A methodology and a case-study for Network-on-Chip based MP-SoC architectures

Sergio V. Tota, Mario R. Casu, Paolo Motto, Massimo Ruo Roch, Maurizio Zamboni
VLSI LAB
Electronic Department
Politecnico di Torino, Italy
{sergio.tota, mario.casu, massimo.ruoroch, maurizio.zamboni}@polito.it

ABSTRACT

The many-core design paradigm requires flexible and modular hardware and software components to provide the required scalability of 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 work a complete design methodology is proposed, tackling at once the aspects of hardware architecture, programming model and design automation. The proposed design flow has been used in the implementation of a multiprocessor Network-on-Chip based system, the NoCRay graphic accelerator.

The system uses 8 Tensilica LX processors and has been physically implemented on a Xilinx Virtex-4 LX-160 FPGA reporting a 17.3M equivalent gate-count. Performance are compared with a commercial general purpose processor and show good results considering the low frequency of the prototype.

Nano-Net 2007 September 24-26, 2007, Catania, Italy.
 ISBN 978-963-9799-10-3
DOI 10.4108/ICST.NANONET2007.2122

Categories and Subject Descriptors

C.1.2 [**Multiprocessors**]: Interconnection Architectures; D.1.3 [**Concurrent programming**]: Parallel programming

General Terms

Networks-on-Chip, Multiprocessor System-on-Chip

Keywords

NoC, MPSoC

1. INTRODUCTION

Network-on-Chip (NoC) is seen as the interconnection methodology for future homogeneous and heterogeneous Multiprocessor Systems-on-Chip (MP-SoC) [1]. 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.

MP-SoC design is a multi-disciplinary research activity that encompasses on-chip communication infrastructures, microprocessor architectures, programming models, co-design/co-simulation flows and EDA tools for automated system generation. So far the need for higher performance had led to an increase of the architectures complexity in a monolithic way of which the microprocessors evolution is the most evident example. From the various x86 architectures to the more recent super-scalar ones the trend was to implement always more complex solutions, driven by a need for an ever increasing system frequency and instructions per cycle. Recently, this trend has started to slow-down even if the number of transistors is expected to continue to double every two years. 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. The final aggregate-performance will only be limited by the number of transistors available in the same silicon. This approach has the terrific 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 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 bringing the concept of *application-domain* architectures. 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-set [3]. 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). All the components of the framework should be configurable and with modularity capabilities as required in a distributed environment.

Our research activity focused on the various aspects that this new paradigm coalesces. The integration of a scalable NoC communication infrastructure, a configurable microprocessor design, a distributed solid programming model and an automated design flow is demonstrated through a MP-SoC case study: a parallel graphics ray tracer rendering engine has been mapped on a FPGA prototype board.

2. BACKGROUND AND RELATED WORK

An MP-SoC design flow requires several hardware/software components and each of them plays an important part in the overall architecture. One of the first works on the design and implementation of a switch and the relative Network Interface (NI) for the NoC infrastructure is the *Aethereal* framework [4] [5]. In this work a switch with guaranteed services (GEs) as well as best-effort services (BESs) based on the time-division-multiplexing (TDM) approach is used. A worm-hole routing algorithm has been implemented for the switch while the NI offers interfaces for several SoC bus protocols (AXI, OCP, DTL). The architecture is highly dominated by memory elements leading to an area of $0.24mm^2$ for the switch and of $0.11mm^2$ for the NI in a 130nm CMOS technology. A more recent work [6] proposes a six-port 57GB/s Double-Pumped Nonblocking Router Core based again on the worm-hole 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 $12.2mm^2$ in a 150nm CMOS technology. A recent implementation of a complete MP-SoC system using this switch has been recently proposed [7] showing that such a complex architecture does not let the system to scale due to thermal/power-consumption issues. Zhonghai et al. [8] analyze different programming models suitable for NoCs including the Message Passing but this implementation appears to be too complex for light-weight applications and it seems more suitable for Operating Systems.

In this paper a *deflection-routing* switch has been used. The use of this routing algorithm for the NoC environment has been analyzed in [9]. Compared to the usual worm-hole approach, a deflection-routing based switch shows advantages in terms of area since memory elements are kept to the minimum. This approach gives also benefits in terms of power consumption. A 64-bit wide 5 ports switch has an area of $0.038mm^2$ and a power consumption of 29 μW/MHz in a 130nm CMOS technology [10].

With the deflection-routing approach out-of-order flits can reach the target PE thus a NI with reordering capabilities is required. Instead of adopting a bridge-like approach to interconnect the processor to the NoC translating a given bus protocol, the NI has been embedded inside the processor with a point-to-point link using the Tensilica Instruction Extensions (TIE) capabilities. This approach is more efficient in terms of area and more flexibility is added with the native support by the processor of NoC I/O primitives.

