×pipes Lite: A Synthesis Oriented Design Library For Networks on Chips

Stergios Stergiou[1], Federico Angiolini[2], Salvatore Carta[3], Luigi Raffo[3], Davide Bertozzi[2], and Giovanni De Micheli[1]

[1]Computer Systems Laboratory, Stanford University, Stanford, CA, 94305, USA

[2]Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, 40136 Bologna, Italy

[3]Dipartimento di Ingegneria Elettrica ed Elettronica, University of Cagliari, 09123 Cagliari, Italy

Abstract

The limited scalability of current bus topologies for Systems on Chips (SoCs) dictates the adoption of Networks on Chips (NoCs) as a scalable interconnection scheme. Current SoCs are highly heterogeneous in nature, denoting homogeneous, preconfigured NoCs as inefficient drop-in alternatives.

While highly parametric, fully synthesizeable (soft) NoC building blocks appear as a good match for heterogeneous MPSoC architectures, the impact of instantiation-time flexibility on performance, power and silicon cost has not been quantified yet. This work details ×pipes Lite, a design flow for automatic generation of heterogeneous NoCs. ×pipes Lite is based on highly customizable, high frequency and low latency NoC modules, that are fully synthesizeable. Synthesis results provide with modules that are directly comparable, if not better, than the current published state-of-the-art NoCs in terms of area, power, latency and target frequency of operation measurements.

1. Introduction

Increasing transistor density and system interconnect scalability limitations necessitate the introduction of *Networks On Chip* (NoCs) as a viable, scalable packet-switched micro-network interconnect scheme, alternative to bus architectures. Many NoC architectures have recently been proposed [4, 5, 6, 9]. A NoC is depicted in Figure 1.

Topology selection is an integral part of a NoC design flow. Regular (such as mesh or fat tree) topologies have been adopted for interconnecting homogeneous cores [16, 3, 2, 24] in the first generation of NoC approaches. For a heterogeneous system however, such as [21], a regular topology selection may lead to overdesigning and is therefore not favorable in most current NoCs [11]. For this reason, several researchers are currently focusing their efforts on developing complete application-specific NoC synthesis flows, where topology and network bulding blocks are customized at instantiation time [12, 6].

The performance advantage of application-specific NoCs has been convincingly demonstrated at the architectural level [12, 13, 14]. It is a well-known fact, however, that instantiation-time flexibility through automatic synthesis has a price in terms of silicon efficiency [22]. Hence, a key issue for NoC synthesis approaches is to demonstrate that the architectural headroom of synthesized NoCs with respect to less flexible, homogeneous NoCs is maintained throughout the logic and physical synthesis steps.

Our work addresses precisely this issue. It details a complete NoC design flow based on ×*pipes Lite*, a new, synthesis-oriented flow for the ×pipes library. This new version maintains most of the architectural advantages of the original proposed library [9], such as latency insensitive design, packet-switched communication with source routing and wormhole flow-control, and interface support for error control. However, ×*pipes Lite* was re-designed from scratch to be low-latency and synthesis-friendly. The contribution of this work is two-fold: first, it discusses architectural optimizations and specification tuning steps that have been developed to achieve high synthesis efficiency; second, it provides extensive post-synthesis implementation analysis to bound the design space that can be spanned by our flexible building blocks.

The outline of this work is as follows: Section 2 surveys related work. Section 3 analyzes the ×*pipes Lite* architecture and introduces the proposed design flow. Section 4 provides a comparison between the proposed and the original ×pipes architecture. Section 5 provides an insight on the issues addressed by utilizing SystemC as a synthesis front-end language. Section 6 details the synthesis experimental results. Section 7 concludes the paper.

2. Related Work

The most advanced SoC communication architectures used in industry today represent evolutionary solutions with respect to shared busses. For instance, the Sonics MicroNetwork [17] is a TDMA-based bus which can easily adapt to the data-word width, burst attributes, interrupt schemes and other critical parameters of the integrated cores, while providing very high bandwidth utilization. Another example is the STBus interconnect [25], a high-performance communi-

1530-1591/05 $20.00 © 2005 IEEE

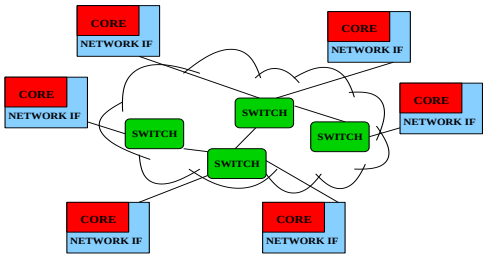

Figure 1. NoC Architecture block diagram

cation infrastructure that allows to instantiate shared busses as well as more advanced topologies such as partial or full crossbars.

