

A case study for NoC based homogeneous MPSoC architectures

Original

A case study for NoC based homogeneous MPSoC architectures / Tota, Sergio Vincenzo; Casu, MARIO ROBERTO; RUO ROCH, Massimo; Macchiarulo, Luca; Zamboni, Maurizio. - In: IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. - ISSN 1063-8210. - STAMPA. - 17:3(2009), pp. 384-388.
[10.1109/TVLSI.2008.2011239]

Availability:

This version is available at: 11583/1793711 since:

Publisher:

IEEE

Published

DOI:10.1109/TVLSI.2008.2011239

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

Special Section Briefs

A Case Study for NoC-Based Homogeneous MPSoC Architectures

Sergio V. Tota, Mario R. Casu, Massimo Ruoch, Luca Macchiarulo, and Maurizio Zamboni

Abstract—The many-core design paradigm requires flexible and modular hardware and software components to provide the required scalability to next-generation on-chip multiprocessor architectures. A multidisciplinary approach is necessary to consider all the interactions between the different components of the design. In this paper, a complete design methodology that tackles at once the aspects of system level modeling, hardware architecture, and programming model has been successfully used for the implementation of a multiprocessor network-on-chip (NoC)-based system, the NoCRay graphic accelerator. The design, based on 16 processors, after prototyping with field-programmable gate array (FPGA), has been laid out in 90-nm technology. Post-layout results show very low power, area, as well as 500 MHz of clock frequency. Results show that an array of small and simple processors outperform a single high-end general purpose processor.

Index Terms—Multiprocessor systems-on-chip (MP-SoC), network-on-chip (NoC).

I. INTRODUCTION

THE unrelenting trend toward higher computation performance had led so far to an increase of the complexity and the number of the functional units of single monolithic microprocessors. Recently, this trend has started to slowdown even if the number of transistors is expected to continue to double every three years [1]. Power-thermal issues as well as design complexity have begun to limit the performance growth-rate compared with the increasing number of transistors available in a single die [2]. One way to cope with this productivity gap is the “tile-design” concept which underlies a simple yet effective paradigm: parallelization through replication of many identical blocks placed each in a tile of a regular array fabric. Instead of focusing on improving the complexity of a single block, the solution aims at delivering performance through several replicas of the same basic blocks. This approach has the major positive consequence of making systems design a matter of instantiation capability instead of architecture complexity, an objective which has to be pursued through innovative scalable hardware/software solutions. The resulting architecture can be certainly seen as an on-chip multiprocessor system. Therefore, we will refer to such system as a “homogeneous” multiprocessor systems-on-chip (MP-SoC), although the recent literature seems to reserve the MP-SoC acronym to the case of “heterogeneous” processors. MP-SoC design is a multidisciplinary research activity that encompasses on-chip communication infrastructures, microprocessor architectures, programming models, codesign/cosimulation flows and flexible methodologies for system level modeling and exploration.

Manuscript received December 10, 2007; revised April 04, 2008. First published February 06, 2009; current version published February 19, 2009.

S. V. Tota, M. R. Casu, M. R. Roch, and M. Zamboni are with the Dipartimento di Elettronica, Politecnico di Torino, I-10129 Torino, Italy (e-mail: sergio.tota@polito.it; mario.casu@polito.it; massimo.ruoch@polito.it; maurizio.zamboni@polito.it).

L. Macchiarulo is with the Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822 USA (e-mail: lucam@hawaii.edu).

Digital Object Identifier 10.1109/TVLSI.2008.2011239

Network-on-chip (NoC) is seen as the interconnection methodology for such systems [3]. The motivations for this choice are the better scalability of performance with the increasing number of processing elements (PEs) compared to standard on-chip busses, the high regularity which improves layout and particularly the allocation of wiring resources and the layered design approach which enables tackling the SoC complexity.

The computation is usually performed by a microprocessor. Even if it is always possible to implement hard-wired PEs, a programmable architecture gives the required flexibility to adapt and reuse the system in different scenarios. The possibility of reusing the same chipset in different devices is the only solution to face the growing costs of R&D as well as of mask-sets [4]. The performance of latest application-specific integrated processors (ASIPs) together with the high availability of transistors is making the design of custom hard-wired logic always less convenient (time-to-market, respin risks). Furthermore, current ASIPs offer a high level of configurability. It is now possible to optimize the code execution adding hardware support for frequent/computation-intensive operations.