Concerning the programming model a set of light-weight Application Programming Interface (API) compliant with a subset of the Message Passing Interface (MPI) protocol has been implemented being more suited for an embedded environment, the embedded MPI (*e*MPI).

The generation of the whole architecture has been automatized with *RapidBuilder*, an in-house developed EDA tool for MP-SoC systems design. It is based on the well known *Eclipse* [12] framework which is natively thought to be modular and scalable. RapidBuilder has been integrated with the Tensilica Processor Generator, Xplorer, and automatically generates the RTL description of the NoC as well as all the scripts to perform cycle-accurate simulations using the Mentor Graphics *Seamless* co-simulation environment. In chapter IV the proposed methodology has been proven with a case-study. A multiprocessor implementation on FPGA of a Raytracer graphic application with 8 LX processors is proposed.

3. MP-SOC DESIGN FRAMEWORK

The design of on-chip multiprocessor architectures requires a completely new approach asking for a higher level of multi-disciplinarity if compared with the current flow. Today more than before the success of a platform depends on the smooth integration of different components. Any choice, if not integrated and verified in the whole environment, can lead to misleading results. In this section an integrated framework is proposed tackling at once all the hardware/software components.

3.1 Hardware Architecture

To make an architecture scalable and modular it is of key importance to appropriately partition all the different subsystems. Partitioning is the only way to catch all the contributions of the different blocks and the relationship between them. The first natural step is to distinguish the computation from the communication. Depending on the application domain these two components can significantly vary.

The computation is based on the Tensilica LX configurable processor [11]. The native support for multiprocessing, the complete configurability of the datapath and I/Os, the support for a strong co-design/co-simulation flow and the integration of the development environment in the *Eclipse* modular framework gave us the possibility to easily integrate it with our design flow.

Multiprocessor capabilities have been enabled with the use

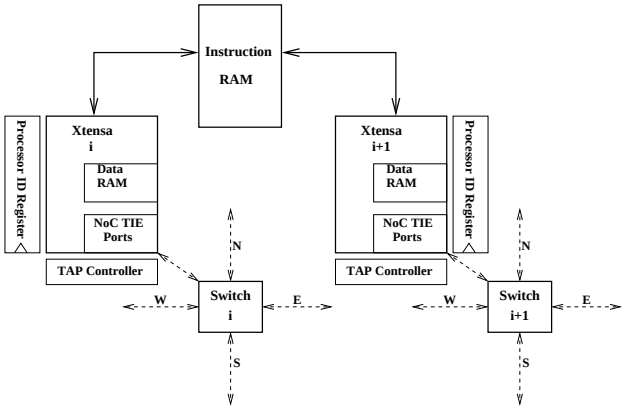

Figure 1: Basic architecture of each processor couple. The *Instruction RAM* is shared between the two instances.

of the *ProcessorId Register* (Fig. 1). The value of this register is used by the software layer to identify the processor where it is running on and behave consequently. This technique make the software self-adapting thus the whole architecture is scalable; with one basic instance of the processor and of the firmware, the design of a MP-SoC architecture becomes a matter of *cloning*, a task that can be easily automated.

The presence of a software locality between adjacent processors gives also the possibility to simplify the hardware infrastructure. For SIMD-like applications a single *Instruction Memory* with dual-port capabilities can be shared between two processors (Fig. 1). Strategies for efficient memory usage will be of key importance considering that next-generation multi-core architectures will be memory-dominated.

To implement an high-speed direct link between each processor and the NoC, the NI has been embedded in the processor using the TIE ports [13]. This I/O is directly connected to the processor register-file and it is modeled as a FIFO controller (Fig.2) thus giving an easy and efficient way to interface the switch to the processor. Even if in this implementation only one clock domain has been used, this approach gives the possibility of using different clock domains between the NoC and the processor. Using the processor internal register-file for flits storage, buffer elements in the switch are kept to the minimum and a double buffer technique has been implemented to support one clock cycle read/write operations with re-ordering capability for out-of-order received flits. Several are the reasons for taking the NI inside the processor. The use of a point-to-point link instead of a classic bus-bridge leads to a bandwidth improvement since no arbitration is required and flit-size of any parallelism is possible. The NoC I/O low-level primitives are directly supported by the tool-chain. To send some data through the NoC the software layer is only required to provide a pointer to the data to be sent and the target processor ID to the corrisponding function. The packetization engine automatically computes the header of the flit, adds the data in the payload field and send the flit through the network. The link parallelism can be customized to accommodate the flit size for a given implementation; the footprint overhead for integrating the NI in the processor core is between 2K-5K gates for 32-64 bit wide flits.