Early works in [1, 15] pointed out the need for more scalable architectures for on-chip communication, and therefore to progressively replace shared busses with on-chip networks. Many NoC architectures have been proposed in the literature, but in most cases the design methodologies and tools are still in the early stage.

Most early NoC proposals are packet switched and exhibit regular structure. The NOSTRUM network [3] adopts a mesh based approach. The *Scalable Programmable Integrated Network (SPIN)* [2] is another regular, fat-tree-based network architecture. The *Linkoeping SoCBUS* [18] is a two-dimensional mesh network which uses a packet connected circuit to set up routes through the network. In [5] the use of the *octagon* communication topology for network processors is presented. Moreover, the implementation of a *star-connected* on-chip network supporting *plesiochronous communication* among system components is described in [7]. These NoCs provide scalable network fabrics for homogeneous system (e.g. symmetric chip-multiprocessors), but they do not allow arbitrary heterogeneous topology instantiation.

Significant steps in the direction of instantiation-time flexibility have been taken in the *Aethereal* NoC design framework, presented in [4], which aims at providing a complete infrastructure for developing heterogeneous NoCs with end-to-end quality of service guarantees. The network supports *guaranteed throughput (GT)* for real time applications and *best effort (BE)* traffic for timing unconstrained applications. *Aethereal*'s NI is highly configurable (it supports several session-layer standards, a variable number of ports, etc.), but the switch architecture is quite rigid. Furthermore, from the implementation viewpoint, both the NI and the switch make use of custom hard-macro FIFO buffers. These structures are not synthesizeable and they must be manually re-tuned when migrating to new technologies.

Support for heterogeneous architectures requires highly configurable network building blocks, customizable at instantiation time for a specific application domain. For instance, the *Proteo* NoC [6] consists of a small library of predefined, parameterized components that allow the implementation of a large range of different topologies, protocols and configurations. ×pipes NoCs [9] push this approach to the limit, by instantiating an application specific NoC from a library of synthesizeable soft macros (Network Interface - NI, switch and link). The components are highly parameterizable and provide reliable and latency insensitive operation. The main drawbacks with the first ×pipes architecture

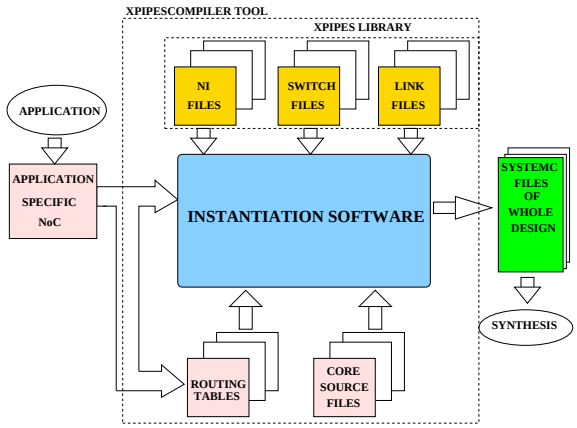

Figure 2. NoC synthesis flow

are the high complexity of its basic building blocks, and the lack of a complete synthesis path to silicon implementation (NoC cost is only estimated using high level models [23]). Both issues are fully addressed in this work.

3. ×pipes Lite Architecture and Synthesis Flow

×*pipes Lite* is a SystemC library of highly parameterizable, synthesizeable NoC Network Interface, switch and link modules, optimized for low-latency and high-frequency operation. Communication is packet switched, with source routing (based upon street-sign encoding) and wormhole flow control.

×pipes utilizes OCP 2.0 [20] as a means to interface with the SoC cores. Adhering to OCP's master/slave semantics, two NI interfaces are implemented for a peer-to-peer communication between two cores. Network communication is transparent to the application. Transferred packets are fragmented into units of fixed size, called flits.