Mastering the complexity of an MP-SoC requires new approaches to replace the standard design flow. Nowadays, the register transfer level (RTL)-to-netlist methodology has reached its maximum of efficiency and must be substituted with proper electronic system level (ESL) methodologies [5].

Our research activity focused on the intersection among the various aspects of this new paradigm: the integration of a scalable NoC communication infrastructure, a configurable microprocessor design, an appropriate distributed programming model and a methodology for system level modeling and exploration. All these aspects have been taken into account and we show their effective integration by means of a significant case of study. The most recent work on NoC design and implementation is in [6] which discusses physical design aspects in terms of automated floorplan, timing, and power issues. However, system-level modeling aspects were not discussed nor the field-programmable gate array (FPGA) prototyping of a real-life application as it is done in this work.

Section II discusses the ESL methodology used for system level analysis and exploration. Its goal was to provide the required abstraction in order to obtain the necessary visibility of different blocks interactions. It was then possible to analyze different candidate architectures and communication schemes, each of them with different tradeoffs in terms of power, performance, and cost. In Section III, we motivate and discuss the design choices concerning the NoC topology and routing, the switch and its interface to the processing element, as well as the software abstraction of the network based on a set of lightweight application programming interfaces (APIs) compliant with a subset of the message passing interface (MPI) protocol more suited for an embedded environment, the *embedded MPI* (eMPI) APIs. Section IV presents a case-study, the NoCRay graphic accelerator: a parallel graphics ray tracer rendering engine mapped first on FPGA for prototyping and then implemented on a 90-nm standard-cell ASIC technology. Conclusions are finally drawn in Section V.

II. SYSTEM LEVEL MODELING

The IP-XACT [7] standard has been used for system level modeling of the MP-SoC NoC-based environment and for automatic RTL generation of the target architecture. This standard is tool-independent thus

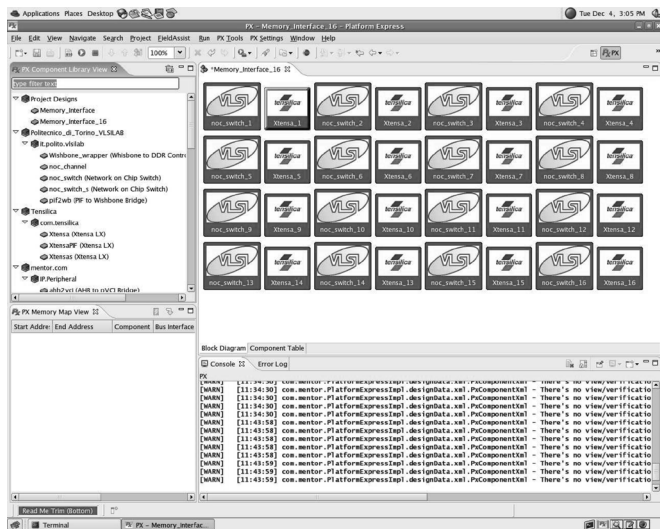


Fig. 1. MP-SoC NoC-based system level modeling with the IP-XACT ESL methodology.

any IP-XACT compliant library can be easily integrated in a custom design flow. To the best of our knowledge, there are no other works publicly available about the usage of the IP-XACT standard for NoC ESL modeling. Two basic IPs have been described using the IP-XACT standard, the switch, discussed in Section III-B and the Tensilica LX configurable processor [8]. These models have been then integrated in the Mentor Graphics *Platform Express* framework [9].

The switch model configurator allows to set a number of options like the routing algorithm (Wormhole or Deflection Routing), the network size, and topology (Mesh or Folded Torus) and the bit width of the connections. On the basis of the configurator constraints, a VHDL RTL model is generated accordingly. In the IP-XACT description of the switch we also included the definitions of the inputs/outputs (I/Os), which are called *connectors*. Only connectors of the same type can be linked together. The switch has two different types of connectors: the switch-to-PE and the switch-to-switch connector. In this way, during system level design, no errors are possible since only compatible interfaces can be linked together.