The communication between processors is based on a Network-on-Chip approach with a folded-torus topology and a switch based on the deflection-routing algorithm [14]. At the cost of a possible out-of-order delivery, this routing algorithm provides an area efficient implementation since it requires only one register for each input port. The flow control has been implemented with a simple yet efficient approach. At a given clock cycle the processor can inject a flit only if at least one of the four output ports (N,S,E,W) are not busy. This is possible because no virtual channels are implemented. Thus it becomes of key importance to appropriately dimension the NoC.

In the NoCRay case-study the deflection-routing switch has been used fulfilling all the bandwidth requirements while keeping the area overhead as low as possible. This switch has been implemented in a $130nm$ CMOS technology with an area of $0.039mm^2$ and a clock frequency of more than 500 MHz [10].

3.2 Programming Model

The scalability of the hardware infrastructure must be fully supported by the software layer, i.e. the programming model. The *shared-memory* paradigm is the most adopted and well understood by software architects. While this approach naturally fits in uniprocessor machines, it does not provide enough capabilities to support a distributed environment where memory is distributed as well.

A more scalable paradigm, natively parallel and architecture independent is the *Message Passing Interface* (MPI) programming model. The MPI standard defines both the syntax and semantics of a core of library routines providing a solid methodology for parallel programming. A heterogeneous environment can take full advantage of MPI using a common protocol for *Inter Process Communications* (IPC). Furthermore such applications can be analyzed with standard MPI profiling tools to extract computation/communication patterns. The full MPI standard provides a wide range of communication primitives particularly suitable for computer networks but for an on-chip environment a more light-weight implementation is preferred.

We defined a subset of MPI primitives oriented more on embedded application, the *embedded MPI* (*e*MPI) with two

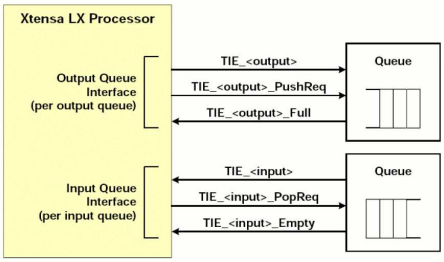

Figure 2: Xtensa TIE queues modeled as FIFO controller (*Tensilica LX Processor User Guide.*)

basic primitives: *send()* and *receive()*. With these two basic primitives the *barrier()* synchronization function can be also implemented. As discussed before, these functions are hardware supported by the LX processor instruction-set and are executed in one clock cycle each. 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. Since the low-level I/Os primitives are hardware implemented, these MPI libraries do not introduce any overhead compared to a custom approach. For this first implementation blocking primitives has been used and thus higher-level communications abstractions such as the many-to-many queues are not supported. It is under development a non-blocking implementation using interrupts in order to make the architecture more flexible.

The *e*MPI implementation thus can provide a compatibility layer for applications when running on different hardware and facilitate the refactoring of available MPI applications to the embedded on-chip environment.

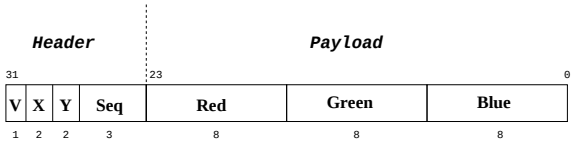

Figure 3: Flit structure with 8-bit Header and 24-bit Payload (R-G-B)

3.3 Design Automation

The complexity of next-generation on-chip multiprocessor architectures will require an higher level of design automation tools, capable of catching system level constraints to automatically generate the required computation/communication infrastructure. The same considerations on modularity discussed previously can be applied in the same way on the EDA tool framework. It must be flexible enough to let the integration of different modules with possibly no limitations on the extension/customization capability.

An ideal platform for this purpose is the *Eclipse* development framework. The high availability of open-source components and the easiness of integration of custom modules provides a industrial-quality environment where sophisticated tools can be tightly integrated. Our proposed design methodology relies on this framework with the integration of the in-house developed *RapidBuilder* MP-SoC generator module (Fig.4). It automatically generates the NoC infrastructure with RTL synthesizable models as well as co-simulation support through the Mentor Graphics *Seamless* environment. The network topology, the switch routing algorithm (wormhole/deflection-routing) and the number of PEs are all parameters that RapidBuilder uses for the generation of the given MP-SoC architecture.

4. CASE STUDY: THE NOCRAY GRAPHIC ACCELERATOR

Even if the Network-on-Chip paradigm is considered the communication infrastructure of the next-generation multicore architectures, there are almost no available physical implementations of full MP-SoC architectures as reference. For this reason a big effort has been done in this work to join together all the previous considerations implementing a real-life application running on the proposed hardware/software infrastructure. As a case study a graphic application, *NoCRay*, has been mapped in a MP-SoC NoC based environment and prototyped on a FPGA.