3.1. NI Architecture

The ×pipes NI is designed as a bridge between an OCP interface and the NoC switching fabric. Its purposes are the synchronization between OCP and ×pipes timings, the packeting of OCP transactions into ×pipes flits and vice versa, the computation of routing information, and the buffering of flits to improve performance.

The ×pipes NI is designed to comply with version 2.0 of the OCP specifications. In addition to the core OCP signals, support includes for example the ability to perform both non-posted or posted writes (*i.e.* writes with or without response) and various types of burst transactions, including reads with single request and multiple responses. This allows for thorough exploration of bandwidth/latency tradeoffs in the design of a system.

To provide complete deployment flexibility, the NI is parameterizable in both the width of OCP fields and of ×pipes flits. Depending on the ratio between these parameters, a variable amount of flits is needed to encode an OCP transaction.

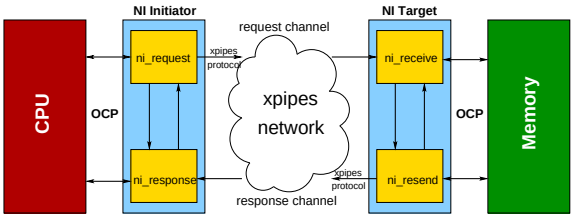

Figure 3. ×pipes Network Interfaces

For any given transaction, some fields (such as the OCP `MAddr` wires, specific control signals, routing information) can be transmitted just once; in contrast, other fields (such as the OCP `MData` or `SData` wires) need to be transmitted repeatedly, for example during a burst transaction. Thus, the NI is built around two registers; one holds the transaction header, while the second one holds the transaction payload. The first register samples OCP signals once per transaction, while the second is refreshed on every burst beat.

A set of flits encodes the header register; subsequently, multiple sets of flits are sent towards the fabric, each encoding a snapshot of the payload register subsequent to a new burst beat. Sets of payload flits are pushed out until transaction completion. Header and payload content is never allowed to mix in the same flit, thus simplifying the required logic. Routing information is attached to the header flit of a packet by checking the transaction address against a Look-Up Table (LUT).

As shown in Figure 3, two NIs are implemented in ×pipes, named "initiator" (attached to system masters) and "target" (attached to system slaves). A master-slave device will need two NIs, an initiator and a target, for operation. Each NI is additionally split in two submodules, one for the request and one for the response channel. These submodules are loosely coupled: whenever a transaction requiring a response is processed by the request channel, the response channel is notified; whenever the response is received, the request channel is unblocked. The mechanism is currently supporting only one outstanding non-posted transaction, but can be extended should any attached core need this feature.

The ×pipes interface of the NI is bidirectional; for example, the initiator NI has an output port for the request channel and one input port for the response channel (the target NI is dual). The output stage of the NI is identical to that of the ×pipes switches, for increased performance. The input stage is implemented as a simple dual-flit buffer with minimal area occupation, but still makes use of the same flow control used by the switches.

3.2. Switch Architecture

The ×pipes switch models the basic building block of the NoC switching fabric. It implements a 2-cycle-latency, output-queued router that supports fixed and round robin priority arbitration on the input lines, and a flow control protocol with ACK/nACK, Go-Back-N semantics.

Allocation of inputs towards specific output lines is handled at the *Allocator* module. Multiple Allocators exist in a switch, each driving one of its output ports (O_i).

Assuming that one input is currently owning access to O_i, it is maintained in its state until a tail flit arrives. Arbitration is subsequently performed upon receipt of a header flit

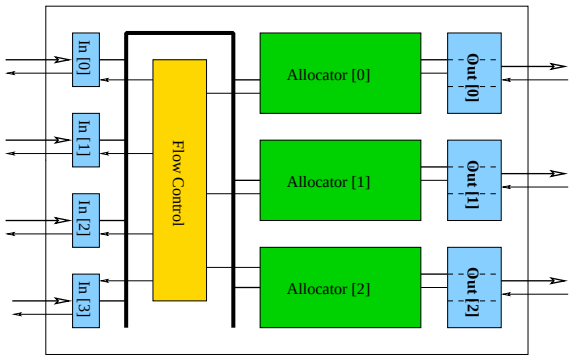

Figure 4. ×pipes 4x3 Switch

with routing information dictating that the incoming packet should exit through O_i.

