhibit a characteristic flat behaviour followed by a steeply rising trend after an inflection point. This trend is well known, and can be explained by the fact that, above some target operating frequency which can be achieved with minimal circuitry, synthesis backends are forced to insert extra gates to comply with increasing performance demands.

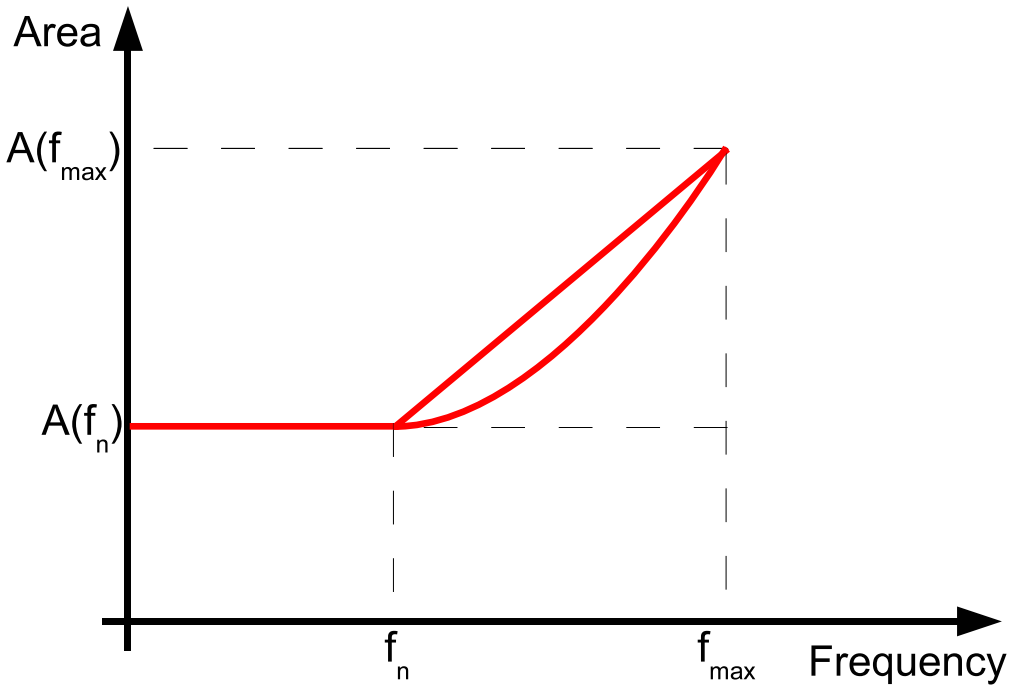


Figure 4.2: Area requirements *vs.* target operating frequency

Figure 4.2 shows a linearized and a parabolic approximation of this trend, and at the same time summarizes the ways this effect is modeled. For each device configuration (*e.g.* 4x4 32-bit switches with 6-deep FIFO buffers), a "native" frequency f_n can be identified. This frequency is that achieved by the synthesizer with relaxed timing constraints. Under this condition, the tool is free to fully pursue its secondary objectives, hence creating minimum area ($A(f_n)$) and power ($P(f_n)$) netlists. Configuring the tools for target frequencies lower than f_n does not result in further decreases of area or power dissipation. For each switch instance, it is also possible to find a frequency f_{max}, that corresponds to the fastest achievable synthesis result. Under this timing constraint, the module has $A(f_{max})$ area and $P(f_{max})$ power consumption. The dependency of area and power overheads is approzimated as linear or parabolic in the range $(f_n; f_{max})$. This assumption allows to character-

ize devices only twice, at f_n and f_{max} (under various combinations of the other architectural parameters), while being able to estimate results over the whole range of frequencies achievable by the module. Since this analysis is not correlated to other model parameters, in the following, for simplicity of notation, the dependency of coefficients on the synthesis target frequency will not be mentioned; the characterization of this parameter will be implicitly assumed.

The linearized or parabolic approximation is a way of abstracting away from low-level details of the logic synthesis process, which are impossible to capture in a high-level model. The experimental results that will be shown in Section 4.3 will be based on a test set which is also spread in terms of target operating frequency, therefore providing a metric of the accuracy of such a model. Subsection 4.3.5 will compare the accuracy of the linear *vs.* the parabolic models.

It must be noticed that developing area and power models which are a function of the target frequency of operation up to f_{max} also implies making available a model of the timing properties of the switches.

Traffic Condition Parameters

These parameters are only relevant to power models, since area models are clearly static. They include downstream congestion and internal congestion (*i.e.* arbitration conflicts). They will be explained in more detail in 4.2.2.

4.2.2 Area and Power Models

Area Model

In general, the area equation must be of the form of Equation 4.1:

$$A = f(bd, fw, np_o, np_i) \tag{4.1}$$

The area model expressed in Equation 4.2 was identified as suitable:

$$\begin{aligned} A(fw, bd, np) = A_1 \cdot np_o \cdot fw \cdot bd + \\ + A_2 \cdot np_i \cdot fw + A_3 \cdot np_o \cdot np_i + A_4 \cdot fw \cdot np_o \cdot np_i \end{aligned} \tag{4.2}$$

The rationale of this formula is that the area of the target switch can be rendered as the sum of four contributions (Section 4.1): (i) output buffers, (ii) input buffers, (iii) arbitration and flow control logic, (iv) crossbar. Each contribution strongly depends on a known combination of architectural parameters:

- Output buffers, which are dominated by flip-flop area, can be supposed to depend linearly on flit width fw and buffer depth bd ($\times$pipes switches are output-buffered), which respectively represent the width and depth of the buffer (Figure 4.3). There are np_o such buffers.

- Input buffers are similar to the case above, but since they have a constant depth, they do not depend on bd. Obviously np_i is used in place of np_o.

- Since a distributed arbitration technique is used in the target switch, one arbiter is instantiated at each output port. Each arbiter has a complexity proportional to the number of candidate input ports np_i, therefore the overall contribution is the product of the input and output cardinalities. The arbiter logic is clearly independent of datapath parameters such as flit width and buffer depth.

- The area overhead due to the crossbar must have a linear dependency on flit width, must be independent of the buffering resources and must have a linear dependency on the product of input and output cardinalities.

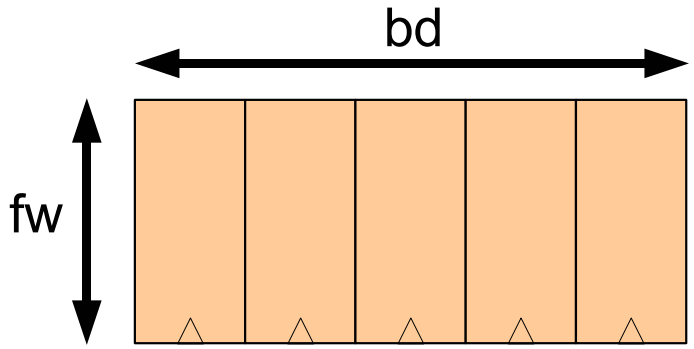


Figure 4.3: Dependency of the output buffer area on fw, bd

Power Model

The power consumption of a module depends on the switching activity of the cells, so, to express the power consumption of a NoC switch, a term that accounts for traffic conditions must be present. The most general way to model the power consumption thus becomes:

$$P = f(bd, fw, np_o, np_i, T) \tag{4.3}$$

with T being a generic variable that summarizes the traffic conditions. Since sequential components exhibit a power consumption even if they are not performing computation, due to the clock switching, a static (traffic-independent) term must appear. After analyzing the possible traffic flows in the ×pipes router, the Equation 4.4 can be used as a general power model:

$$\begin{array}{c} P(bd, fw, np_o, np_i, T) = P_A(...)+ \\ +\sum_{j=1}^{np_o}[P_B(...) \cdot T_{Oj}]+ \\ +\sum_{j=1}^{np_o}[P_C(...) \cdot T_{OCj}]+ \\ +\sum_{j=1}^{np_i}[P_D(...) \cdot T_{ICj}] \end{array} \tag{4.4}$$

where the dots express dependencies on bd, fw, np_o, np_i which will be analyzed in more depth in the following. The first term models the power dissipated by inactive, but still clocked, registers. The remaining terms depend on traffic conditions. An accurate representation of the traffic conditions requires a separate analysis of the state of each input and output port. Therefore, np_o traffic variables T_{Oj} and T_{OCj} are defined, to model the lack or presence of external congestion, and np_i traffic variables T_{ICj}, to model internal contention for resources. More specifically:

- T_{Oj}: Percentage of time during which the output port j is successfully transmitting flits. This coefficient models traffic in absence of congestion.
- T_{OCj}: Percentage of time during which the output port j is trying to transmit, but flits are rejected. This coefficient models external congestion due to traffic spikes.
- T_{ICj}: Percentage of time during which the input port j of the switch is trying to transmit flits through one of the output ports, but arbitration is denied by the switch logic. This coefficient models the contention for the same output port inside of the switch.

This set of traffic percentages is linearly independent, since the complex arbitration and flow control patterns within a NoC switch make it very easy for some of these time windows to overlap. Please consider the following:

Example 4.2.1. A 4x2 switch (see Figure 4.4) may feature one established input-to-output connection where traffic is freely flowing (which is expressed

by the condition T_{O1}), another established input-to-output connection which is stuck due to congestion in the downstream switch (modeled within T_{OC2}), while the third input port is unsuccessfully trying to transmit to one of the two output ports, which in this example are already busy (T_{IC3}), and the fourth is simply idle (this contribution is therefore included in the coefficient P_A).

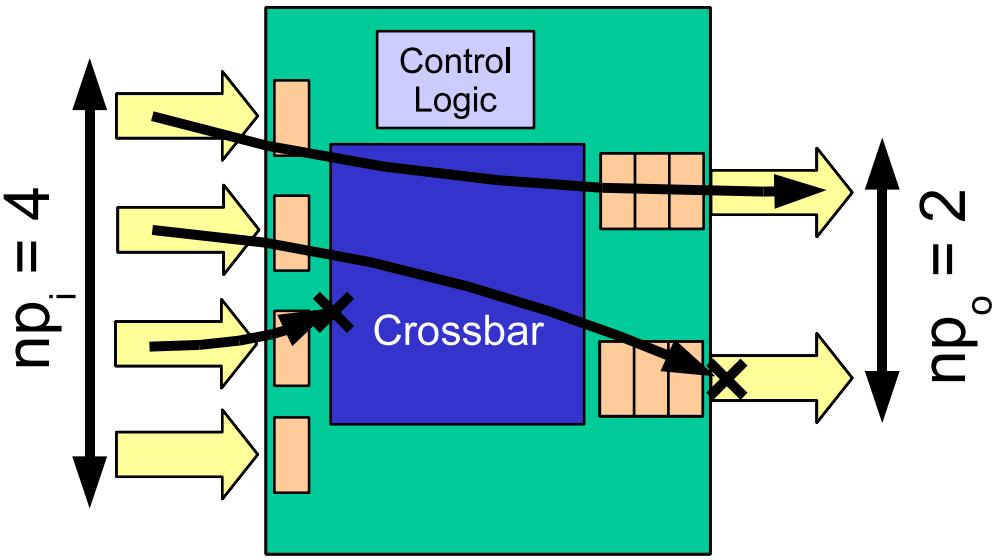


Figure 4.4: Example traffic in a 4x2 switch

The coefficients P_A, P_B, P_C, P_D depend on architectural parameters, as for the area model. They account for the power consumption in the traffic states described above, as follows:

- P_A accounts for the static power dissipated by the switch and it is due to the non-combinational logic in the design. Therefore, it simply depends linearly on the number of flip-flops in the design, which are:
  - input buffers
  - output buffers
  - state registers in the control logic

  whose dependencies on architectural parameters are summarized in Table 4.1.

- P_B accounts for the dynamic power dissipated by flowing packet streams, due to the enabled registers and to the switching activity of combinational logic. We identify four contributions to the power dissipation:

Contribution	*Dep. on* fw	*Dep. on* bd	*Dep. on* np_i	*Dep. on* np_o
output buffering	linear	linear	none	linear
input buffering	linear	none	linear	none
spare registers	none	none	linear	linear

Table 4.1: Dependency on architectural parameters of the static power coefficient P_A

- output buffers. In these buffers, during every cycle, one of the flit registers (fw bits wide) samples a new piece of data; a $bd \times 1$ multiplexer then brings a flit to the output port. Therefore, this contribution is itself the sum of two terms.
- input buffers
- control logic
- selected crossbar branch

The dependencies of these contributions on the architectural parameters are summarized in Table 4.2.

Contribution	*Dep. on* fw	*Dep. on* bd	*Dep. on* np_i	*Dep. on* np_o
output buffer (register)	linear	none	none	none
output buffer (mux)	linear	linear	none	none
input buffer	linear	none	none	none
control logic	none	none	linear	none
crossbar branch	linear	none	linear	none

Table 4.2: Dependency on architectural parameters of the dynamic power coefficients P_B, P_C

- P_C accounts for the dynamic power dissipated by the switch under a scenario where downstream congestion is preventing a free flow of packets. Although numerically different, the P_C coefficient is similar to P_B, in that it still involves an established input-to-output channel, and therefore its dependency on architectural parameters is the same (see Table 4.2).
- P_D accounts for the power dissipated by the switch when an incoming stream requires the access to an output port, but the arbitration is

denied. The contributions to this portion of the power consumption are related to the following logic blocks:

- input buffers
- control logic

The dependencies on this contributions can be summarized as shown in Table 4.3.

Contribution	*Dep. on* fw	*Dep. on* bd	*Dep. on* np_i	*Dep. on* np_o
input buffer	linear	none	none	none
control logic	none	none	linear	linear

Table 4.3: Dependency on architectural parameters of the dynamic power coefficient P_D

The dependencies of the power coefficients are thus summarized in Table 4.4.

Model Coefficient	*Dep. on* fw	*Dep. on* bd	*Dep. on* np_i	*Dep. on* np_o
P_A	linear	linear	linear	linear
P_B	linear	linear	linear	none
P_C	linear	linear	linear	none
P_D	linear	none	linear	linear

Table 4.4: Dependency of power coefficients on architectural parameters

It must be noticed that some coefficients, which could be intuitively expected to quadratically depend on parameters, are instead linearly dependent, because they characterize a single input or output port. The quadratic behaviour is indirectly restored by the summation symbols in Equation 4.4.

4.2.3 Choice of a Relevant Training Set

To characterize the coefficients of our area and power models, a training set is defined, composed of switch configurations chosen in such a way as to uniformly cover the relevant design space for the particular NoC under study. In the case of the ×pipes NoC, which is focused on the highest customizability of topologies, design space spanning over a large variety of cardinalities (np_i

and np_o of 4, 10, 16 and 20) was studied. Since ×pipes is also focused on the best performance/overhead tradeoff point [2], and therefore on low hardware cost, we consider moderate buffer depths bd of 5 and 7 FIFO locations and flit widths fw of 21, 28 and 38 bits.