We used the high degree of configurability of the Tensilica LX processor to add a custom point-to-point interface in order to interconnect it to the switch with the result of embedding the Network Interface within the processor. In this way, we could use the same connector definition of the switch for the IP-XACT description of the processor. We included the instruction set simulator (ISS) model, automatically generated by the Tensilica Xplorer processor configuration environment [8], in the IP-XACT definition of the processor making it possible to speedup the simulation time compared to a RTL HDL model.

Once a given system configuration has been selected, it is possible to perform a co-simulation both for system exploration and software development. The generator automatically creates the top-level description of the system and the models of each IP. In our case as previously described, we used an ISS model for processors instances and RTL VHDL models for switches instances. The generator also automatically generates scripts for the co-simulation environment, the Mentor Graphics *Seamless* framework [9]. With this approach, it is possible to quickly build different configurations, test them, and then start the software development in parallel with the physical implementation.

Fig. 1 is a screenshot of the IP-XACT ESL environment used for system-level exploration. In the top-left window a list of available IPs is shown including our NoC models. Dragging and dropping a selected model into the graphic window automatically generates the given instance and interconnects the IP I/O with compatible interfaces of other

IPs. The graphic window on top-right shows a configuration with 16 Tensilica processors and 16 NoC switch instances. Once the system has been properly built it is possible to generate different views like the RTL or co-simulation one (bottom-right window).

III. NOC AND PROCESSING ELEMENT DESIGN

A NoC-based design consists of a number of different components like the network topology, the routing algorithm, the network-interface and the programming models. Even if it is possible to generate different topologies and routing algorithms during the system-level modeling step described in Section II, a specific configuration has been chosen for the case-study consisting in the use of the *folded-torus* topology and the *deflection-routing* algorithm. The motivation of this choice was chiefly the intention to keep area overhead of the network on the system as small as possible, possibly at a cost of some performance degradation. Since the use of deflection routing may imply out-of-order arrival of the elementary constituents of a packet, the flits, a network-interface, with reordering capabilities, has been embedded as part of the processor hardware, which is made possible by its ISA configurability.

Custom NoC instructions facilitated the implementation of the support for a subset of the message passing interface (MPI) programming model. It was possible to port current parallel MPI applications into the on-chip environment as shown in Section IV.

A. Topologies and Routing

Intrinsically 2-D networks such as meshes and tori better suit a silicon implementation. Wire lengths can be more accurately controlled thus providing better scalability properties of the mesh-like networks in terms of electrical performance, as detailed in [10]. A mesh topology can be implemented in a straightforward way by connecting all the switches to their physical neighbors, thus guaranteeing the minimal interconnection length. A more flexible configuration is that of the bidirectional torus, where the symmetry of the network is guaranteed for any node and any node-to-node connection, at the expense of higher wiring requirements. Furthermore, in this way, it is also possible to reduce the average packet latency. This topology has been used in this work.

Concerning the network routing strategies, the *Deflection Routing* algorithm has been adopted to implement the switch even if with our ESL flow it is possible to choose the *worm-hole* strategy as well. *Deflection Routing*, uses a full-blown packet-switching methodology by allowing different routing for every flit of the same packet. The basic idea of DF is that of choosing the presently “best” route for each incoming flit, without ever keeping more than one flit per input channel (thus the alternative name of “Hot Potato” routing). Deflection routing does not suffer from deadlock by construction (see [11] for a thorough analysis) while livelock may be an issue, as well as the case of non-minimal latencies due to the “deflection” of flits from their ideal path. However, in our previous works (see [12]), with a benchmark suite of parallel applications we observed sporadic cases of single flits delivered with high latency (larger than average) that did not significantly hamper execution times. In this case too we did not observe any overhead due to excess of latency.

The advantages of deflection-routing are that the switch area is greatly reduced, as its storage requirements are the theoretically minimal ones (as much memory as the incoming flits), no bottleneck is created by long packets as in wormhole routing, and no back pressure mechanism is necessary. The expense is that of introducing a potentially out-of-order reception of flits belonging to the same packets at the destination. These considerations have been taken into account during the implementation of our network interface as discussed in paragraph C.