An input flit can be rejected, and therefore nACKed, due to one or more of the following reasons:

- The output line is occupied by a previous transmitting packet.
- The buffering space for output O_i is already filled.
- Another header flit requesting the same output is concurrently appearing on another input port, and arbitration is won by the latter.

If a flit A is dropped, then all subsequent incoming flits are dropped as well, until flit A reappears at the input, adhering to a Go-Back-N flow control mechanism.

After a packet has won the arbitration, its header flit is properly adjusted in order to prepare for the next switch along the routed path to the slave. More specifically, routing information pertaining the current switch is rotated away; this allows positioning of the per-hop routing bits at a fixed offset within the flits.

The switch is parameterizable in the number of its inputs and outputs, its arbitration policy (fixed priority or round-robin,) as well as in the size of the buffering at the outputs. A 4x3 instantiation is depicted in Figure 4.

3.3. Link Architecture

A solution to overcome the interconnect-delay problem consists of pipelining links. The link data introduction rate is decoupled from link delay by trading it with latency. The NoC NI and Switch modules are designed in such a way that their behavior does not depend on the latency of the communication channels (latency insensitive operation) [19]. Pipelining has been used both for data and control lines.

3.4. Design Flow

The design flow is shown in Figure 2. The ×pipes compiler instantiates the desired NoC by means of configuration scripts that are either automatically generated in a previous step, or manually specified [12].

Configuration scripts detail general, core (NI), switch, and link specific parameters. General parameters comprise flit size, and core address space. Core (NI) parameters include type *(master/slave/both)*, address on the NoC and the switch connection endpoint.

Flit Width (bits)	Output Buffer Stages 4	16	64
Initiator NIs			
16	22.7	30.4	59.6
32	33.5	47.7	104.8
64	49.3	77.1	194.8
128	87.8	144.5	379.3
Target NIs			
16	27.5	34.8	62.6
32	36.9	51.1	106.9
64	55.4	82.0	194.3
128	93.3	151.1	382.5

Table 1. Cell power (mW) for initiator and target NIs.

Flit Width (bits)	Output Buffer Stages 4	16	64
Initiator NIs			
16	0.025	0.037	0.093
32	0.036	0.061	0.179
64	0.051	0.111	0.361
128	0.092	0.186	0.724
Target NIs			
16	0.035	0.053	0.110
32	0.045	0.078	0.193
64	0.063	0.114	0.352
128	0.117	0.230	0.701

Table 2. Area (mm^2) for initiator and target NIs.

Switch parameters include the number of inputs and outputs, along with the output buffering capacity. Finally, link parameters comprise the switch endpoints that they connect to, along with the number of their pipeline stages.

4. ×pipes Lite Design Revision

The original ×pipes design is essentially a full-featured NoC specification, addressing all aspects of a NoC design flow, including error control at the switch level, with parameterization pushed to the extreme. ×*pipes Lite* has instead been designed from the ground up to be a library of fully synthesizeable, low-latency, high-frequency NoC modules. In this context, architectural changes have been performed onto the original ×pipes library [9].

One key observation driving the development of ×*pipes Lite* is that NoCs typically provide abundant and scalable bandwidth, but they lack in providing low-latency services. In order to address this limitation, one of the most noticeable differences between ×*pipes Lite* and the first version of ×pipes is the dramatic reduction in pipeline depth, with no compromises on the frequency target. Switches, for example, now incur in a latency of just two cycles, against seven in the previous design. Network interfaces underwent a similar redesign, with four fewer pipeline stages in output channels and one less in input channels. This significant improvement was achieved by removing the logic related to virtual channel management and to error detection, even though the switch protocol still supports retransmissions. A side benefit of the shortening of ×pipes pipelines is greatly reduced buffering requirements, which drastically reduces synthesized area and increases achievable frequency.

Additional area savings, again compounded with latency improvements, were achieved by slightly constraining the ×pipes packet format. In contrast with the original specification, which aimed at maximum packing of information within the flits, a format with fixed field offsets and an immediate forwarding policy was chosen as more effective area- and latency-wise. Unnecessary bloat of synthesized hardware was also avoided by assuming routing information to fit in the first flit of a packet, by enforcing a fixed worst-case width, and by preventing header and payload content to mix within the same flit.