In the modeling approach called Full Factorial Design, all the possible permutations of the values of the independent design parameters should be studied to create the training set. This is often impractical due to the quick rise in the number of instances as soon as new design knobs are added, leading to approaches to select only a subspace of the characterization set (Fractional Factorial Design). In the presented case, based on the knowledge of the target architecture, a very simple way of pruning the training set was chosen. The rationale is based on the observation that rectangular switches add a smaller amount of information to the training set; for example, when studying the power consumption, a rectangular switch is by design unable to simultaneously feature traffic flows on all of its input and output ports (see Figure 4.4), and is therefore behaving similarly to a square switch of smaller cardinality. Preliminary internal testing confirms this property, at least for the ×pipes NoC. Therefore, the np_i and np_o axes are coalesced for the generation of the training set, and only include 4x4, 10x10, 16x16 and 20x20 instances.

Finally, all the possible parameter values were permutated, resulting in 24 (4 cardinalities times 2 buffer depths times 3 flit widths) configurations being synthesized.

Yes, we applied a load on the output ports which, in our technology library, is roughly equivalent to that of 2 mm of wiring. This load is also approximately equivalent to that featured by the links in the mesh topology.

4.2.4 Fitting Model Coefficients

Fitting Area Model Coefficients

To estimate A_1, A_2, A_3, A_4, two different methods are proposed:

- Methodology 1: Coefficients can be derived directly from synthesis reports, which hierarchically list every switch sub-block. For example, once the area cost of an output buffer which is bd_0 flits deep and fw_0 bits wide is gathered from one report, it can be called $A_{obuf}|_{bd_0,fw_0}$. Since A_1 is expected to increase linearly with both bd and fw, it can be approximately derived as in Equation 4.5:

$$A_1 = \frac{A_{obuf}|_{bd_0,fw_0}}{bd_0 \cdot fw_0} \tag{4.5}$$

Other coefficients can be similarly computed.
Advantages: With this methodology, each contribution in the formula keeps a strict physical meaning. Only one synthesis run is needed to extrapolate coefficients for any switch instance; for the sake of the results presented in this chapter, a 10x10, 28-bit switch was arbitrarily chosen as a reference. This instance is close to the center of the design space of interest (see the previous Subsection); its choice will be further discussed in Section 4.3.
Disadvantages: This simplified approximation discards any constant offset that may be present in the coefficients. Further, the nature of synthesis tools introduces unpredictable fluctuations in the netlist area and power trends under different architectural configurations. This noise does not have any easily characterizable property. Thus, the model incurs a non-negligible error when compared against actual switch instances. Moreover, the choice of the specific switch instance for characterization might skew the computed coefficient values.

- Methodology 2: Coefficients can be derived by leveraging the multivariate non-linear regression algorithms natively provided by several mathematical and statistical packages. In this case, the input is a set of characterization syntheses (the training set described in the previous Subsection). The target polynomial for the regression is chosen based on insight of the dependency of area on the architectural design parameters (see Equation 4.2).
  Advantages: The model fits better to actual synthesis results.
  Disadvantages: Longer characterization time; with a thorough characterization set like that chosen in Subsection 4.2.3, experiments must be performed in 24 device instances, against just one. The actual improvement in accuracy depends on the smoothness of the native behaviour of the synthesis tools. Some coefficients may lose their physical meaning (*e.g.*, they may become negative).

Both methodologies can be readily adapted to any parameterizable NoC architecture.

Fitting Power Model Coefficients

To characterize the P_A, P_B, P_C, P_D coefficients, for first, traffic was injected into the switch netlists under test, one at a time. This is achieved by ModelSim [6] simulation of the Verilog netlists (please refer to Figure 4.1), to which traffic generators are attached. The traffic generators are configured to inject into the switch one of the four patterns described above (idle, free

flow, downstream congestion, internal contention) at a time. The switching activity is logged and fed as an input to Synopsys PrimePower [10], which provides a hierarchical report of the power consumption of the switch sub-blocks. For each netlist of the training set, four hierarchical reports are therefore generated.

At this point, the power model coefficients are determined by using either of the techniques just outlined for the area models. The P_A, P_B, P_C, P_D scenarios are separately accounted for; the fitting polynomials are directly derived from Table 4.4. For each of them, the coefficients modeling the dependency on architectural parameters were extract, either by direct derivation or by non-linear regression,.

4.3 Experimental Results

To evaluate the accuracy of the proposed techniques, a test set of 70 switch configurations spread across the design space of interest (both in terms of architectural parameters and target synthesis frequencies)was randomly chosen, taking care of do not overlap with the training set previously used for characterization. Each switch is synthesized with Design Compiler to extract its area requirements, then stimulated with traffic streams within ModelSim and studied in PrimePower to evaluate its power consumption (Figure 4.1). A reference set of experimental results is therefore collected. The area and power consumption of the same set of switches is then estimated according to the proposed methodology, and the statistical distribution of the resulting error is plotted to study the behaviour of both coefficient fitting strategies.

The implementation flow went through several steps, among which the two major ones are logic synthesis (*i.e.* mapping functionality onto elementary cells from a technology library), which results in a netlist, and placement and routing (*i.e.* placing and interconnecting the netlist within a target floorplan), generating a layout as the outcome. Netlists can be generated in a relatively short time, but they do not include any information about the placement of the cells, and thus do not give any information about the length of the wires needed for the interconnections. This is a key missing piece of information, especially as designs become wire-dominated. Therefore, logic synthesis tools try to model the effect of wires by means of predictive models provided by the vendor of the technology library. These models are necessarily simplistic, and therefore may impact the accuracy of any area and power evaluation at the netlist level. On the other hand, creating the layout of a complex circuit provides more accurate estimations of its area and power cost, but this extra step is at least as time-consuming as the initial logic

synthesis. Therefore, designers would clearly like to avoid performing this extra phase repeatedly during a modeling activity, if at all possible.

To assess the usefulness of the models, their inference and their application to both netlists and layouts were investigated. This can be seen in Figure 4.1, where the placement and routing step is optional.

4.3.1 Experiments with Netlist-Based Models and a Netlist-Level Test Set

In this Subsection, models were generated starting from synthesized (but not placed and routed) switch instances, and their accuracy was checked against a test set which is also at the netlist level. The results are depicted in Figure 4.5, where the vertical axis reports the number of occurrences of inaccuracies comprised in the ranges listed on the horizontal axis. As can be seen, in around 80% of the cases, the models result in an error margin smaller then 10% of the actual value. Sporadically, relatively high error rates of up to 20% are detected; however, as can be seen for example in Figure 4.6, the distribution of the errors is quite randomly spread over the design space, and comprises both under- and overestimations. The figure reports modeling inaccuracy for a subspace having as axes the flit width and the switch cardinality; these numbers are thus only a subset of the whole test set. Similar plots can be derived for varying buffer depths and target synthesis frequencies. Therefore, we can attribute inaccuracies to the unpredictability which is intrinsic in the behaviour of synthesis tools, and not to a problem of the modeling approach.

Comparing the results of the two techniques for coefficient fitting presented in Subsection 4.2.4, it is possible to see that the tails of the inaccuracy distributions drop more sharply for Methodology 2, indicating a lower chance of large modeling errors. However, Methodology 1 exhibits just marginally worse average inaccuracy rates: 6.26% against 5.30% for power models and 5.97% against 5.45% for area models. In terms of characterization effort, experience says that a designer can roughly assume that one hour may be needed in average for the analysis of an instance of the training set; therefore, Methodology 1 requires one hour of runtime, while Methodology 2 needs 24 hours to provide numerical values of coefficients (the actual time depends on how thoroughly the design space is covered). Due to the drastically lower effort, Methodology 1 becomes a natural candidate for fast yet accurate modeling. However, this approach leverages upon a single switch instance to characterize all the coefficients. The choice of the reference switch configuration is therefore key, and may impact the robustness of the flow. Internal testing

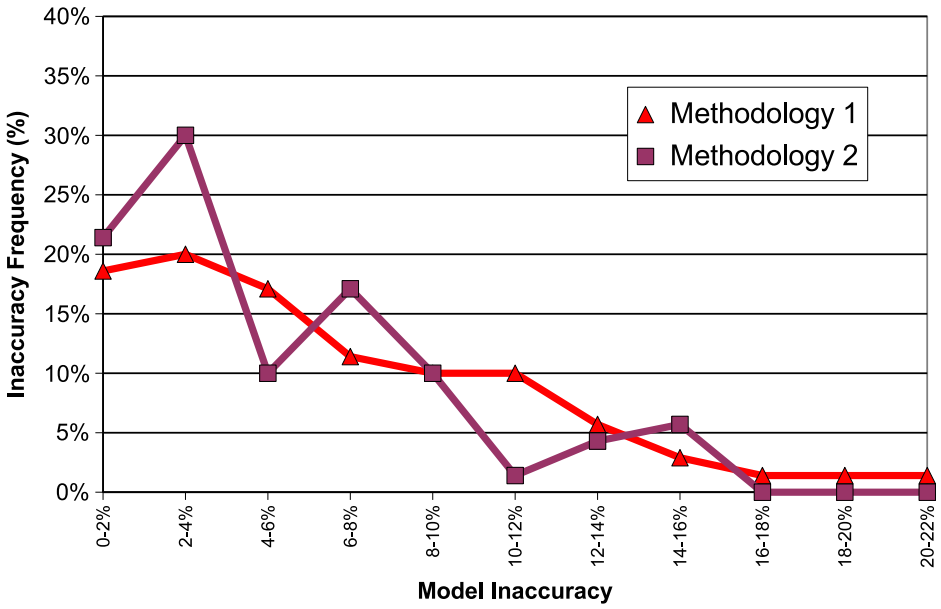
shows that coefficients are quite accurately rendered under a wide range of possible choices of the reference switch. However, when manually picking an "outlier" instance as the reference, errors over the whole design space turn out to be larger. As a possible workaround, Methodology 1 could be applied to multiple switch instances to minimize the chance of choosing bad references; outliers could be effectively discarded. This hybrid approach provides better reliability, but requires a modeling effort which is progressively closer to that of Methodology 2 as its robustness is increased. Methodology 2 remains the most accurate and reliable, and its characterization time can still be assumed to be fully acceptable for both academic and industrial environments.

4.3.2 Test Case: a Complete NoC Topology

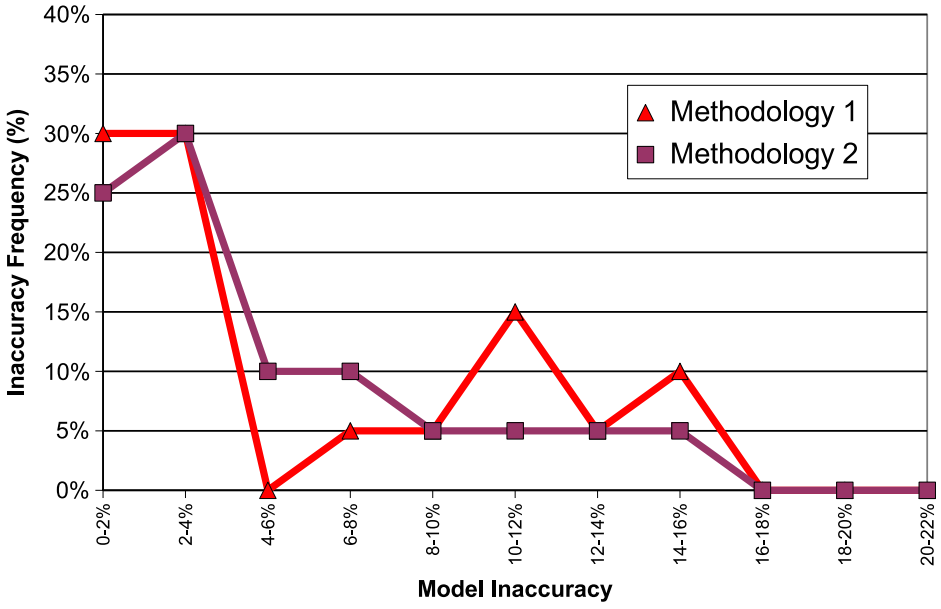
To further validate the most complex part of our methodology, *i.e.* the power modeling, a whole NoC topology, such as a 5x3 mesh, was studied. The mesh includes switches with three different cardinalities of 4x4, 5x5 and 6x6. Functional traffic was injected, namely that required to drive a multimedia application from the MPARM suite, in the topology, and the resulting power consumption was compared against that predicted by the model (characterized with Methodology 2). Traffic patterns in the mesh are irregular, due to application needs, causing the switches to spend variable amounts of time in each possible state. The results are plotted in Figure 4.7. The average inaccuracy is 5%, with only two switches out of fifteen (about 13%) exhibiting inaccuracies greater than 10%. Since the power consumption of some switches is overestimated while that of others is underestimated, the margin of error on the consumption of the whole mesh is as low as 1.3%. This result confirms the usefulness of our modeling strategy for integration within a CAD mapping and design space exploration flow.