B. Switch Design

The switch is perhaps the most critical element in NoC. On the one hand, it has to receive and deliver flits at very high rates in order to sustain network throughputs of tens to hundreds of gigabits per second, requiring clock rates of hundreds of megahertz. On the other hand, the switch logic complexity might be quite relevant, then possibly limiting the switch clock frequency. One of the designer's option is to break the logic depth by inserting pipeline stages, then raising the frequency to the desired level. However, the increased throughput comes at the cost of an additional switch internal latency that increases by the same amount of added pipe stages. Depending on the model of computation and the communication protocol implemented by the processing elements of the MPSoC environment, the latency increase has a direct impact on the overall performance degradation. Therefore, it should be kept at its minimum, even at the cost of some frequency reduction. Besides throughput and latency, two other factors are of great importance in the design of NoC switches. Silicon area and power consumption. It must be observed that for NoC topologies like meshes and torus, there is one switch for each processing element. It is then necessary to reduce switch area and power to a limited fraction of the processor ones. Typically, buffers consume the largest part of the switch area as it emerges from previous published works. The *Aethereal* framework [13] is one of the first works in this sense where a switch with guaranteed services (GEs) as well as best-effort services (BESs) has been implemented, using a time-division-multiplexing (TDM) and a Wormhole, respectively. The architecture is highly dominated by memory elements leading to an area of 0.24 mm² for the switch in a 130-nm CMOS technology.

A more recent work [14] proposes a six-port 57 GB/s Double-Pumped Nonblocking Router Core based again on the wormhole routing algorithm. While this implementation provides high-performance, the cost in terms of area is high as well requiring almost two million transistors per switch with an area of 12.2 mm² in a 150-nm CMOS technology. An implementation of a complete MP-SoC system using this switch has been recently proposed [15] showing that such a complex architecture does not let the system to scale due to thermal/power-consumption issues.

In this sense, deflection routing is better suited than wormhole where at least twice the amount of registers is needed, due to the need for buffers to implement flit-level flow control. At the same time, deflection routing has drawbacks like possible out-of-order flits delivery and limitation of the payload due to addressing information put into all flits. In this case, we privileged the smaller area of the switch and the absence of flit-level flow control. However, in case the application requires wormhole, the IP-XACT supporting tool allows to quickly redesign the system by instantiating the proper switch IP and the processing core with the proper network interface.

According to this rationale, a 5-ports (one devoted to the PE) deflection routing-based switch has been designed and physically implemented, with minimum latency (one clock cycle) and 32 bit wide flit size on a 90-nm CMOS standard-cell technology. All such parameters reflect the Tensilica processor ones which has a 32-bits register file and works at 500 MHz. We put much effort in meeting a target clock rate of 500 MHz without additional pipeline stages in the switch. We thus avoided clock-rate conversions and serialization/deserialization issues. Moreover, speeding-up the network with pipelined switches when the processors are not fast enough may lead to over design not justified for the majority of the applications, included the one developed in the case-study discussed further on, and will inevitably add latencies to the PE-to-PE communication. We obtained, after the layout, an area of 0.012 mm², a total power consumption of 3 mW (of which 45 μ W of leakage with a switching activity of 30% at 500 MHz). These results show that the overhead of the switch implementation, in terms of

area and power consumption, if compared with other typical blocks of a MP-SoC design, i.e., memories and processors, is very low.