From the point of view of the core interface, OCP support was updated from version 1.0 to 2.0 of the specification, significantly improving the support for burst transactions in the process. Single request, multiple responses burst reads are now possible, and write commands can be either with or without response, trading speed for reliability.

5. Synthesis and Simulation Back-ends

While the original ×pipes library was mostly targeted to functional simulation, the structure and design choices of the ×*pipes Lite* NoC modules were heavily influenced by the target goals of high-speed, low-latency soft core synthesizeability. Still, simulation functionality was kept as a design objective, with the issue of instantiation-time parameterizability in the foreground. For this reason, we chose to keep a single codebase for the ×*pipes Lite* library, conditionally adding simulation- or synthesis-specific code when appropriate.

One of the main issues in keeping the simulation and synthesis flows consistent was the need for different configurations of individual instances of the same module. For example, two switches with different numbers of I/O ports or amounts of buffering may coexist in a ×*pipes Lite* design. While such configuration is easily achieved in a simulation environment, simply by building a generic SystemC class and by passing appropriate parameters to the constructor of every instance at runtime, the issue is more serious if the module needs to be synthesized, because synthesis tools require static code. Our approach was to use a single configurable class for simulation, and adding preprocessor directives when needed to automatically but statically differentiate every instance to be synthesized. For example, the single `class switch()` used for simulation would be renamed `class switch_4_4()` and `class switch_7_2()`, and then appropriately configured, when synthesizing a 4x4 and a 7x2 switch simultaneously.

For increased flexibility, switch I/O ports are instantiated differently during simulation. Code like `IN = new sc_in<sc_bv<FLIT_WIDTH> > [SWITCH_INPUTS]` is utilized, while synthesis resorts to static definitions and multiple class naming. In a dual manner, signal widths which are defined by means of templates in SystemC, can be accurately tuned for synthesis (e.g. `sc_bv<LOG_INPUTS> mux_sel`), and are set to upper limit values for simulation.

Various non synthesizeable syntactical alternatives were

Flit Width (bits)	Output Buffer Stages 4	16
4x4 Switches		
16	0.054 @ 1 GHz	0.126 @ 1 GHz
32	0.091 @ 1 GHz	0.214 @ 1 GHz
64	0.161 @ 1 GHz	0.378 @ 1 GHz
128	0.311 @ 1 GHz	0.777 @ 1 GHz
4x6 Switches		
16	0.091 @ 990 MHz	0.226 @ 950 MHz
32	0.151 @ 970 MHz	0.389 @ 950 MHz
64	0.287 @ 970 MHz	0.777 @ 915 MHz
128	0.518 @ 925 MHz	1.637 @ 900 MHz
6x4 Switches		
16	0.070 @ 980 MHz	0.167 @ 900 MHz
32	0.112 @ 950 MHz	0.268 @ 900 MHz
64	0.213 @ 935 MHz	0.533 @ 890 MHz
128	0.400 @ 875 MHz	1.028 @ 830 MHz

Table 3. Area (mm^2) and timing for 4x4, 4x6, 6x4 switches.

Flit Width (bits)	Output Buffer Stages 4	16
4x4 Switches		
16	31.1	62.7
32	55.5	116.9
64	104.4	219.3
128	199.1	430.5
4x6 Switches		
16	42.8	90.1
32	79.7	170.8
64	148.2	334.4
128	283.2	617.3
6x4 Switches		
16	35.9	69.1
32	62.9	123.2
64	117.2	237.6
128	223.4	453.5

Table 4. Cell power (mW) for 4x4, 4x6, 6x4 switches.

adopted for simulation, in order to improve performance. For example, missing synthesis support for range selection (`sc_uint<>.range(a, b)`) on non-constant `a, b` led to a code fork, where simulation used the above syntax, while synthesis resorted to a bit-by-bit assignment.

One more complex issue arose with synthesis of NI routing LUTs. Our choice was to integrate the LUT generation and the related code changes within the *×pipes compiler* tool. This tool, leveraging on the codebase hooks described above and on its own topology view, generates the customized code base for both simulation and synthesis along with the suitable Design Compiler synthesis scripts, as specified on the provided input script file.

6. Experimental Results

The feasibility of automatic generation of NoC topologies from parameterizable SystemC source code was thoroughly verified by performing extensive synthesis experiments in $0.13\mu m$ technology with Synopsys Design Compiler for all modules. Results were obtained for worst case commercial conditions under an average switching pattern for power.