4.3.3 Experiments with Netlist-Based Models and a Layout-Level Test Set

Previously mentioned models, which are based on netlist-level analyses, were applied to a layout-level test set, by placing and routing the test set described above. This activity generates a very realistic test set, and is a demanding metric for the accuracy of the models, since extra unpredictable noise is added. The results are presented in Figure 4.8, which should be compared to Figure 4.5(b). The two plots exhibit a comparable trend and errors of roughly the same magnitude, even though the average modeling error for the layout-level test set is about 3% higher. This means that models developed by only taking netlists into account still show good accuracy even



(a)



(b)

Figure 4.5: Area and power coefficient modeling inaccuracy under different characterization policies: (a) area coefficients, (b) power coefficients. Models and test set are at the netlist level

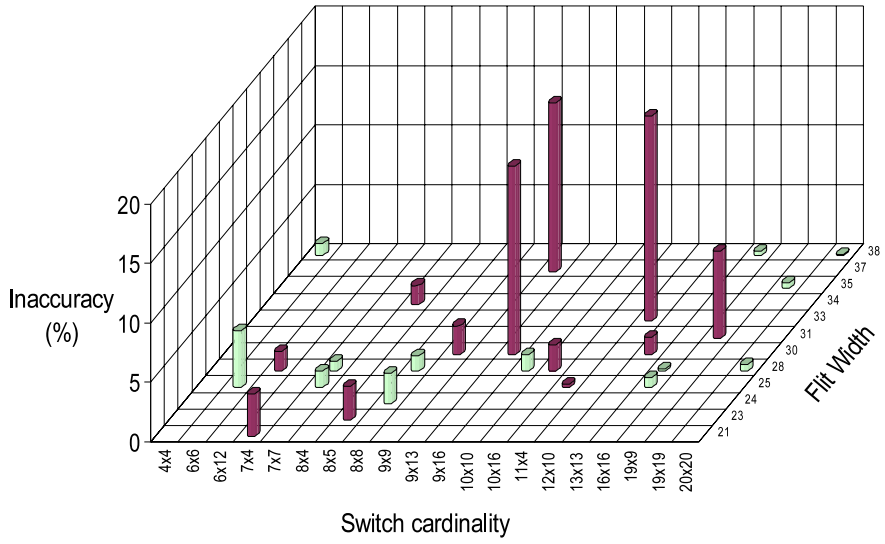


Figure 4.6: Distribution of the area modeling inaccuracy over a subset of the design space for Methodology 2. Dark colour: underestimations; light colour: overestimations

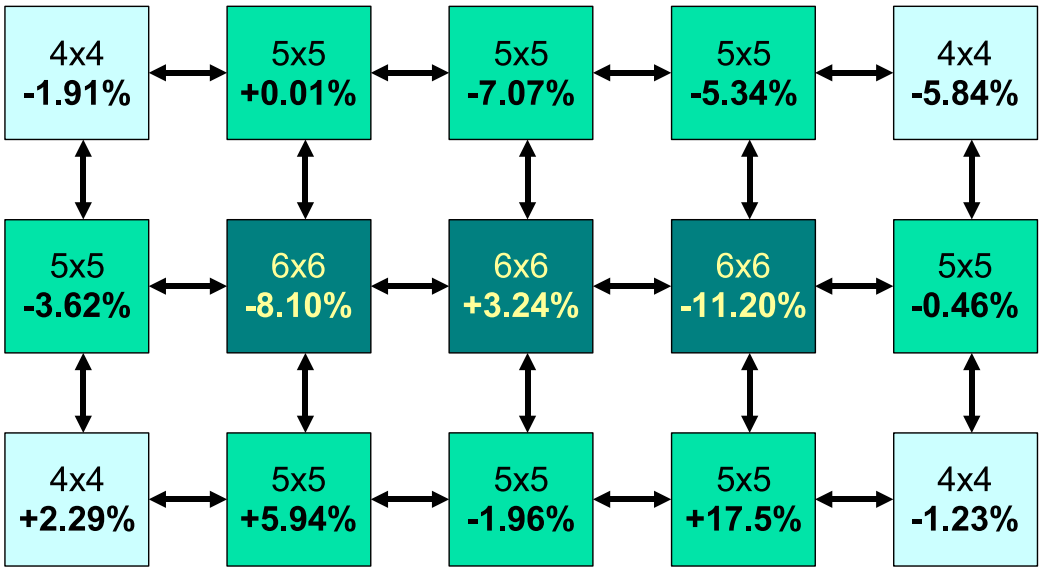


Figure 4.7: Distribution of the power modeling inaccuracy for the switches of a 5x3 NoC mesh

with respect to layout-level power evaluation. The added noise also blurs the accuracy difference between Methodology 1 and Methodology 2, both in maximum error (26% *vs.* 23%) and average error (8.8% *vs.* 8.35%). While Methodology 2 remains marginally more accurate, these results seem to suggest that the unpredictability introduced by the logic synthesis process is somewhat unrelated to that introduced by the placement and routing phase. In other words, even though Methodology 2, thanks to its interpolation of results, can compensate for some of the non-idealities of the logic synthesis process better than Methodology 1, this compensation is less effective when trying to predict the power consumption after placement and routing.

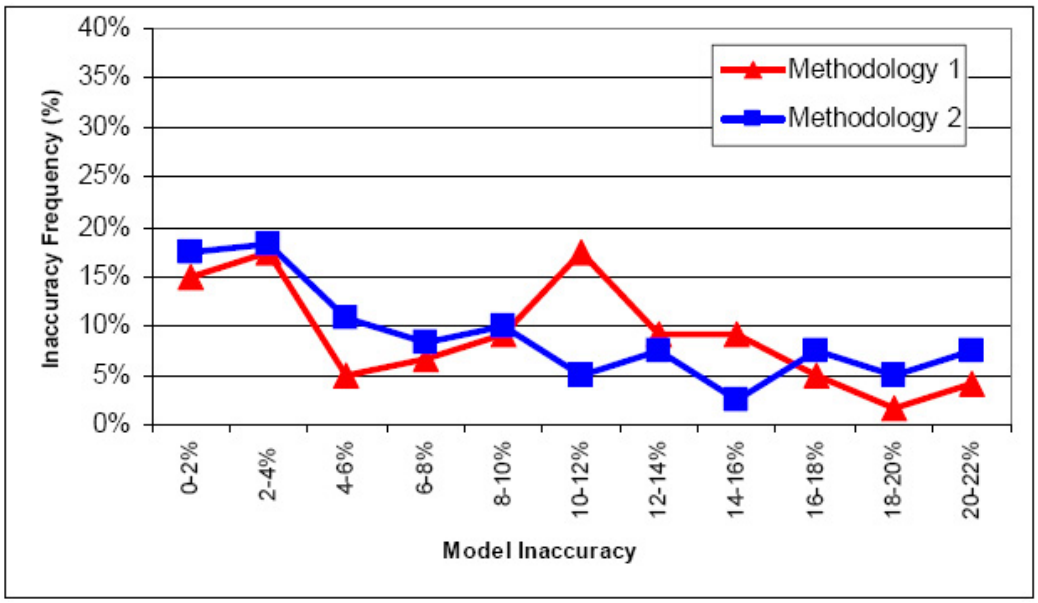


Figure 4.8: Power coefficient modeling inaccuracy under different characterization policies. Models are at the netlist level, test set is at the layout level

4.3.4 Experiments with Layout-Based Models and a Layout-Level Test Set

In an attempt to check whether more accurate models can be built, the numerical coefficients were recomputed starting from a layout-level version of the training set and applying Methodology 2. This model is very close to an ideal reference point, since it is derived from a regression on experimental results which already encompass most of the unpredictable elements of the synthesis flow. However, the time required to build the model coefficients

is noticeably longer. Both logic synthesis and placement steps require a computation time which is not easy to predict, as it largely depends on many factors, such as the switch cardinality and the target operating frequency. However, as a rule of thumb, the two steps are about equally time consuming; therefore, the modeling time is approximately doubled.

The error distribution resulting from the usage of the layout-level test set when validating the model coefficients achieved from a layout-level training set is shown in Figure 4.9.

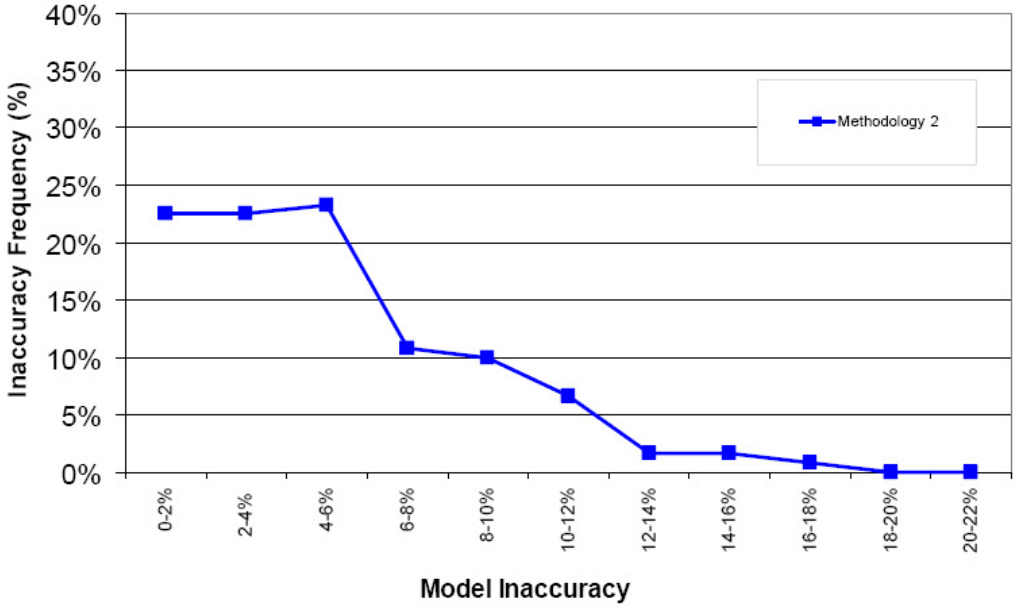


Figure 4.9: Power coefficient modeling inaccuracy for Methodology 2. Models and test set are at the layout level

As can be noticed, and as expected, the average error and the maximum error values both noticeably decrease when compared to Figure 4.8. However, the decrease is not huge. The remaining inaccuracies in Figure 4.9 can be attributed to the intrinsic unpredictability of the synthesis tools. Even after taking into account all the systematic behaviours in the synthesis flow, the trend is the result of residual instance-to-instance variations due to heuristics in the CAD tools and to degrees of freedom which can only vary in a discrete fashion.

The accuracy improvement guaranteed by a layout-level characterization is associated to a doubling of the runtime overhead, and still does not completely eliminate the presence of some "outlier" instances. The designer may certainly choose to adopt this methodology to characterize devices at the

layout level for maximum accuracy. However, a result that can be derived from the present experiments is that, at least at the 0.13 μm technology node, it is still feasible to use accurate netlist-based models in order to save characterization time.

4.3.5 Experiments with a Parabolic Model for the Dependency on the Target Synthesis Frequency

It is useful to find out whether a linear model is accurate enough to characterize the dependency of synthesis results on the target synthesis frequency (see Figure 4.2). A parabolic model was leveraged as a potentially more accurate approximation of the actual dependency of model coefficients on the target frequency, then the model accuracy was re-checked on the test set. The results are reported in Table 4.5.

Experiment		*Linear approx.*	*Parabolic approx.*
Netlist training set, Netlist test set	Average error	5.19%	6.32%
	Maximum error	14.61%	15.27%
Netlist training set Layout test set	Average error	8.23%	6.57%
	Maximum error	22.88%	19.94%
Layout training set, Layout test set	Average error	5.04%	9.23%
	Maximum error	16.10%	22.23%

Table 4.5: Accuracy of the linear *vs.* parabolic models for the dependency of synthesis results on the target synthesis frequency. Coefficients derived with Methodology 2

These results do not seem to indicate a strong bias towards any of the alternatives. The linear approximation seems to cope much better with a netlist-level or layout-level test set when the model is derived from experiments on a training set at the same level, but the parabolic model is quite a bit better at predicting layout-level results starting from netlist-level models. This behaviour can be attributed to the impact of noise. In other words, although synthesis results do clearly change depending on the target frequency, the choice of a linear or parabolic model to describe this trend does not matter much, since the non-idealities introduced by the synthesis flow induce enough noise to blur the distinction. Overall, the usage of the linear model, which is simpler, seems to be justified.

Bibliography

[1] Federico Angiolini, Paolo Meloni, Davide Bertozzi, Luca Benini, Salvatore Carta, and Luigi Raffo. Networks on chips: A synthesis perspective. In *Proceedings of the 2005 ParCo Conference*, 2005.

[2] Federico Angiolini, Paolo Meloni, Salvatore Carta, Luca Benini, and Luigi Raffo. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In *Proceedings of the Design, Automation and Test in Europe (DATE) Conference and Exhibition*, pages 124–129, 2006.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[4] J. Chan and S. Parameswaran. NoCEE: Energy macro-model extraction methodology for network on chip routers. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 254–259, 2005.

[5] Noel Eisley and Li-Shiuan Peh. High-level power analysis of on-chip networks. In *Proceedings of the 7th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 104–115, 2004.

[6] Mentor Graphics. ModelSim. www.model.com.

[7] Gianluca Palermo and Cristina Silvano. PIRATE: A framework for power/performance exploration of network-on-chip architectures. In *Proceedings of the 14th Intl. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 521–531, 2004.

[8] C. S. Patel, S. M. Chai, S. Yalamanchili, and D. E. Schimmel. Power constrained design of multiprocessor interconnection networks. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, pages 408–416, 1997.

[9] Synopsys Inc. Design Compiler. www.synopsys.org.

[10] Synopsys Inc. PrimePower. www.synopsys.org.

[11] Hang-Sheng Wang, Li-Shiuan Peh, and Sharad Malik. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Proceedings of Design, Automation and Testing in Europe Conference 2005 (DATE05)*, pages 1238–1243. IEEE, March 2005.

[12] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 294–305. IEEE Computer Society Press, 2002.

[13] Terry Tao Ye, Luca Benini, and Giovanni De Micheli. Analysis of power consumption on switch fabrics in network routers. In *Proceedings of the 36th Design Automation Conference (DAC'02)*, pages 524–529, 2002.

[14] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the 34th Design Automation Conference (DAC'00)*, pages 340–345, 2000.

[15] Hui Zhang, Marlene Wan, V. George, and Jan Rabaey. Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs. In *Proceedings of the IEEE Workshop On VLSI '99*, pages 2–8, 1999.

Chapter 5

Routing Aware Switch Hardware Customization

As mentioned, for NoCs to be feasible in today's SoC designs, a NoC architecture with low hardware overhead is required. The NoC architecture typically has a large area overhead when compared to the current bus-based systems. In this chapter, a method for reducing the hardware complexity of the NoC by automatically configuring the architecture of the NoC switches to suit the application traffic characteristics is presented. The crossbar matrix and the arbiters of each switch in the NoC design are customized to support the traffic flows utilizing that switch. This application-specific switch customization is integrated with the ×pipes design flow, the tool flow for application specific Noc design deeply illustrated in this thesis.

5.1 Introduction

A typical NoC switch consists of input ports, arbiters, crossbar matrix, output ports and buffers (present at the input/output ports). In state of the art NoC designs, the internal architectures of all the switches in the NoC are uniform, with all the input ports of a switch connected to all its output ports, through the crossbar matrix and arbiters. Such an architecture is needed when the packet routes cannot be determined at design time, so that during run-time, data from any input port can be sent to any output port of the switch. However, as deeply discussed, in most NoC designs, application-driven system optimization is possible and very helpful, since deterministic routing is employed, where the routes for the packets of the various traffic flows are obtained at design time and the route selection is usually performed during the NoC topology synthesis phase [20]. Thus, the

general architecture leads to an over-design of the NoC switches, resulting in large area-power overhead for the NoC. The hardware design technique proposed in this chapter is aimed to reduce the hardware complexity of the NoC by automatically configuring the architecture of the NoC switches to match the designed routes and the given application traffic pattern. While there are several research works that have addressed several aspects of the application-specific NoC design process, and even if the ×pipes design flow is already tought to support application-driven hardware optimization taking into account area obstruction and power consumption during the topology synthesis, it is very useful and attractive to have a method for automatically customizing the architecture of the switches in the NoC design. The proposed approach is general, and complementary to most of the existing works on topology design, link insertion and buffer sizing and can be used in conjunction with them. In the proposed customization method, the crossbar matrix and arbiters of each switch in the NoC are tuned. That is, based on the designed routes, the proposed method automatically prunes the set of input-to-output connections in the crossbar matrix and arbiters that are not utilized, in each switch of the NoC. The routing-aware switch architecture customization method is integrated in the ×pipes design flow, tackling it at several levels.