C. Network Interface and NoC Programming Model

The high degree of configurability of the Tensilica LX processor was used to implement a high-speed direct link between each processor and the switch using the TIE ports. This I/O is directly connected to the processor register-file. When a packet of length L flits must be sent, the interface puts a sequence number into all flits. An address in the form $X-Y$ is put as well. In order to speedup the operation and to afford the maximum throughput of a single clock cycle for sending each flit, an additional counter for the sequence number and a lookup table (LUT) for addressing has been instantiated within the processor core and is supported by custom TIE instructions. The sequence number is cleverly used at the receiver to avoid any buffer for sorting out-of-order received flits. When a flit arrives, the PE: 1) reads a flit from the NoC storing it into a register and 2) uses the sequence number of the given flit as an offset address for the storage into the processor local data memory. Another register contains the base address. A double buffer technique has been implemented to support one clock cycle read operations. The size of the sequence-number field determines the size of the logic packet. The additional hardware consists of a small adder and two registers, which are seamlessly integrated by the Tensilica core development tool in the processor pipeline. In the NoCRay implementation 3 bits out of 32 are reserved for the sequence number, thus a maximum of 8 flits are possible within the logic packet. In this configuration the area overhead of the network interface is 0.014 mm² that is around 5% of the processor itself. In general the link parallelism can be customized to accommodate the flit size for a given implementation with a gate count overhead between 2–5 k gates for 32–64 bit wide flits. Interestingly, building a microprocessor system with the native processor bus (called PIF) instead of the NoC would require a bus controller with almost the same area, according to the Tensilica development tool.

The choice of embedding the interface with the processor allowed the ISA and consequently the compiler to support all NoC I/O thus facilitating the development of an *ad-hoc* scalable programming model. This is necessary because the scalability of the hardware infrastructure must be fully supported by the software layer. In this work, a scalable paradigm, natively parallel and architecture independent has been implemented: the MPI programming model [16]. Zhonghai *et al.* [17] analyze different programming models suitable for NoCs including the Message Passing but this implementation appears to be too complex and it seems more suitable for Operating Systems. The full MPI standard provides a wide range of communication primitives particularly suitable for computer networks; a more lightweight implementation might be preferable for an on-chip environment. We defined a subset of MPI primitives that we called *eMPI* with two basic primitives: *send()* and *receive()*. These functions are supported by the extended LX processor instruction set. When invoked, the *send()* automatically segments the variable size data into a number of fixed size flits, computes the header and injects the flits into the network. The *receive()* function reads an incoming flit and analyzes the sequence number to check for out-of-order data and performs reordering if needed.

D. Flow-Control and Synchronization

Although flow-control is not needed at flit-level because of the use of deflection routing, still it is needed at packet level as well as synchronization and these functions are supported by the basic *eMPI* primitives *send()* and *receive()*. We decided to implement them as blocking routines. Therefore, once a pair of processor open a communication, by exchanging a request-reply single flit packet, they will stop doing anything else but sending/receiving flits. This choice can be limiting in terms of performance (for instance, multiple outstanding transactions

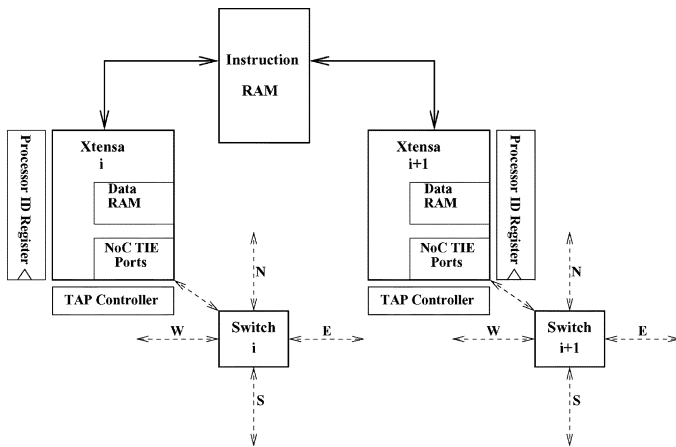


Fig. 2. Basic architecture of each processor couple. The *Instruction RAM* is shared between the two instances.

are not allowed) but on the one hand simplifies hardware and software design as well as testing. On the other hand it solves the problem of flow-control. Since flits are sent/received at the same rate—1 per clock cycle—there are no stalls to back-propagate.

Concerning synchronization, the *barrier()* eMPI blocking primitive is implemented as a semaphore. This function will release the control back to the application only when the number of synchronization flits received from other processors equals the expected one. This means that in a pool of processors, some will send synchronization flits and some will wait for one or more of them.

IV. CASE STUDY: THE NoCRay GRAPHIC ACCELERATOR

Graphic applications are one of the most suited for the MP-SoC NoC environment since they are highly parallelizable and they require both high computation capabilities as well as scalable communication infrastructures. In this section, the hardware/software (HW/SW) system components previously described have been then used to design a multiprocessor graphic accelerator that we called *NoCRay*. The system has been modeled and co-simulated using the virtual platform generated with our ESL flow. We were able, thanks to this development environment, to design in parallel the software layer and the hardware, targeting both an FPGA prototype and a 90-nm ASIC technology.