Configuration parameters for the NIs comprise the flit width and the size of the output buffers. For a variety of NI configurations, total cell power and area results are depicted in Tables 1 and 2. A frequency of 1 GHz was achieved for all variations, and is therefore not reported in the tables.

Switches are also parameterizable in terms of flit width and size of the output buffers. In order to study the effect of the number of input and output ports, three switch instances were analyzed, namely 4x4, 4x6 and 6x4. The resulting area and frequency results are depicted in Table 3; a frequency of 1 GHz was achieved for all the 4x4 switch variations. Total cell power results are reported in Table 4.

An exploration of the area/frequency tradeoff was performed on 32-bit 5x5 switches. By varying the target synthesis clock, different area results were reported (see Figure 6). The maximum operating frequency achieved for the 5x5 switch module was 925 MHz. As expected, area requirements are almost constant until a certain frequency (800 MHz in this case), after which synthesis efforts to achieve higher operating frequencies result in a steep area increase.

An example illustrating the importance of the estimation of the area and clock frequency of a NoC can be shown by referring to Figure 6 and Figure 5. The diagram in Figure 5 features two alternative topologies for the same system, comprising four processors, four privately accessible memories, and three shared slave devices. Topology (a) leverages upon a single 11x11 switch in a crossbar-like fashion, while topology (b) creates three clusters with 5x5 switches. We integrated these ×pipes topologies within a complete cycle-true simulation environment ([8]), and ran an operating system-based benchmark on them, thus stressing the platforms with functional traffic. To complete a benchmark involving interprocessor communication, topology (a) required, as expected, less clock cycles, with about a 10% advantage due to lower access latency towards shared devices.

However, 11x11 ×pipes switches have a maximum working frequency of approximately 780 MHz, while 5x5 switches can reach 925 MHz (a 19% gain). Area-wise, one 11x11 switch requires 0.48 mm^2, while three 5x5 switches (synthesized for maximum frequency) take 0.51 mm^2 overall. The net result is that the designer can choose the topology with 5x5 switches at maximum frequency, achieving 10% overall faster performance than the "crossbar"

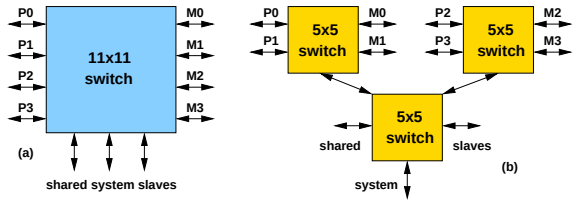

Figure 5. Test ×pipes topologies

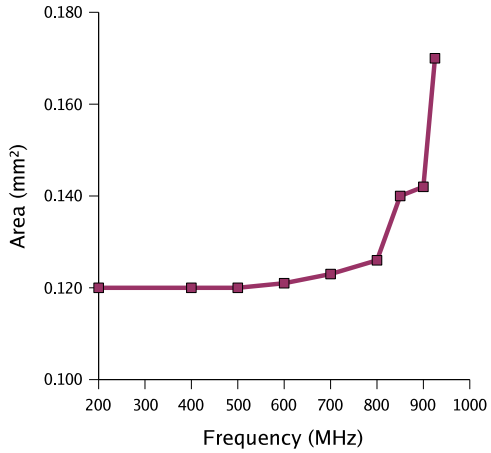

Figure 6. Area-frequency tradeoff

topology despite a comparable area footprint. Alternatively, the designer can clock the 5x5 switches about 10% higher than the 11x11 one (850 MHz), achieving roughly similar performance with a total area of 0.42 mm^2, a saving of 14%. A third alternative is synthesizing 5x5 switches with a frequency constraint of 800 MHz, about the same as the 11x11 one, thus achieving about 10% lower performance but with an overall area of just 0.38 mm^2 - a 34% saving over the 11x11 solution.

Finally, we performed place&route tests on a 5x5 ×pipes switch instance. The layout analysis shows that the area reported by the synthesis tools was an underestimate by less than 10%.