5.2 Reference design flow

The customization method proposed in this work is integrated with the ×pipes NoC design flow, toroughly explained in chapter 3. As mentioned the design flow guides the SoC designer to achieve the most power/performance efficient NoC architecture, starting from the application characteristics and is composed of three major design steps: NoC topology synthesis, NoC instantiation and back-end implementation.

- **Step 1: Topology synthesis:**
  In the first step, *SUNFLOOR* [29], a custom CAD tool is used to synthesize the most power/performance efficient NoC topology that satisfies the application requirements. The application traffic characteristics, size of the cores, and the area and power models for the network components are obtained as inputs to the synthesis engine. The tool generates different NoC switches and maps the cores onto the switches. During the synthesis process, it determines deadlock-free routes for the different traffic flows of the application. The tool also produces the 2D floorplan of the synthesized NoC topology. The output produced by

the tool is a text file that defines the synthesized topology, which is fed to the next step.

- **Step 2: Topology instantiation:**
  In the second step, another custom tool ×*pipesCompiler* [30], reads the topology definition file generated by SUNFLOOR and instantiates the RTL description of the NoC components using ×*pipes* [9], a pre-designed SystemC RTL component library. The modules in the component library support a large number of instantiation parameters, such as different input/output ports, buffer sizes, etc. The tool also interconnects the RTL description of the processor/memory cores of the SoC (which are pre-designed components, taken as user inputs) with the RTL code of the network components. The output of this step is the RTL design of the entire NoC that can be simulated and synthesized.

- **Step 3: Back-end implementation:**
  This step includes the utilization of a commercial tool-chain to implement the designed topology at the layout level. The RTL description of the interconnect from the previous step is synthesized and the placement&routing of the design is performed using the commercial tool, Cadence SoC Encounter [31]. The post-layout design is then verified for functional correctness and is used for obtaining accurate performance estimations of the design.

5.3 Routing aware hardware optimization

The objective of the proposed switch customization method is to automatically remove those resources that are not utilized in the NoC from the interconnect hardware, based on the designed topology and the paths that are selected for routing the different traffic flows. The base-line switch architecture of the ×pipes library, before applying the switch customization, is composed of four main blocks:

- **Input ports:** At each input port, the incoming flit and control signals are latched onto registers, so that the critical path of the switch is reduced.

- **Crossbar:** The switch crossbar includes several fully connected sets of multiplexers, one per output port, which connect all the input ports to each output port. The control signals that are required for data

selection for each multiplexer are generated by the arbiters of the corresponding output port.

- **Arbiters:** Each output port of the switch has an arbiter that grants access to one of the input ports requesting the corresponding output port, based on some arbitration policy.
- **Output ports:** In the ×pipes architecture, output buffering is employed, where the flit buffers are present at the output ports of the switch. The output ports also accommodate the combinational logic needed to handle the flow-control operations.

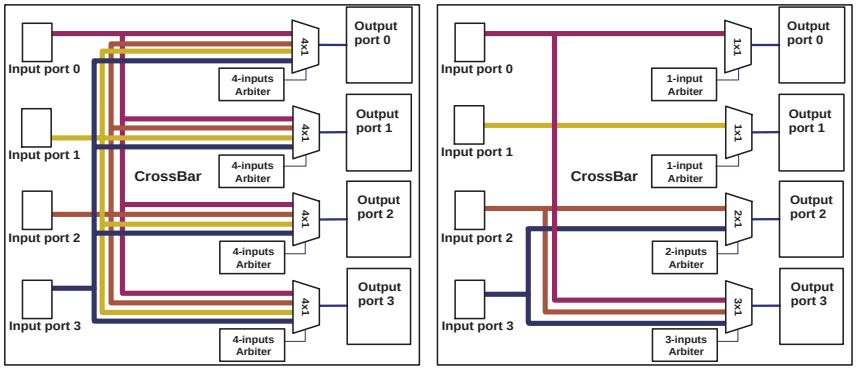
The used procedure analyzes the designed topology and routing paths, and evaluates what input ports actually communicate packets to the different output ports of each switch in the design. Then, we remove those set of input-to-output connections from the crossbar matrix and arbiters that are not used by the chosen routing paths.

Table 5.1: Switch routing table example

	input 0	*input 1*	*input 2*	*input 3*
output port 0	x			
output port 1		x		
output port 2			x	x
output port 3	x		x	x

Example 5.3.1. *As an example, consider the set of input-to-output connections that are required at a particular switch (a* 4×4 *switch) of a NoC (refer Table 5.1), which are obtained from the routing paths established by the topology synthesis procedure. In the table, the presence of a cross signifies that the input-to-output connection is utilized by the routing paths. In Figure 5.1(a), a traditional architecture for this switch is presented, where all the input ports are connected to all the output ports of the switch. In Figure 5.1(b), the switch architecture obtained by the proposed method is represented, where the crossbar matrix and arbiters are customized to match the required input-to-output connections of the designed routes. The switch customization for this example, leads to 56.25% reduction in the input-to-output connections of the switch.*

To achieve this switch customization, the design flow was tackled at two points, i.e. at the component library level (*hardware level*) and at the RTL code generation level (*software level*). These levels are explained in detail in the following sub-sections:



(a) Traditional design (b) Customized design

Figure 5.1: Switch architecture before and after the routing aware customization

5.3.1 Hardware-Level Customization Support

A layer was added to the configuration of the switch building blocks in the ×pipes SystemC RTL macros, specifying a set of parameters for each instance of the multiplexer and the arbiter. More in detail, before the enhancement, the multiplexers were configured automatically, connecting all the input ports to each output port. After the customization enhancement, each multiplexer module is also configured in terms of the number of input ports that need to be connected to each output port. In the same way, each arbiter is configured for the number of ports for which it has to arbitrate for. While the individual multiplexer and arbiter modules are parameterized in this fashion, a tool chain is needed to instantiate the different modules for the designed NoC and to interconnect the sub-blocks with the switch top module. To achieve this, the ×pipesCompiler tool was extended at the software level to customize, instantiate and interconnect the different modules together.

5.3.2 Software-Level Customization Support

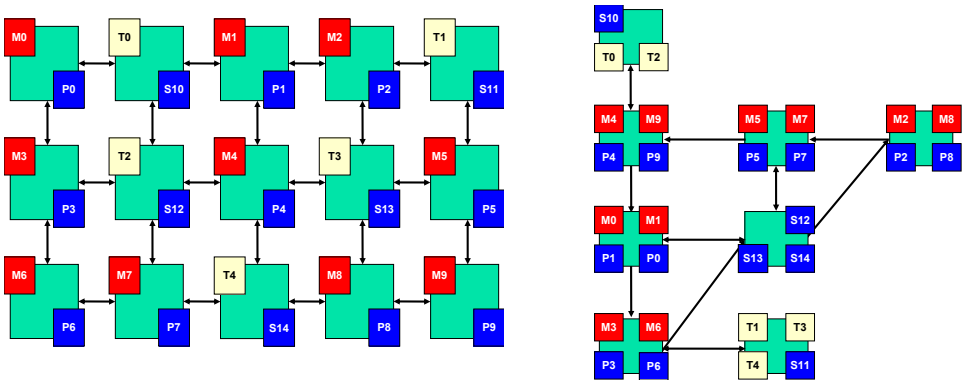
A software layer integrated with the ×pipesCompiler tool in the design flow , that analyzes the routing paths and finds the unneeded hardware in the switches, was defined. The software thus:

- parses the topology definition file from SUNFLOOR, extracting the information about every single routing path,
- for each arbiter and multiplexer, defines a set of parameters that summarize how many and which input ports require access to the related output port,

- for all the switches, generates a top module that instantiates all the sub-blocks and defines the connectivity between them according to the routing needs.

The output of this switch customization enhanced NoC instantiation tool is the RTL description of the customized NoC design that can be simulated and synthesized

5.4 Customization method effectiveness evaluation



(a) Mesh topology (b) Custom topology

Figure 5.2: Mesh and application-specific custom topologies for the MULT benchmark. The P0-P9 are the processors, T0-T4 are the hardware cores, M0-M9 are the private memories and S10-S14 are the shared devices. The shaded boxes connecting the cores are the switches in the design.

The switch customization method was applied on the NoCs designed for several SoC applications: *Multimedia design (MULT-30 cores), IMage Processing design 1 (IMP1-25 cores), IMage Processing design-2 (IMP2-21 cores), FFT based SoC (FFT-29 cores), Data Processing SoC (DP-15 cores) and SoC implementing a DES encryption system (DES-19 cores).* In the next sub-sections the details of one of the applications (MULT, the largest of the benchmarks) and the customization impact on the different designs are reported.

5.4.1 Experiments on the Multimedia benchmark

This subsection presents the designs obtained by the proposed approach for two different topologies (a regular and a custom NoC topology) that are

Table 5.2: Total switch area of the designs

Topology	*Total area base-line* (mm^2)	*Total area customized* (mm^2)	*Area reduction* (%)	*# I-to-O links reduction* (%)
5×3 Mesh	0.73	0.51	30.14	69.63
Custom	0.45	0.31	31.11	66.38

Table 5.3: Combinational area of the switches for the designs

Topology	*Comb. area base-line* (mm^2)	*Comb. area customized* (mm^2)	*Comb. area reduction* (%)
5×3 *Mesh*	0.32	0.158	50.63
Custom	0.22	0.09	59.09

used to interconnect the cores of the MULT SoC benchmark. Such different topologies are used to show the generality of the proposed customization method. The MULT SoC consists of fifteen processor/hardware cores, ten private memories for the different processors and five shared slave devices. The regular topology is a 5×3 mesh, which is hand-designed to suit the application characteristics of the benchmark (presented in Figure 5.2(a)) [9]. The second topology is an application-specific custom topology obtained from the SUNFLOOR tool (presented in Figure 5.2(b)). To achieve the optimum network throughput, 3 flit buffers are utilized at each output port [9]. The total area and the area of the combinational blocks of the switches for the two topologies, for the base-line design and for the design where the proposed switch customization technique is applied are shown in Tables 5.2 and 5.3. The area numbers are obtained from synthesizing the RTL code of the different switches in the design. As seen from the table, the use of the switch customization technique leads to a large reduction (an average of 30.63%) in the total switch area of the design. As the proposed procedure optimizes the switch crossbar and multiplexers, which are predominantly combinational blocks, a large savings in the combinational area of the switches is obtained, which leads to the significant total switch area savings.

The power consumption of the different architectures are shown in Table 5.4. The power numbers are based on the switching activities of the components, which are obtained from functional simulations. For the power consumption estimations, the accurate switching resistance and capacitance values of the components, obtained from the post-synthesis net-lists of the NoC designs, is also considered. The synthesis experiments were performed using Synopsys Design Compiler [32], with 0.13μ technology library, an operating frequency of 500 MHz and an operating voltage of 1.2 V. Clock-gating was also used in the architectures, so that the elements that are not heavily

Table 5.4: Switch power consumption for the designs

Topology	*Power base-line* (mW)	*Power customized* (mW)	*Power reduction* (%)
5×3 Mesh	66.2	29.4	55.6
Custom	36.3	28.6	21.2

utilized have a lower power consumption value. The proposed customization technique also leads to a large reduction in the power consumption of the switches (an average of 38.4%) for both the topologies.

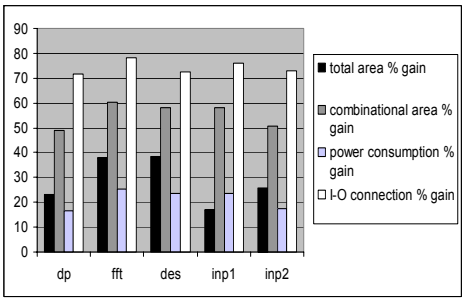


Figure 5.3: Switch area, power and input-to-output connection savings for the SoC designs

5.4.2 Experiments on SoC benchmarks

The best topologies for the different SoC designs were synthesised using SUNFLOOR. The area, power and input-to-output connection savings for the different designs for the customized architecture, when compared to the base-line architecture are shown in Figure 5.3. For all the designs, the switch customization technique leads to significant reduction in the switch area (28% on average) and power consumption (21% on average) values.

Finally, as the proposed customization technique leads to a reduction in the crossbar and arbiter complexity, the critical path of the switches also reduces significantly. This is because, the critical path of the switch in the ×pipes architecture is the path from the input ports to the output ports, that traverses the arbiters and the crossbar multiplexers. As an example, for a 4×4 base-line switch architecture, the critical path is around 1 ns, while for the customized 4×4 switch architecture, on average, the critical path is around 0.57 ns. These timing values are obtained from RTL synthesis of

the switches. Thus, the customization technique also leads to a significant speed-up (of 75%) of the NoC.

Bibliography

[1] L.Benini, G.De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, pp. 70-78, Jan. 2002.

[2] D.Wingard, “MicroNetwork-Based Integration for SoCs”, Proc. DAC, pp. 673-677, Jun 2001.

[3] M. Sgroi et al., “Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design”, Proc. DAC, pp. 667-672, June 2001.

[4] S. Dutta, R. Jensen, A. Rieckmann, “Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems”, IEEE Design and Test of Computers, pp. 21-31, Vol. 18, No. 5, Sep/Oct 2001.

[5] A. Artieri, V. D Alto, R. Chesson, M. Hopkins, M. C. Rossi, “Nomadik Open Multimedia Platform for Next-generation Mobile Devices”, STMicroelectronics Technical Article TA305, 2003, available at http://www.st.com

[6] S. Kumar et al., “A Network on Chip Architecture and Design Methodology”, Proc. ISVLSI, pp. 117-122, April 2002

[7] P. Guerrier, A. Greiner, “A generic architecture for on-chip packet-switched interconnections”, Proc. DATE, pp. 250-256, March 2000.

[8] K. Goossens et al., “The Aethereal network on chip: Concepts, architectures, and implementations”, IEEE Design and Test of Computers, Vol. 22(5), pp. 21-31, Sept-Oct 2005.

[9] S. Stergiou et al., “xpipes Lite: A Synthesis Oriented Design Library For Networks on Chips”, Proc. DATE, pp. 1188-1193, March 2005.

[10] I. Saastamoinen et al., ”Proteo interconnect IPs for networks-on-chip”. In Proc. IP Based SoC Design, France, 2002.

[11] E. Bolotin et al., “QNoC: QoS architecture and design process for network on chip”, Journal of Systems Architecture, Feb. 2004.

[12] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", DATE 2005.