A. Parallel Raytracing Algorithm

The SPLASH-2 shared-memory parallel raytracing algorithm [18] has been used as a starting point. Basically, it renders a three-dimensional scene using a hierarchical uniform grid which is used to represent the scene, and early ray termination and anti-aliasing are implemented. A ray is traced through each pixel in the image plane, and reflects in unpredictable ways off the objects it strikes. Each contact generates multiple rays, and the recursion results in a ray tree per pixel. The image plane is partitioned among processors in contiguous blocks of pixel groups, and distributed task queues are used with task stealing.

This algorithm has been ported into the distributed-memory programming model using our eMPI APIs and fitted to the on-chip embedded environment. The code requires 32 kB of Instruction Ram and 32 kB of Data Ram (see Fig. 2). The instruction code is shared between two adjacent processors to reduce memory requirements.

Every line of the image is assigned to a specific processor belonging to the work-pool and each processor can be identified by the *ProcessorID Register* (see Fig. 2) thus the same firmware can be uploaded to all the processor instances using the Test Access Port (TAP) controller. The code was written in such a way to self-adapt to a generic number of

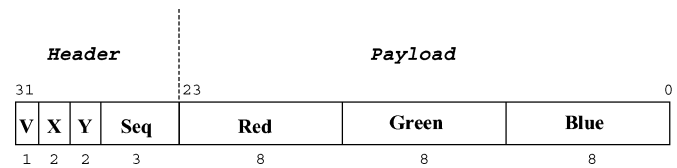


Fig. 3. Flit structure with 8-bit Header and 24-bit Payload (R-G-B).

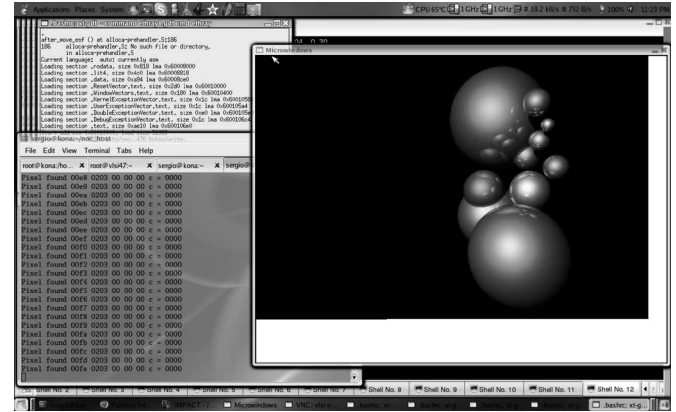


Fig. 4. Image generated by the NoCRay FPGA prototype.

processors, making the software as well scalable as the underlying hardware; with one basic instance of the processor and of the firmware, the design of a MP-SoC architecture becomes a matter of *cloning*. Moreover, the fact that adjacent processors share the same code, leaves room for further hardware optimization. In this case two processors share a single *Instruction Memory* with dual-port capabilities as shown in Fig. 2.

Each image line is segmented into blocks of 256 pixels. As soon as a processor ends line computation, sends it through the NoC to the *Master Processor* which has another TIE custom interface to an Ethernet controller. Computed pixels are sent through the ethernet interface to a host computer which displays the image into a screen. With this approach the architecture can compute images of any size without the need of any external memory. Since the computation is highly dominated by floating point operations, a hardware 16-bit multiplier has been added to the Tensilica LX base processor to improve performance. Its effect on the system-level performance could be immediately tested prior to implementation thanks to the ESL co-simulation flow.

In each NoC flit, 24 bits out of 32-bits are dedicated to the pixel payload (Red, Green, Blue, 8 bits each), while 8 bits are for the header (see Fig. 3). The latter is composed of three sub-fields: the valid-bit *V*, the destination address *X*, *Y*, and the packet sequence number *Seq*, that is used to help the reordering phase as described in Section III-C.