7. Conclusion

We have presented *xpipes Lite*, a design flow for the generation of synthesizeable and simulateable models for application-specific Networks on Chip. Synthesis results with commercial tools prove the value and competitiveness of automatic NoC generation. Whereas higher performance can be achieved by custom layout of NoCs, *xpipes Lite* benefits designers by reducing their effort and allowing them to explore the design space spanned by various NoC topologies and parameters.

8. Acknowledgement

This research is supported by MARCO Gigascale Systems Research Center (GSRC) and NSF, under contract CCR-0305718 and a grant by STMicroelectronics for DEIS.

References

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

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

[3] S.Kumar et al., ”A network on chip architecture and design methodology”, ISVLSI 2002, pp.105–112, Apr 2002.

[4] E.Rijpkema et al., ”Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip”,DATE 2003, pp. 350-355, Mar 2003.

[5] F.Karim et al., ”On-chip communication architecture for OC-768 network processors”, Design Automation Conference, pp. 678-678, June 2001.

[6] I.Saastamoinen, D.Siguenza-Tortosa, J. Nurmi, “Interconnect IP node for future system-on-chip designs”, Proc. of The First IEEE International Workshop on Electronic Design, Test and Applications, pp. 116-120, Jan. 2002.

[7] S.J.Lee et al.,“ An 800MHz Star-Connected On-Chip Network for Application to Systems on a Chip”, Digest of Technical papers, ISSCC 2003, pp.468-469, Feb. 2003.

[8] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, “Analyzing on-chip communication in a MPSoC environment”, Proc. DATE 2004.

[9] M. Dall’Osso et. al, “×`pipes`: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs”, pp. 536-539, ICCD 2003.

[10] A.Jalabert et. al, “×`pipesCompiler`: A Tool For Instantiating Application Specific Networks on Chips”, Proc. DATE 2004.

[11] S.Murali, G.De.Micheli, “Bandwidth Constrained Mapping of Cores onto NoC Architectures”, Proc. DATE 2004.

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

[13] A. Pinto , L. Carloni , A. L. Sangiovanni-Vincentelli, ”Efficient Synthesis of Networks On Chip”, *International Conference on Computer Design*, 2003.

[14] W. H. Ho , T. Pinkston, ”‘A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns”, *International Symposium on High-Performance Computer Architecture*, 2003.

[15] W.J.Dally, B.Towles, ”Route Packets, not Wires: On-Chip Interconnection Networks”, Design and Automation Conference DAC 2001, pp. 684-689, Jun 2001.

[16] W.J.Dally, S.Lacy, ”VLSI Architecture: Past, Present and Future”, Conf. Adv. Research in VLSI, pp. 232-241, 1999.

[17] D.Wingard,”MicroNetwork-Based Integration for SoCs”, Design Automation Conference DAC 2001, pp. 673-677, Jun 2001.

[18] D.Wiklund, D.Liu, ”SoCBUS: switched network on chip for hard real time embedded systems”, Proc. International Parallel and Distributed Processing Symposium 2003, pp. 78-85, 2003.

[19] L.P.Carloni, K.L.McMillan, A.L.Sangiovanni Vincentelli, ”Theory of Latency-Insensitive Design”, IEEE Trans. on CAD of ICs and Systems, Vol.20, No.9, pp. 1059-1076, Sep 2001.

[20] Open Core Protocol, “http://www.ocpip.org/”

[21] H.Yamauchi et al., “A 0.8 W HDTV video processor with simultaneous decoding of two MPEG2 MP@HL streams and capable of 30 frames/s reverse playback”, ISSCC, Vol.1, pp. 473-474, Feb. 2002

[22] D. G. Chinnery, and K. Keutzer, *Closing the Gap Between ASIC & Custom: Tools and Techniques for High-Performance ASIC Design*, Kluwer 2002.

[23] H.-S. Wang, X. Zhu, L.-S. Peh, S. Malik, ”Orion: A Power-Performance Simulator for Interconnection Networks.”, *International Symposium on Microarchitecture*, 2002.

[24] J. Liang, S. Swaminathan, R. Tessier, ”aSOC: A Scalable, Single-Chip communications Architecture”’, *International Conference on Parallel Architectures and Compilation Techniques*, 2000.

[25] G. Strano, S. Tiralongo, C. Pistritto, ”OCP STBUS Plug-in Methodology” *The International Embedded Solutions Event*, 2004.