[13] D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multi-Processor Systems-on-Chip", IEEE Transactions on Parallel and Distributed Systems, Feb 2005.

[14] P. Pande et al., “Design of a Switch for Network on Chip Applications”, pp. 217-220, Porc. ISCAS 2003.

[15] C. Zeferino et al., “A Parameterizable Interconnect Switch for Networks-on-Chip”, pp. 204-209, Proc. SBCCI 2004.

[16] T. Bjerregaard, J. Spars, “A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chips”, Proc. DATE 2005.

[17] J. M. Kim et al., “A Low-Latency Router Supporting Adaptivity for On-Chip Interconnects”, pp. 559-564, Proc. DAC, June 2005.

[18] K. Lee et al., “A High-Speed and Lightweight On-Chip Crossbar Switch Scheduler for On-Chip Interconnection Networks”, ESSCIRC 2004.

[19] T. Ye et al., “Analysis of Power Consumption on Switch Fabrics in Network Routers”, Proc. DAC 2002.

[20] J. Hu, R. Marculescu, “Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures”, Proc. DATE, March 2003.

[21] S. Murali, G. De Micheli, “SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs”, Proc. DAC 2004.

[22] W. Hu et al., “Thermal Aware Placement for NoC Architectures”, pp. 430-437, Proc. ICCD 2004.

[23] A. Hansson et al., “A unified approach to constrained mapping and routing on network-on-chip architectures”, pp. 75-80, Proc. ISSS 2005.

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

[25] T. Ahonen et al. "Topology Optimization for Application Specific Networks on Chip", Proc. SLIP 04.

[26] K. Srinivasan et al., “An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks”, Proc. ICCAD ’05.

[27] U. Y. Ogras, R. Marculescu, “Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion”, Proc. ICCAD 2005.

[28] J. Hu, R. Marculescu, “Application Specific Buffer Space Allocation for Networks on Chip Router Design”, Proc. ICCAD 2004.

[29] S. Murali et al., “Designing Application-Specific Networks on Chips with Floorplan Information”.

[30] A. Jalabert et al., “×pipesCompiler: A Tool for Instantiating Application-Specific Networks on Chips”, Proc. DATE 2004.

[31] www.cadence.com

[32] www.synopsys.com

Chapter 6

65 nm NoC Design

Networks-on-Chip (NoCs) have been proposed as a scalable solution to both physical design issues and increasing bandwidth demands. However, this claim has not been fully validated yet, since the design properties and tradeoffs of NoCs have not been studied in detail below the 100 nm threshold.

This chapter is aimed at shedding light on the opportunities and challenges, both expected and unexpected, of NoC design in nanometer CMOS. We present fully working 65 nm NoC designs, a complete NoC synthesis flow and detailed scalability analysis.

While the key advantages offered by the use of NoCs have been largely accepted nowadays, the practical implementation of NoCs in very deep submicron technology, below the 100 nm threshold, is a very open challenge. The crucial issue is again related to wiring. Even if capacitive loads and propagation delays can be controlled much better than in shared buses, 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.

This chapter presents a detailed description of some updates required by the reference ×pipes design flow to supprto 65 nm technology and outlines some of the tradeoffs that a next-generation back-end implies. Link performance, placement issues, scaling results in shifting from 90 nm to 65 nm technologies, and the degrees of freedom allowed by the availability of multiple libraries (with different power/performance tradeoffs) at the same technology node are explored.

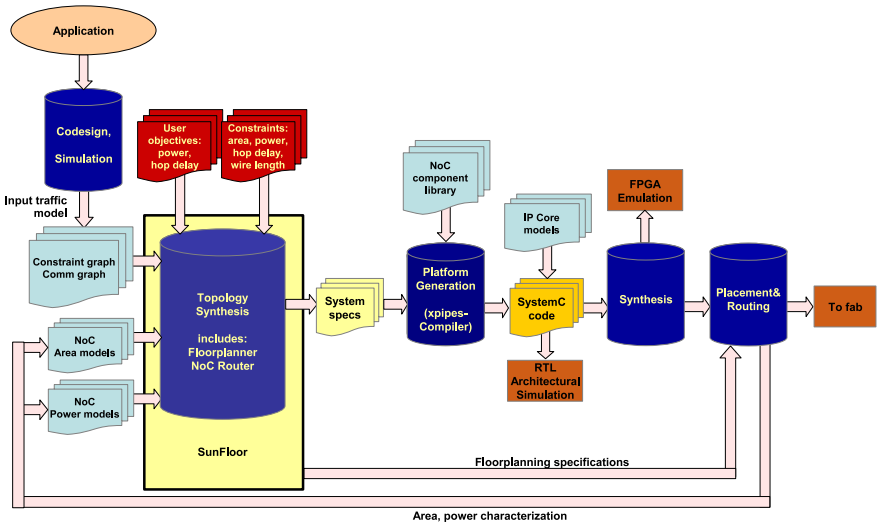


Figure 6.1: Our proposed complete NoC design flow for MPSoCs

6.1 NoC Design Flow

An overview of the complete ×pipes flow for designing NoCs for MPSoCs is presented in Chapter 2 and Chapter 3. As mentioned this flow comprises several tools that are integrated together. First, SunFloor, which automates the front-end process of NoC design. Second, ×pipesCompiler, which automates the architecture generation phase (leveraging the ×pipes library of NoC components). Finally, several industrial tools that automate the back-end processes, *i.e.* the logic synthesis and physical design. In this chapter, approaching to deep sub micron technologies, some updates were required to face modern technology node features. For exemple, the SunFloor tool, used to synthesize the best custom NoC topology for a given MPSoC platform, required, in addition to the already discussed switch area and power models, derived using the procedures deeply explained in Chapter 4, detailed area and power consumptions models for the links. This models for the links are derived using the same methodologies proposed in Chapter 4 and taking into account as architectural parameters their length and flit width. Morever, as said, the topologies synthesized by SunFloor required that the links could be traversed in a single clock cycle. In the updated version of the design flow, this assumption was removed by including in SunFloor the pre-characterization of link delay information. Therefore, Sunfloor automatically pipelines long links in the design, based on the targeted frequency of operation. When a link is pipelined and its latency increases, SunFloor considers this information to determine the average latency of the NoC and, therefore,

takes it into account in its cost metrics. The NoC component library taken as reference did not require any change, due to the its high parameterizability and thanks to the fact that it supports link pipelining, where logical buffers are interleaved along links. This feature reduces signal propagation delays, and as we illustrate in our analysis (Section 6.2.1) and results (Section 6.3), it is a very relevant element in latest technology nodes.

6.1.1 Flow Back-End

The ×pipes back-end flow based on standard cell synthesis needed to be updated to face the need, imposed by DSM technologies to tightly close the gaps between the different steps of the flow. As previously mentioned, in the previously illustrated verion of the flow, the first step of the implementation flow consists of the logic synthesis phase, obtained automatically by utilizing standard Synopsys tools. In a second phase, the flow scheduled a floorplanning phase, which was performend inside the Cadence SoC Encounter suite devoted to the place and route procedure. Thus, in the traditional flow logic synthesis and placement are conceived as two clearly decoupled stages, even if informations obtained by the area and power models are used by SunFloor to define the topology floorplan. As shown by the experiments presented in previous chapters, this flow achieved reasonable enough results for 130 nm NoC design. Anyway the assumption considering the two steps indipendently appears to be substantially inadequate at the 65 nm node. The origin of the problem lies in the same concept that enables the splitting of the two steps, namely, *wireload models*. Wireload models are 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 wireload models becomes very complex, and the prediction inaccuracies become critical. While performing the presented 65 nm test explorations, unacceptable performance degradation due to inaccuracies in wireload estimation could be observed. 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. 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 wireload expectations, not only a performance loss is certain, but the placement runtime skyrockets. To address this issue *placement-aware* logic synthesis tools, such as Synopsys Physical Compiler [36] were introduced in an update version of the flow. In this type of flow, after a very quick initial logic synthesis based on wireload 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 do not incur major penalties. In addition, other wiring- and placement-related problems were observed within soft macros due to congestion. In test designs, placements tools performed poorly both when modules had to be placed within too small and too wide fences. While the former case is clearly understandable, the unexpected latter effect can be attributed to the placement tool heuristics, which are probably performing worse when the solution space becomes very large. Thus, the problem must be solved by proper tuning of the spacing among the soft macro fences and, consequently, accurate area models of the NoC modules are required to avoid very time-consuming manual interventions in the synthesis process.

This procedure is followed by using 90 and 65 nm technology libraries by a partner foundry, tuned for different performance/power tradeoffs, with different threshold and supply voltages. While full custom design would certainly improve results, it would also greatly decrease flexibility and increase design time.

To take profit of the advantages derived by the use of a complete suite developed by the same vendor, and to avoid small file portability problems that may arise using software provided by different companies (although Phisical Compiler and SoC Encounter are conceived for mutual compatibility), the original design flow was updated to perform the detailed placement&routing step within Synopsys Astro [35]. Two of the main placement strategies commonly available within industrial tools are *virtual flat* and *soft macros*. In the former option, the tool is fed with the complete design, and albeit placement guidelines can be given, the tool is allowed to modify the global floorplan. This theoretically allows for maximum optimization and better handling of design violations; unfortunately, for a design as large as a whole NoC-based chip, it appeared to be extremely demanding on system resources (more than 5 GB of RAM were needed by the placement process, and runtimes were unacceptable). The *soft macro* alternative is based on rigid *fences* which separate floorplan areas. Each module of the design is assigned to one such area;

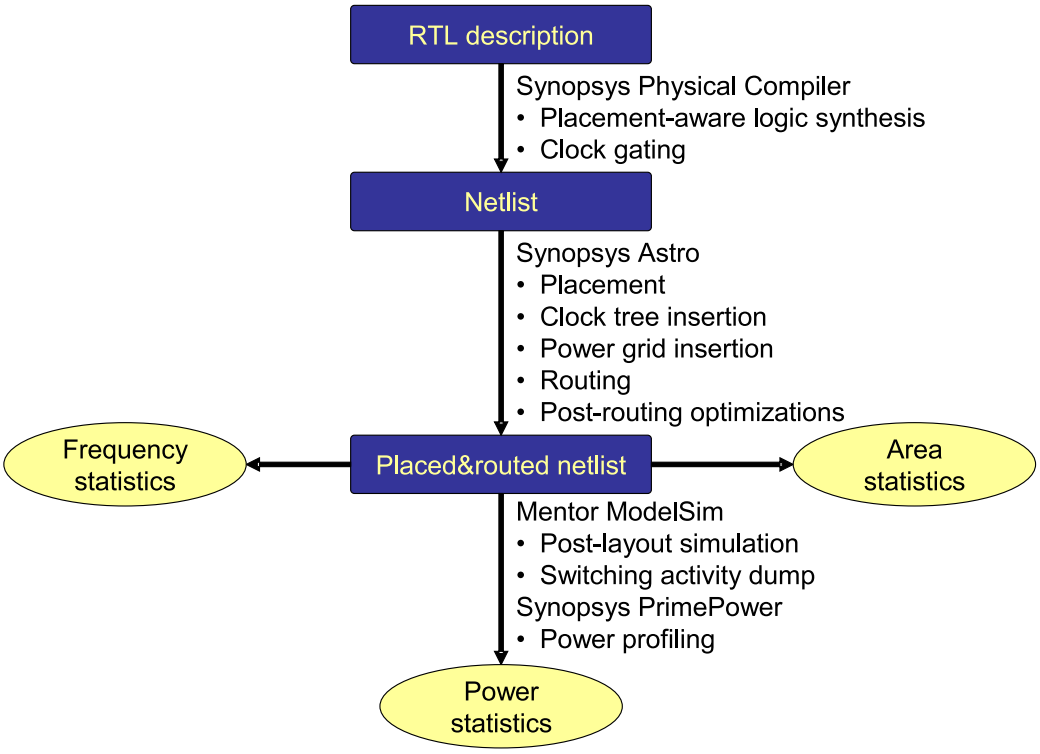


Figure 6.2: The synthesis flow for ×pipes

the tool is able to freely perform placement operations within such modules and areas, but it is not allowed to trespass fences. The option integrated in the flow is a mix of the two strategies, aimed to obtain optimal results with reasonable computational effort. First, Astro was fed with a rough floorplan, generated either manually or by SunFloor. This floorplan contains *hard macros* and soft macros, separated by fences. The hard macros represent IP cores and memories, and are modeled as black boxes. Hard macros are defined with a Library Exchange Format (LEF) file and a Verilog Interface Logical Model, and obstruct an area of our choice. These boxes also obstruct some of the metal layers laying directly above; the exact number of obstructed levels is configurable, depending on how many metal layers the IP cores are supposed to require and on whether over-the-cell routing for the NoC wires *vs.* between-the-cell is preferred. Soft macros enclose the modules of ×pipes; by constraining the placement tool to operate on one tile at a time, faster runtimes can be achieved. For proper results, however, it becomes necessary to specify rough timing constraints at the soft macro boundaries; this was achieved thanks to the pre-characterization of the links (please see Section 6.2.1 below).

The next step in the flow is clock tree insertion. While a separate clock tree could be added to each soft macro, it would be difficult to control the skew when joining the trees together and attaching them to a single clock source. Therefore, this step operates at a global level. The clock tree is added by leveraging *clock borrowing* algorithms in the tools; in other words, clock skews are exploited to accommodate the delay properties of the circuits, by supplying wider clock periods where the logic paths are most critical. Once the clock tree has been generated, its wires are kept untouched within the tool, to prevent further skews from appearing.

At this point, the power supply nets are added. To improve supply stability, the *power grid* scheme was chosen instead of the traditional *power ring*; power nets are distributed from the topmost metal layers of the chip, instead of from a ring around the die. This minimizes *IR drops* (voltage drops and fluctuations due to resistive effects in the supply networks and to the current draw). After the power nets have been routed, the tool begins to route the logic wires. After an initial mapping, *search&repair* loops are executed to fix any violations.

As a final step, post-routing optimizations are performed. This stage includes crosstalk minimization, antenna effect minimization, and insertion of filler cells. Finally, a *signoff* procedure can be run by using Synopsys PrimeTime [38] to accurately validate the timing properties of the resulting design.

6.1.2 Post-Layout Analysis

Post-layout verification and power estimation is achieved as described in relationship with the original flow.

6.2 Wire Design in 65 nm Technologies

As mentioned above, wires are a very important element in sub-100 nm technologies. Experiments with a 65 nm design flow have shown that wires are critical both within NoC modules and for inter-module links. The problems related to the wires within a module, as already mentioned, were solved by performing a placement aware synthesis of the modulese. The following subsections will briefly describe the results of this analysis at global level.