B. Hardware Implementation

The design has been first mapped on a Xilinx Virtex-4 LX-160 FPGA with 8 processors running at 33 MHz. It uses 90036 4-input LUTs and an equivalent gate count of 17.3M. The FPGA prototype successfully runs the application and the computed image was acquired and displayed shown by the host computer (see Fig. 4). With a host gdb debug server running an OCDemon, using a Wiggler JTAG interface, it was possible to upload the firmware and debug each processor separately.

We then implemented the NoCRay architecture on a 90-nm standard-cell technology. In Fig. 5, the manual floorplan is shown. The manual floorplan was required in order to obtain a symmetric design as well as to keep the switch-to-switch wire length as low as possible. The floorplan reflects the folded-torus physical layout mentioned in Section III-A.

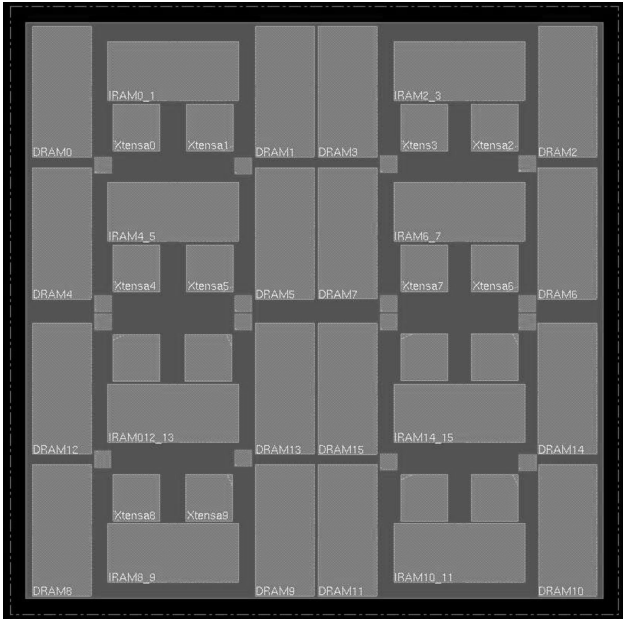


Fig. 5. Manual floorplan of the NoCray architecture.

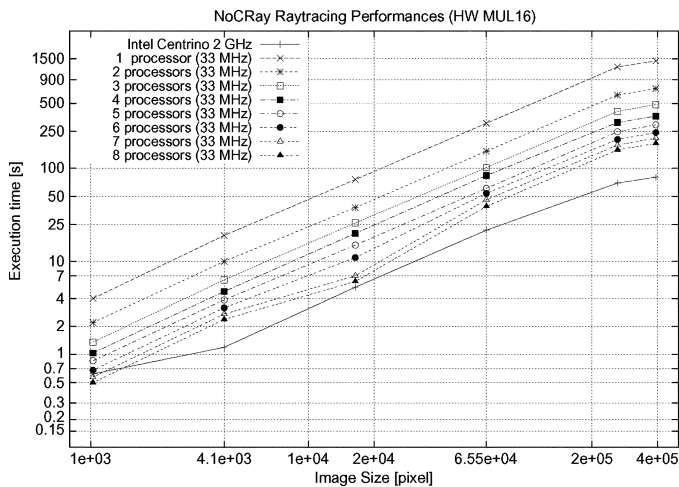


Fig. 6. NoCray multiprocessor architecture FPGA execution time.

C. Results

The final chip has an area of 34 mm² and a total power consumption of 600 mW, of which 50.6 mW of leakage while the maximum switch-to-switch distance is 2.5 mm. Processors give a total contribution of 3.24 mm² for the area and 480 mW for the power consumption while memories a total of 17.92 mm² and 74 mW. The contribution of the switches in terms of area is of 0.46 mm² and of 48 mW in terms of power consumption. Since an over-the-cell routing was not used, the total switch-to-switch routing area overhead is of 4.216 mm². Buffer insertion and timing-driven physical design has been done to reach the 500 MHz target frequency.

The FPGA system has been benchmarked varying both the number of active processors, between 1 and 8, and the image size, from 32 × 32 to 756 × 512.