6.2.1 Link Delay and Link Power

In order to assess the impact of global wires, 65 nm NoC links were studied in isolation from the NoC modules. An overview of some of our analyses can be found in Figure 6.3. The results show that several factors have to be considered in link design. Two obvious factors are link length and desired clock frequency. Short links or links clocked at a very slow frequency do not pose problems. However, as either length or target frequency are increased, an undesired effect appears in the form of high power consumption. The reason is that when links are pushed for high performance, back-end tools automatically insert large amounts of buffering gates, increasing the energy cost of the links. In some validation experiments, the feasibility threshold of high-frequency or very long links was in some cases set by the inability to decrease delay further and in some cases by crosstalk concerns. In other words, the added buffers would sometimes be too large to be safely deployed.

Another extremely important dependency was on the specific technology library in use. As Section 6.3 shows, especially at the 65 nm node, a single "technology library" no longer exists for standard cell design. In fact, manufacturing technologies are spreading across a variety of libraries optimized for specific uses, such as low power or high performance, with several intermediate levels featuring for example different threshold voltage values. In this case, if very low power libraries are used, the size and speed of the buffers that can be interleaved along wires becomes dramatically inferior, which results in much tighter constraints on frequency of operation or length. Figure 6.3(a) reports power consumption for a 65 nm low power library tuned for a low threshold voltage (called LP-LVT in the following), and therefore for a power/performance tradeoff. Figure 6.3(b) is based on a 65 nm low

power library tuned for a high threshold voltage (LP-HVT), and therefore for minimum power consumption. As can be seen, the LP-HVT library is substantially more power effective than the LP-LVT library, but puts much tighter constraints on link feasibility.

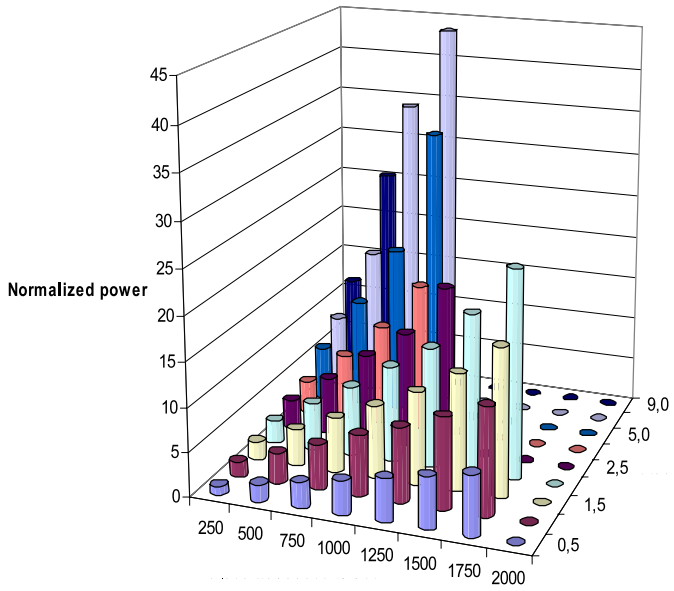
Link repeaters can be used to tackle this issue. Repeaters can be defined as clocked registers along links. By providing one or more extra clock periods to traverse long distances, they solve the link infeasibility problem at a much lower cost than that of deploying whole NoC switches in the middle of the links. In some cases, repeaters may even produce more power-effective solutions than regular wire buffering along particularly critical links, but at a performance cost (*i.e.*, one extra cycle of latency). In all cases, the NoC flow control protocol must be designed in such a way as to enable a transparent insertion of the repeaters. Alternatively, repeaters must contain extra logic to properly handle the flow control handshake signals. In ×pipes design flow, support for pipelined links is present at all levels of abstraction, starting from the high-level SunFloor tool down to final layout tools.

6.3 Experimental Results

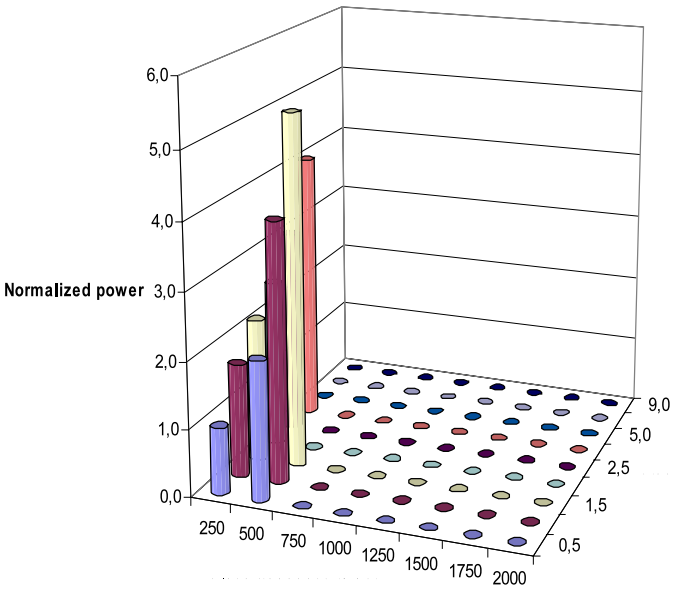
6.3.1 Technology Scaling from 90 to 65 nm

In a first set of results (see Figure 6.4) the effect of scaling is studied when the ×pipes switches are synthesized in four different libraries, namely, two 65 nm and two 90 nm ones, tuned for different power/performance tradeoffs (LP-LVT and LP-HVT). In these experiments, switches were fully placed&routed, including the addition of a clock tree. Then, syntheses were tuned for the maximum operating frequency. To this end, clock gating option was disabled. As can be seen in the results, 65 nm libraries provide large opportunities for improvement over their 90 nm predecessors. In fact, observed power consumptions about 50% lower (up to 75% lower when comparing the LP-HVT versions) could be observed, and area savings of 40-50%.

It is also important to observe the large difference in synthesis results among two different libraries at the same technology node. For the 65 nm case, the LP-HVT library is consuming one order of magnitude less power than the LP-LVT variant. In addition, the results indicate that this performance spread is increased compared to the 90 nm libraries. For example, by observing the achievable clock frequency, LP-HVT 65 nm libraries reach 50% lower frequencies than their 90 nm equivalents, but LP-LVT 65 nm libraries are actually 25% faster than their 90 nm equivalents. This trend suggests that new degrees of freedom are available to designers in new technology



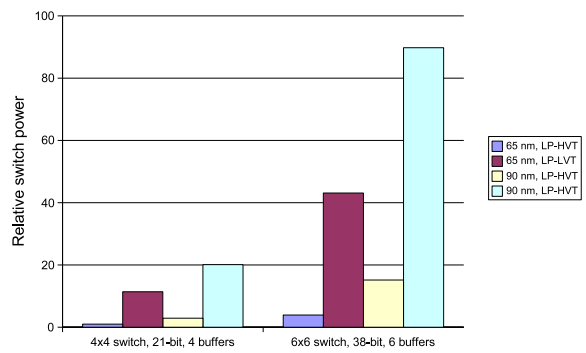
(a) performance/power oriented 65 nm library (LP-LVT)



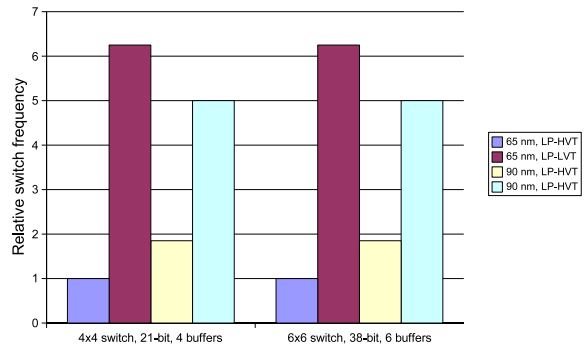
(b) very low-power 65 nm library (LP-HVT)

Figure 6.3: Power consumption of 38-bit links of varying lengths at different operating frequencies. Values normalized to shortest link at slowest frequency. Missing columns represent infeasible length/frequency combinations.

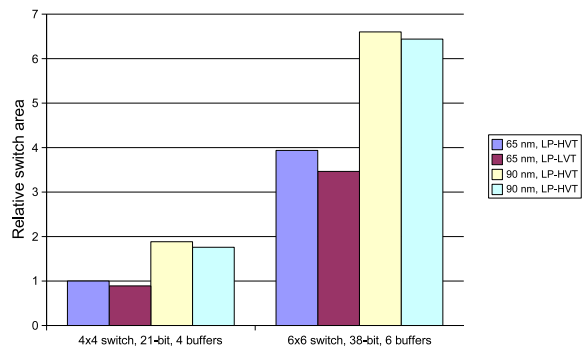
nodes.



(a) power



(b) operating frequency



(c) area

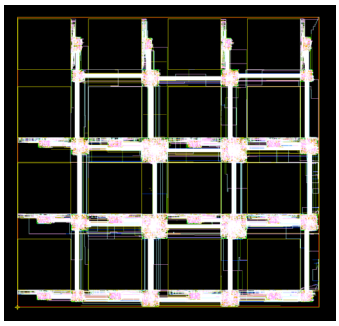
Figure 6.4: Analysis of two representative ×pipes switches in different technology libraries. Figures normalized to the 4x4 switch in the LP-HVT library.

In a second set of experiments complete NoC topologies were analyzed, namely 4x4 meshes (see Figure 6.5). The topologies were synthesized with the higher-performance version of the 90 nm and 65 nm libraries presented above.

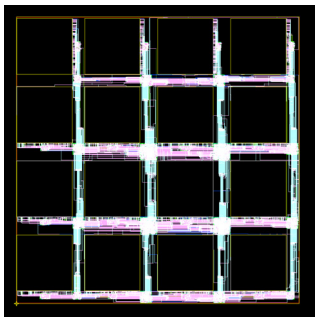
	90 nm, 1 mm^2	65 nm, 1 mm^2	65 nm, 0.63x0.63 mm^2
Relative frequency	1.00	1.25	1.25
Relative cell area	1.00	0.49	0.48
Relative power	1.00	0.66	0.63
Relative bandwidth	1.00	1.25	1.25
Relative power/bandwidth	1.00	0.53	0.50
Relative link power	1.00	1.16	0.71

Table 6.1: Synthesis results on three 4x4 NoC meshes. Figures normalized to the 90 nm results.

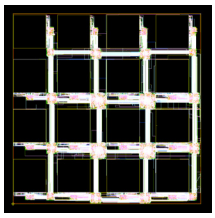
For the 90 nm case, IP cores were modeled as 1 mm^2 obstructions, while, for the 65 nm topologies, tho different IP models were used: one assuming the same size of 1 mm^2 and one assuming a scaled size, where IP cores require 0.63x0.63 mm^2. The area scaling factor is derived from datasheet analyses and experiments on adder designs.



(a) 90 nm, 1 mm^2 obstructions



(b) 65 nm, 1 mm^2 obstructions



(c) 65 nm, 0.63x0.63 mm^2 obstructions

Figure 6.5: Three 4x4 ×pipes meshes.

As the results in Table 6.1 show, the jump to the 65 nm node presents large advantages in area, power consumption and maximum achievable frequency. The most impressive result is the power over bandwidth metric, which improves by a factor of 2. The gains are similar to those for single switches reported above, except for the power consumption figure, which features smaller savings. The main reason is that, in regular meshes, links are generally short (at most 1.2 mm in the meshes with 1 mm^2 cores), enough so to not represent a performance bottleneck even at the 65 nm node. However, in 65 nm technology, there is still a power consumption penalty to be paid due to the extra required buffering along the wires. For this reason, the links in the 65 nm mesh with 1 mm^2 cores, which are the most constrained of this experiment due to a mix of technology properties, length and operating frequency, are the most power-expensive and have an impact on overall figures.

The scaled 65 nm mesh is less link-constrained, leading to slightly smaller area and power consumption.

6.3.2 Topology design

Next, the SunFloor tool was applied to a high bandwidth application, typical of today's video applications, and to a low bandwidth application, typical of mobile applications.

6.3.3 High Bandwidth Application

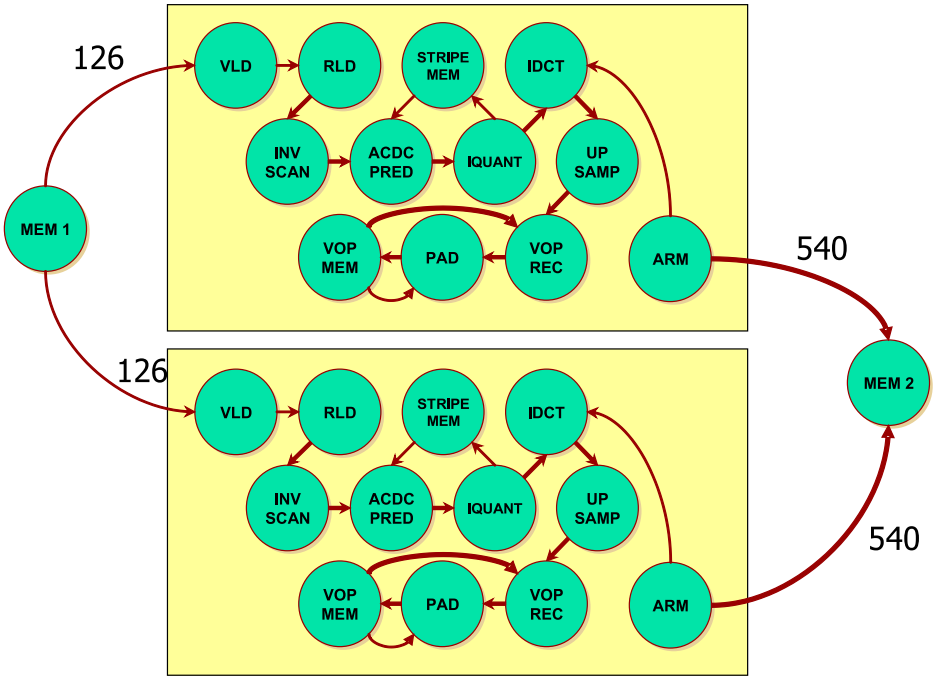


Figure 6.6: Enhanced VOPD application, called DVOPD, with the capability to decode two streams in parallel.

The objective of this experiment, whose results are outlined in Table 6.2, was twofold. First, it aimed at finding the impact of technology scaling on the sizes of the communication architectures and on the topologies required to match the application characteristics. Second aim was to analyze the impact of the choice of libraries (*i.e.* LP-LVT or LP-HVT) used for the technology process. The performed comparisons are:

- *Same Platform for both 90 nm and 65 nm* In this experiment, the same platform was assumed to be used in 90 nm and 65 nm nodes, and the

Library, Application	Max Freq.	Switch Count	Largest Switch	Switch Power	Link Power	Total NoC Power	Avg. latency
90 nm LP-LVT, DVOPD	400 MHz	4	10x9	140.83 mW	57.58 mW	198.3 mW	3.42 cycles
90 nm LP-HVT, DVOPD	-	-	-	-	-	-	-
65 nm LP-LVT, DVOPD	400 MHz	4	10x9	59.13 mW	24.46 mW	83.59 mW	3.91 cycles
65 nm LP-HVT, DVOPD	-	-	-	-	-	-	-
65 nm LP-LVT, DVOPDX2	800 MHz	6	7x6	131.99 mW	47.98 mW	179.97 mW	4.24 cycles
65 nm LP-LVT, TVOPD	800 MHz	10	7x7	189.35 mW	79.93 mW	269.29 mW	4.35 cycles

Table 6.2: High Bandwidth Application Results

impact of technology scaling on the designed NoCs was evaluated. This is often done by system designers, who reuse the same platform (possibly as a part of a bigger system) to reduce the design and verification efforts. This analysis is based on a *Dual Video Object Plane Decoder (DVOPD)* application, where two video streams are decoded in parallel by utilizing 26 processing/hardware cores. This application is a scaled version of the VOPD benchmark presented in [8]. The communication characteristics of the DVOPD benchmark are shown in Figure 6.6. The core (each core in the application is represented by a vertex in Figure 6.6) size was assumed of 1 mm^2 in 90 nm technology and would shrink to 0.63x0.63 mm^2 when migrating to 65 nm.

- *Higher Bandwidth Platform in 65 nm* To evaluate the scalability of the interconnect in 65nm technology, a second benchmark was additionally considered , where the bandwidth requirements of the DVOPD were doubled, referred to as *DVOPDX2*.

- *Larger and Higher Bandwidth Platform in 65nm* As the core sizes are smaller in 65 nm technology, more cores could fit on the chip in comparison to 90 nm. Therefore, to take this effect into account, a third benchmark was considered, called *TVOPD*, where 3 video streams were decoded in parallel following the same graph as in the DVOPD application (shown in Figure 6.6), instead of 2 video streams as in DVOPD. This new design consisted of 38 cores. We also assumed that the base application bandwidth requirements would be doubled, as in DVOPDX2.

The characteristics of the NoCs synthesized by our tool chain for the benchmarks are shown in Table 6.2. The average latency presented in the table is defined as the latency for a head flit of a packet to move from the

output of the initiator NI to the input of the target NI, when there is no congestion in the network. In this study, the network flit width was fixed to match the data width of the cores (equal to 32 bits). The DVOPD application bandwidth requirements demanded a 400 MHz operation for the NoC, which was automatically determined by the SunFloor tool. Several interesting facts could be observed:

- The switches designed using the LP-HVT libraries were not able to meet the required frequency and bandwidth requirements, due to their focus on very low power operation. Thus, only the LP-LVT libraries resulted in valid designs for the benchmark.
- For the DVOPD application (represented by the rows 1-4 in the table), the best topology synthesized by our tool flow remains the same (*i.e.*, same switch count and sizes), with both 90 nm and 65 nm libraries. The ratio of link power to switch power consumption, however, increased when moving to the 65 nm technology. This is despite the fact that, for this benchmark, the core sizes were smaller in 65 nm technology, which led to an overall reduction in the total length of wires. The reason for this reduction was that the switch power consumption reduces by 55% when moving from 90 nm to 65 nm, whereas the wire power consumption was reduced only by 31%. This result is in agreement with the findings in Table 6.1.
- The number of switches needed increased to 6 and 10 for the DVOPDX2 and TVOPD scenarios, respectively. This is because these benchmarks have doubled bandwidth requirements with respect to the DVOPD application; thereby, they require double the operating frequency for the NoC (800 MHz). In fact, as big switches cannot satisfy such a high operating frequency, the SunFloor tool synthesized a design with many smaller switches. As the topology size increases, as expected, the average head flit latency also increases.
- The 65 nm technology is very power efficient. In fact, this technology supported twice the application bandwidth requirements (the DVOPDX2 benchmark) at a lower power consumption than the 90 nm technology library.

6.3.4 Effect of Link Pipelining

The SunFloor tool automatically pipelines long links, based on the required NoC operating frequency and the link lengths obtained from the floorplan of

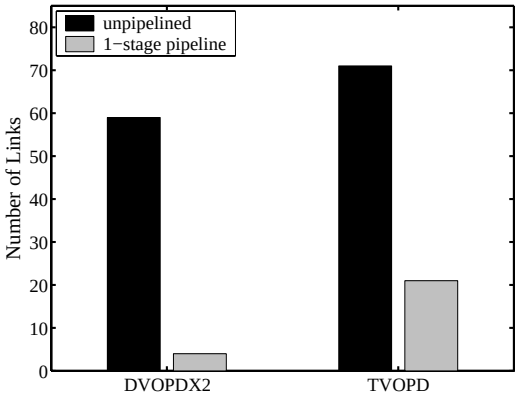


Figure 6.7: Amount of pipelined links in two sample benchmarks.

Library, Application	Max Freq.	Switch Count	Largest Switch	Switch Power	Link Power	Total NoC Power	Avg. latency
90 nm LP-LVT	50 MHz	2	11x11	10.46 mW	5.47 mW	15.93 mW	3.94 cycles
90 nm LP-HVT	50 MHz	2	11x11	4.27 mW	2.1 mW	6.36 mW	3.94 cycles
65 nm LP-LVT	50 MHz	2	11x11	4.72 mW	2.31 mW	7.03 mW	3.94 cycles
65 nm LP-HVT	50 MHz	5	9x9	2.61 mW	1.65 mW	3.86 mW	3.94 cycles

Table 6.3: Low Bandwidth Application Results.

the design. Such link pipelining is needed for NoCs that require a high operating frequency. As an example, without link pipelining support, the NoC for the DVOPDX2 and TVOPD designs could only operate at 500 MHz, while the application bandwidth requirements necessitate 800 MHz operation. In Figure 6.7, the number of pipeline stages required for the different links in the DVOPDX2 and TVOPD designs was plotted. A non-pipelined link requires one clock cycle for traversal, while a link with a single pipeline stage requires two clock cycles. For all these benchmarks, all the links could be traversed within 2 clock cycles. As the design complexity increases (moving to the TVOPD design), the portion of links that require pipelining also increases. The SunFloor tool automatically considers the increase in latency due to link pipelining when determining the average latency of the NoC, and is therefore able to account for the overhead in its performance metrics.

6.3.5 Low Bandwidth Application

NoCs can be used effectively not just for high bandwidth applications, but also for low bandwidth applications that have tight power budget constraints. Therefore, in a final set of experiments, is demonstrated that the performance of NoCs forthcome requirements of low-power applications and mobile systems.

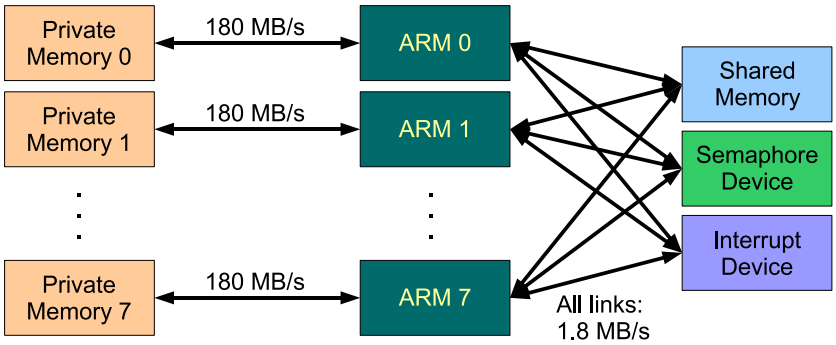


Figure 6.8: DES benchmark.

Application	Library	Bandwidth per mW
DVOPD	90 nm LP-LVT	67.27 MB/s/mW
DVOPD	65 nm LP-LVT	159.64 MB/s/mW
DES	90 nm LP-LVT	94.62 MB/s/mW
DES	90 nm LP-HVT	229.56 MB/s/mW
DES	65 nm LP-LVT	207.68 MB/s/mW
DES	65 nm LP-HVT	378.23 MB/s/mW

Table 6.4: Bandwidth supported per milliwatt of power consumption

In order to represent mobile applications with these low power requirements, the *DES* encryption benchmark implemented on 19 cores was considered. The communication characteristics for the benchmark are shown in Figure 6.8. The designed NoCs for the 2 different technologies for the LP-LVT and LP-HVT libraries are shown in Table 6.3. As seen from the table, for low power requirements, the LP-HVT libraries are far superior to the LP-LVT libraries. As an example, for the DES mapping in the 65 nm LP-HVT technology, the resulting chip layout is also presented in Figure 6.9.

In addition,the energy efficiency of the NoCs for the different applications was investigated across the different technology generations. The total bandwidth required by the DVOPD application is 13.34 GB/s, while for the DES application, it is 1.46 GB/s. In Table 6.4, the bandwidth supported per milliwatt of power consumption by the different NoC designs is presented for the DVOPD and DES applications. This metric captures the energy efficiency of the different technology libraries. The 65 nm technology libraries have much higher energy efficiency. For example, for the DES application, using the LP-LVT libraries, a 2.19X improvement is obtained when compared to the 90 nm technology. Another interesting fact to note is that, for the DES application, the NoC supports a higher bandwidth per mW power consumption than for the DVOPD application. This is because of two reasons: firstly, the DVOPD application needs a higher operating frequency, which requires the

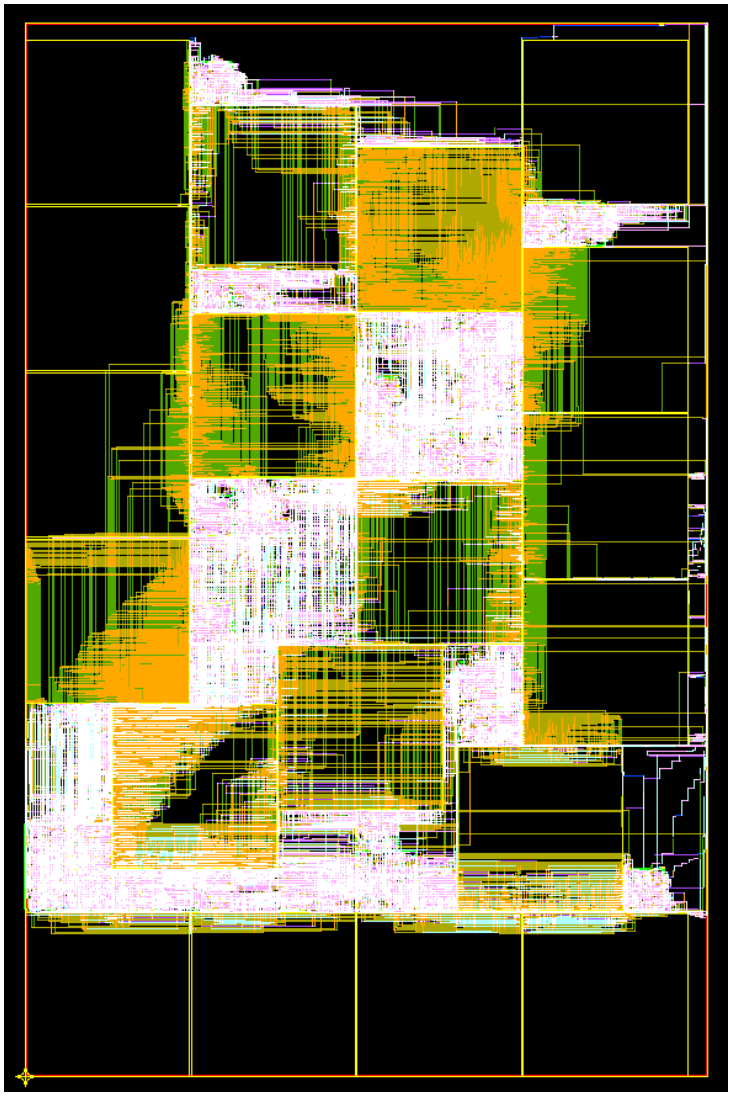


Figure 6.9: Layout of the DES mapping on 65 nm LP-HVT technology. Over-the-cell routing was allowed in this example.

synthesis tools to utilize more power intensive components for the switches. Secondly, the communication traffic is more evenly spread in the DVOPD application, thereby requiring more inter-switch traffic flows than the DES application.

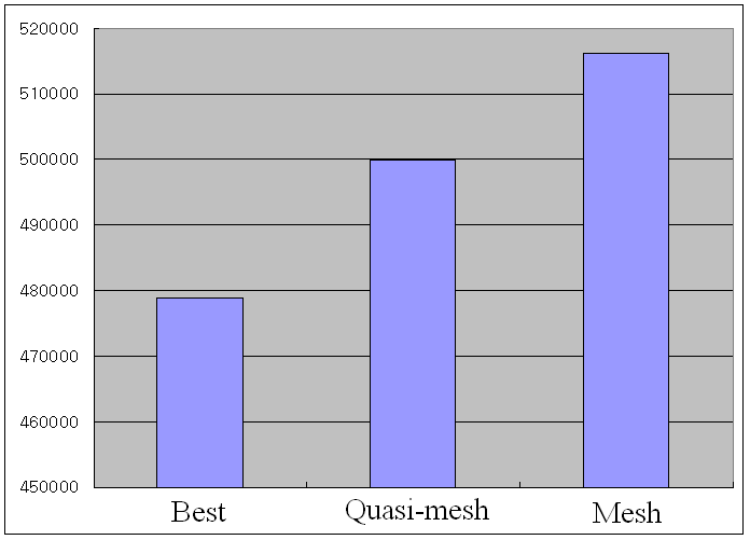


Figure 6.10: Comparisons of best topology synthesized by SunFloor *vs.* quasi-mesh and mesh topologies

Finally, the quality of the custom topology generated for the DES benchmark was compared with that of a mesh topology (19 switches, with each core connected to a switch) and a quasi-mesh topology (10 switches, with 2 cores connected to a single switch). In this case cycle-accurate simulations of the DES benchmark was performed with the designed NoCs using the ×pipes platform. The total application runtimes for the 3 designs are shown in Figure 6.10. As this figure indicates, the entire application performance (which also includes the time for computation) improves by 7% when the custom topology is used.

Bibliography

[1] T. Ahonen, et al. Topology optimization for application-specific networks-on-chip. In *Proc. SLIP)*, 2004.

[2] F. Angiolini, et al. Contrasting a NoC and a traditional interconnect fabric with layout awareness. In *Proc. DATE*, 2006.

[3] M.-N. K. Bambha, et al. Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors. *IEEE Trans. PDS*, 2005.

[4] N. Banerjee, et al. A power and performance model for network-on-chip architectures. In *Proc. DATE*, 2004.

[5] L. Benini, et al. Networks on chip: a new SoC paradigm. *IEEE Computer*, 2002.

[6] L. Benini, et al. *Networks on chips: Technology and Tools.* Morgan Kaufmann Publishers, 2006.