Fig. 6 shows the execution time for each image-size/active-processors configuration of the FPGA prototype. The results have been compared with the execution time needed by an Intel Centrino running at 2 GHz. Considering that the FPGA implementation has a clock frequency of almost 2 orders of magnitude less than the Centrino processor, performance comparison clearly shows that, even if not taped-

out, the ASIC implementation of the proposed architecture, running at a still relative lower frequency (400–500 MHz) could anyway outperform the Centrino with a less complex design. The Centrino processor has been chosen just as a reference architecture for easy results comparison. Commercial GPUs will be used as a performance comparison in a future work.

V. CONCLUSION

In this paper, a modular and scalable methodology for MPSoC design has been proposed. Different aspects of the design flow have been tackled. A system level modeling approach for design exploration and co-simulation based on the latest ESL technologies, microprocessor characteristics for an efficient NoC link, a high-speed low-area deflection-routing switch implementation, an efficient Network Interface solution and a scalable programming model based on the MPI paradigm have been all proven in the design of a graphic parallel application, the NoCray MP-SoC Raytracer. The design has been first prototyped on FPGA and then implemented on a 90-nm standard-cell ASIC technology. Results show the feasibility of the proposed design flow achieving good results when compared with a general-purpose high-speed processor.

REFERENCES

- [1] "International Technology Roadmap for Semiconductors," [Online]. Available: <http://www.itrs.net>
- [2] J. Held, J. Bautista, and S. Koehl, "From a few cores to many: A Tera-scale computing research overview," in *Intel Development Forum*, San Francisco, CA, Sep. 2006.
- [3] A. Jerraya and W. Wolf, Eds., *Multiprocessor Systems-on-Chip*. San Francisco, CA: Elsevier Morgan Kaufmann, 2005.
- [4] C. M. Weber, C. N. Berglund, and P. Gabella, "Mask cost and profitability in photomask manufacturing: An empirical analysis," *Trans. Semicond. Manuf.*, pp. 465–474, Nov. 2006.
- [5] T. Kogel, R. Leupers, and H. Meyr, *Integrated System-Level Modeling of Network-on-Chip Enabled Multi-Processor Platforms*. Dordrecht, The Netherlands: Springer, 2006.
- [6] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, M. Srinivasan, L. Raffo, G. De Micheli, and L. Benini, "NoC design and implementation in 65 nm technology," in *Proc. 1st Int. Symp. Networks-on-Chip (NOCS)*, 2007, pp. 273–282.
- [7] The SPIRIT Consortium, Napa, CA, "The SPIRIT Consortium," 2008. [Online]. Available: <http://www.spiritconsortium.org>
- [8] Tensilica Inc., Santa Clara, CA, "Tensilica website," 2008. [Online]. Available: <http://www.tensilica.com>
- [9] Mentor Graphics Inc., Wilsonville, OR, "Mentor Graphics Inc. website," 2008. [Online]. Available: <http://www.mentor.com>
- [10] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, "Timing analysis of network on chip architectures for MP-SoC platforms," *Microelectronics J.*, vol. 36, no. 9, pp. 833–845, 2005.
- [11] M. Steenstrup, Ed., *Routing in Communication Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1995, pp. 263–305.
- [12] S. V. Tota, M. R. Casu, and L. Macchiarulo, "Implementation analysis of NoC: A MPSoC trace-driven approach," in *Proc. ACM Great Lakes Symp. VLSI*, Philadelphia, PA, Apr./May 2006, pp. 204–209.
- [13] K. Goossens, J. Dielissen, and A. Radulescu, "Aetherial network on chip: Concepts, architectures, and implementations," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 414–421, Sep.–Oct. 2005.
- [14] S. Vangal, N. Borkar, and A. Alvandpour, "A six-port 57 GB/s double-pumped nonblocking router core," in *Dig. Symp. VLSI Circuits*, Jun. 2005, pp. 268–269.
- [15] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS network-on-chip in 65 nm CMOS," in *Proc. ISSCC*, 2007, pp. 98–589.
- [16] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. Boston, MA: MIT Press.
- [17] Z. Lu and R. Haukilahti, *Networks on Chip*. Norwell, MA: Kluwer, 2003, pp. 239–260.
- [18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Symp. C. A.*, Santa Margherita Ligure, Jun. 1995, pp. 24–36.