[7] D. Bertozzi, et al. xpipes: A network-on-chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine*, 2004.

[8] D. Bertozzi, et al. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. PDS*, 2005.

[9] J. Chan, et al. Nocgen: a template based reuse methodology for NoC architecture. In *Proc. ISVLSI*, 2004.

[10] M. Coppola, et al. OCCN: a network-on-chip modeling and simulation framework. In *Proc. DATE'04*, 2004.

[11] W. Dally, et al. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, 2003.

[12] M. Gasteier, et al. Bus-based communication synthesis on system level. *ACM TODAES*, 1999.

[13] W. Hang-Sheng, et al. A technology-aware and energy-oriented topology exploration for on-chip networks. In *Proc. DATE*, 2005.

[14] W. Hang-Sheng, et al. Orion: a power-performance simulator for interconnection networks. In *Proc. MICRO*, 2002.

[15] A. Hansson, et al. A unified approach to constrained mapping and routing on NoC architectures. In *Proc. CODES+ISSS*, 2005.

[16] W. H. Ho, et al. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *Proc. HPCA*, 2003.

[17] J. Hu, et al. System-level point-to-point communication synthesis using floorplanning information. In *Proc. ASP-DAC*, 2002.

[18] J. Hu, et al. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proc. DATE*, 2003.

[19] Y. Hu, et al. Communication latency aware low power NoC synthesis. In *Proc. DAC '06*, 2006.

[20] A. Jalabert, et al. xpipescompiler: A tool for instantiating application specific NoC. In *Proc. DATE*, 2004.

[21] A. Jantsch, et al. *Networks on chip*. Kluwer Academic Publishers, 2003.

[22] K. Lahiri, et al. Design space exploration for optimizing on-chip communication architecture. *IEEE T-CAD)*, 2004.

[23] S. Manolache, et al. Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC. In *Proc. DAC*, 2005.

[24] S. Murali, et al. Mapping and physical planning of NoC architectures with quality-of-service guarantees. In *Proc. ASP-DAC*, 2005.

[25] S. Murali, et al. A methodology for mapping multiple use-cases onto NoCs. In *Proc. DATE*, 2006.

[26] S. Murali, et al. An application-specific design methodology for stbus crossbar generation. In *Proc. DATE*, 2005.

[27] S. Murali, et al. Designing application-specific networks on chips with floorplan information. In *Proc. ICCAD*, 2006.

[28] OCP-IP. Open core protocol standard, 2003. http://www.ocpip.org/home.

[29] G. Palermo, et al. Pirate: A framework for power/performance exploration of network-on-chip architectures. In *Proc. PATMOS)*, 2004.

[30] S. Pasricha, et al. Fast exploration of bus-based on-chip communication architectures. In *Proc. CODES+ISSS*, 2004.

[31] C. S. Patel. Power constrained design of multiprocessor interconnection networks. In *Proc. ICCD*, 1997.

[32] A. Pinto, et al. Efficient synthesis of NoCs. In *Proc. ICCD*, 2003.

[33] D. Siguenza-Tortosa, et al. Vhdl-based simulation environment for proteo noc. In *Proc. HLDVT Workshop*, 2002.

[34] K. Srinivasan, et al. An automated technique for topology and route generation of application specific on-chip interconnection networks. In *Proc. ICCAD*, 2005.

[35] Synopsys. Astro. http://www.synopsys.com.

[36] Synopsys. Physical Compiler. http://www.synopsys.com.

[37] Synopsys. PrimePower. http://www.synopsys.com.

[38] Synopsys. PrimeTime. http://www.synopsys.com.

[39] T. T. Ye, et al. Analysis of power consumption on switch fabrics in network routers. In *Proc. DAC*, 2002.

[40] X. Zhu, et al. A hierarchical modeling framework for on-chip communication architectures. In *Proc. ICCAD*, 2002.

Chapter 7

Conclusions

This thesis sheds light on some questions which were previously unanswered or only partially answered. For first, the usefulness of NoC architectures and the benefits that can be obtained by their use inside a SoC were assessed taking into account the aspects related to the chip layout and the global wiring capacitance, since this factors are crucial for a meaningful and thorough analysis. In this aim, a comparative analysis of interconnect fabrics is presented, ranging from low-cost classic schemes to NoCs. While the traditional shared bus proves completely unable to cope with the heavy load injected by 30 IP cores, the evolutionary ML AMBA design and the ×pipes NoC handle the load with different degrees of effectiveness and area and power overhead. The Multi-Layer architecture is much more efficient in terms of area and power at the current technology node, but the NoC is faster and promises better scalability and predictability in future lithographic processes. NoC handles wiring delays and congestion much better than even a medium sized 5x5 crossbar component, so well so that most of its area and power overhead seems to be concentrated in buffers. This is contrary to expectations in some previous literature, where wire switching activity was expected to be dominant. While this ratio shifts in smaller technologies, using a 130 nm process, proper buffering optimizations are very useful. Clock gating, short FIFOs and narrow flit widths are instrumental in keeping area and power to reasonable levels. The comparison showed also how the NoC topology design, where a NoC is tailored to fit the target application, has noticeable potential benefits. The experiments related to the crossbenchmarking activity show more than 10% savings in global area, 8% savings in power, and constant performance. These savings are visible even despite being masked by large system-level overheads, such as the resources required by the clock tree and the cores themselves, and also by topological considerations - in a system with 30 IP cores and 30 NIs, the improvements that can be achieved

by optimizations of the network of switches alone have limited headroom. While it is not fair to compare a custom designed NoC to a fairly generic ML AMBA topology, the superior customizability of NoCs should be taken into account.

To take advantage of the mentioned customizability, a complete platform generation flow using NoC interconnects to generate fully functional chip layouts from initial high-level application models was developed. The integration of this flow represents the core of this thesis. It can be used to obtain a power and latency efficient design, by building a communication architecture that closely matches the application traffic characteristics, satisfying the different design constraints. Synthesizing such NoC architecture is non-trivial, given the large design space that needs to be explored. In this thesis, a complete methodology that automates the process, generating efficient NoCs that satisfy the design constraints of the application, is presented. To have fewer design re-spins and faster time-to-market, fast and accurate floorplan information is needed early in the design cycle. This leads to detecting timing violations on the NoC links during the NoC synthesis phase, thereby leading to timing closure with quicker convergence between the high level design and the physical design phases. Accurate switch and link power models that are based on layouts of the components and accurate link power estimates based on the wire-lengths obtained from floorplanning are used. Deadlock free routing methods have been integrated in the NoC synthesis process, to face a critical issue for proper NoC operation. Experiments on several SoC benchmarks show that the synthesized topologies are much better (an average of 2.78× power reduction, 1.59× hop-count reduction) than the best mesh topology and mesh-based custom topologies for significant case studies.

A methodology for the developement of the mentioned accurate models for the characterization of NoC switch area and power requirements is presented. The proposed approach is based on thorough parameterization on several architectural, deployment and runtime variables. This guarantees excellent applicability within a NoC CAD flow like the proposed ×pipes design flow, for topology mapping and/or design space exploration. The area and power models for the ×pipes case study turn out to be very accurate within the limits allowed by the non-idealities of synthesis tools, even when applied to a whole NoC topology with irregular traffic flows. Experiments show that, at least at the 0.13 μm node, applying the proposed methodology to netlist-level devices yields an acceptable approximation of the actual behaviour even after placement and routing, but that even greater precision can be achieved, if desired, by applying the same technique at the layout level. Another tradeoff among accuracy and modeling effort exists, namely,

coefficients can be extracted based on a single (or on just a few) device instances, by normalization of the synthesis report, or on several of them, by an interpolation process. With respect to the modeling methodologies, future work includes minimizing the characterization effort, and especially a detailed testing verifying how well the proposed technique scales to finer process technologies.

For NoCs to be feasible in today's SoC designs, a NoC architecture with low hardware overhead is required. Chapter 5 involves a novel method for reducing the NoC hardware overhead by automatically customizing the architecture of the switches of the NoC to match the designed topology and routing paths. The customization process is integrated with the ×pipes tool chain. The customization process leads to large reduction in network area and power consumption. Moreover, the critical paths of the switches reduce significantly, thereby leading to a significant speed-up of the NoC design. In future, the customization method will be extended for sizing the buffers of the NoC switches as well.

Finally, a complete and thorough study of the trends imposed by deep submicron manufacturing processes was performed in fully working 65 nm NoC designs. Experimental results show that, while new technology nodes allow for large benefits in terms of power consumption, device area and operating frequency, they also pose non-trivial challenges, which must be properly tackled by NoC design flows. The ×pipes design flow was thus updated to support the issues emerging with the technology shrinking, especially modifications were needed in the back-end phase, and architectural support (pipelined links) was also required for optimal results. A very positive outcome, however, is that the scalability of NoCs does not deteriorate even for large 65 nm designs, and that NoCs prove capable of tackling the challenges of 65 nm processes.

To understand exhaustively the NoC architectures, it is mandatory to investigate asynchronous or Globally Asynchronous Locally Synchronous (GALS) paradigms for their design. Asynchronous, or partially so, design styles may be the only way to counter extreme process variation issues in future technologies. Of course, asynchronous designs feature a very different set of tradeoffs compared to fully synchronous logic, including better power efficiency but much more complex implementation flows. Whether the advantages will outweigh the problems remains to be seen.

List of Figures

1.1 Comparison between gate delay and local/global interconnect delay:local wires and gate delay are scaling down while the relative contribution of the global wire delay is increasing with technology 19
1.2 A general view of a NoC 23
1.3 Examples of common network topologies 28
1.4 State of the art in NoCs overview 34
1.5 ×pipes building blocks: (a) switch, (b) NI, (c) link 38

2.1 The platforms under test. (a) shared bus AMBA AHB; (b) ML AMBA AHB; (c) ×pipes mesh; (d) ×pipes custom topology. **M**: ARM7 masters; **T**: traffic generators; **P**: privately accessed slaves; **S**: shared slaves 43
2.2 Task graphs for the two applications under test: (a) multimedia processing application, (b) LMS filtering application 46
2.3 The MPARM SystemC virtual platform 49
2.4 The synthesis flow for our test fabrics: ×pipes 50
2.5 The synthesis flow for our test fabrics: AMBA 52
2.6 Relative execution times for (a) **multi-high**, (b) **multi-low**, (c) **des** with varying cache sizes. The ML AMBA AHB execution time is the baseline at 100. AMBA AHB shared bus results lay beyond the upper limit of the Y axis scale 63
2.7 Latency of (a) single read, (b) burst read, (c) single write transfers on the interconnects. AMBA AHB shared bus results lay beyond the upper limit of the Y axis scale 64
2.8 Layouts of our test fabrics: (a) ML AMBA, (b) ×pipes meshes, (c) ×pipes custom 65
2.9 Split report for a ×pipes topology: (a-c) area of the 21-bit mesh, the 38-bit mesh and the 21-bit custom NoC; (d-f) power consumption of the three topologies 66

3.1 NoC Design Flow 72
3.2 Sunfloor NoC architecture synthesis (Second phase of the design flow) 73
3.3 (a) Filter application (b) Core graph with sustained rates and critical streams 75
3.4 Algorithm examples 78
3.5 (a), (b) Hand-designed topology and layout. M: ARM7 processors, T: traffic generators, P, S: private and shared slaves (c), (d) Automatically synthesized topology and layout. In Figure (c), bi-directional links are solid and uni-directional links are dotted. 82
3.6 Run time and latency for different cache sizes 86
3.7 VOPD custom topology floorplan and core graph 87
3.8 Performance comparisons 87

4.1 Outline of our characterization activity. The placement and routing step is optional both for the training and the test set . 95
4.2 Area requirements *vs.* target operating frequency 97
4.3 Dependency of the output buffer area on fw, bd 99
4.4 Example traffic in a 4x2 switch 101
4.5 Area and power coefficient modeling inaccuracy under different characterization policies: (a) area coefficients, (b) power coefficients. Models and test set are at the netlist level 109
4.6 Distribution of the area modeling inaccuracy over a subset of the design space for Methodology 2. Dark colour: underestimations; light colour: overestimations 110
4.7 Distribution of the power modeling inaccuracy for the switches of a 5x3 NoC mesh 110
4.8 Power coefficient modeling inaccuracy under different characterization policies. Models are at the netlist level, test set is at the layout level 111
4.9 Power coefficient modeling inaccuracy for Methodology 2. Models and test set are at the layout level 112

5.1 Switch architecture before and after the routing aware customization 121
5.2 Mesh and application-specific custom topologies for the MULT benchmark. The P0-P9 are the processors, T0-T4 are the hardware cores, M0-M9 are the private memories and S10-S14 are the shared devices. The shaded boxes connecting the cores are the switches in the design. 122

5.3 Switch area, power and input-to-output connection savings for the SoC designs 124

6.1 Our proposed complete NoC design flow for MPSoCs 130
6.2 The synthesis flow for ×pipes 133
6.3 Power consumption of 38-bit links of varying lengths at different operating frequencies. Values normalized to shortest link at slowest frequency. Missing columns represent infeasible length/frequency combinations. 137
6.4 Analysis of two representative ×pipes switches in different technology libraries. Figures normalized to the 4x4 switch in the LP-HVT library. 138
6.5 Three 4x4 ×pipes meshes. 139
6.6 Enhanced VOPD application, called DVOPD, with the capability to decode two streams in parallel. 140
6.7 Amount of pipelined links in two sample benchmarks. 143
6.8 DES benchmark. 144
6.9 Layout of the DES mapping on 65 nm LP-HVT technology. Over-the-cell routing was allowed in this example. 145
6.10 Comparisons of best topology synthesized by SunFloor *vs.* quasi-mesh and mesh topologies 146

List of Tables

2.1 Pre- and post-placement achievable frequencies 57
2.2 Power consumption of the fabrics 59
2.3 Energy consumption of the fabrics 59

3.1 Component Area-Power 76
3.2 Comparisons with standard topologies 88

4.1 Dependency on architectural parameters of the static power coefficient P_A 102
4.2 Dependency on architectural parameters of the dynamic power coefficients P_B, P_C 102
4.3 Dependency on architectural parameters of the dynamic power coefficient P_D 103
4.4 Dependency of power coefficients on architectural parameters . 103
4.5 Accuracy of the linear *vs.* parabolic models for the dependency of synthesis results on the target synthesis frequency. Coefficients derived with Methodology 2 113

5.1 Switch routing table example 120
5.2 Total switch area of the designs 123
5.3 Combinational area of the switches for the designs 123
5.4 Switch power consumption for the designs 124

6.1 Synthesis results on three 4x4 NoC meshes. Figures normalized to the 90 nm results. 139
6.2 High Bandwidth Application Results 141
6.3 Low Bandwidth Application Results. 143
6.4 Bandwidth supported per milliwatt of power consumption . . 144