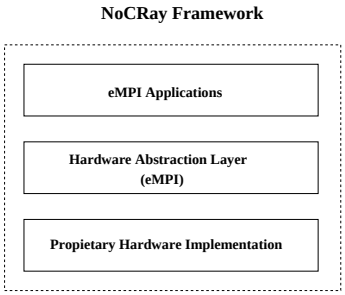

Figure 4: NoCRay Hw-Sw stacked layers.

The SPLASH-2 shared-memory parallel raytracing algorithm has been used as a starting point. For more details on this algorithm refers to this work [15]. This algorithm has been manually ported to the distributed-memory programming model using our *e*MPI APIs and fitted for the on-chip embedded environment. The code requires 64KB of Instruction Ram and 32KB of Data Ram (Fig. 1). 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. Each processor can be identified by the *ProcessorID Register* thus the same firmware can be uploaded to all the processor instances using the Test Access Port (TAP) controller. Depending on the value of the ProcID register the firmware computes the assigned lines. To avoid the use of external memory, each line is segmented into blocks of 256 pixels. As soon as a block is computed it is sent through the NoC to the *Master Processor*. This special processor has, in addition to the standard TIE-NoC interface, another TIE custom interface for a 10/100 Mbits OpenCores [16] Ethernet controller. Computed pixels are sent through the ethernet interface to a host computer which displays the image to 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 LX processor to improve performance. The flit is 32-bit wide with 24 bits for the payload (Red, Green, Blue) and 6 bits for the header (Fig. 3).

The system has been generated with RapidBuilder and verified with the Seamless co-simulation environment before the physical implementation. The design has been mapped on a Xilinx Virtex-4 LX-160 FPGA with 8 processors running at 33MHz. It uses 90,036 4-Input LUTs and a gate count of 17.3M is reported by the XIlinx ISE implementation tool.

5. RESULTS

The system has been benchmarked varying both the number of active processors, between 1 and 8, and the image size,

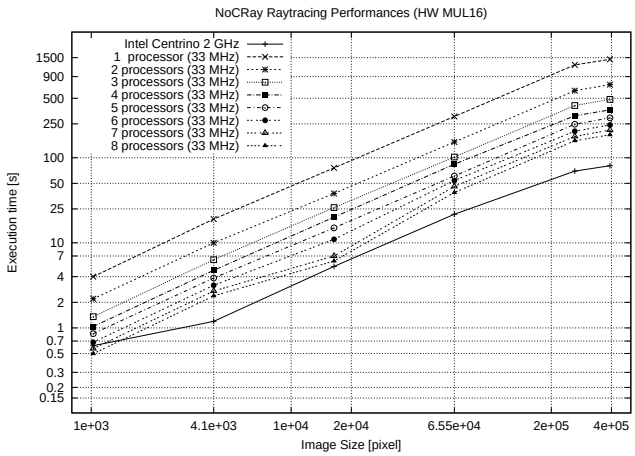

Figure 5: NoCRay multiprocessor architecture execution time with hardware MUL16 support.

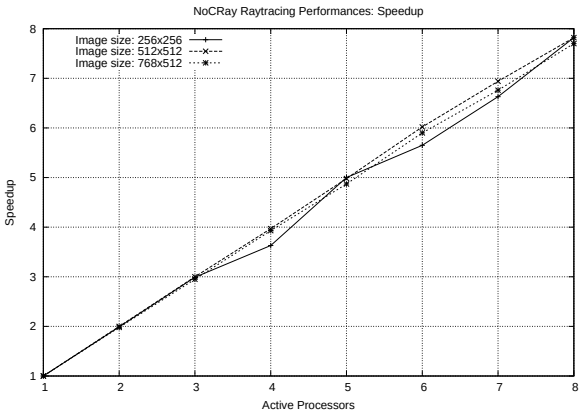

Figure 6: NoCRay multiprocessor architecture speedup.

from 32x32 to 756x512. Fig.5 shows the execution time for each image-size/active-processors configuration. The results have been compared with the execution time needed by an Intel Centrino running at 2GHz. Fig.6 shows the completely linear speedup of the performance varying the number of active processors. Considering that our prototype has a clock frequency of almost 2 order of magnitude less then the Centrino processor, performance comparison clearly shows that an ASIC implementation of the proposed architecture, running at a still relative lower frequency (200-300 MHz) could anyway outperform the Centrino with a less complex design.

6. CONCLUSIONS

In this paper a modular and scalable methodology for Multiprocessor System-on-Chip design has been proposed. Different aspects of the design flow have been tackled. Microprocessor characteristics for an efficient Network-on-Chip link, a high-speed low-area deflection-routing switch implementation, an efficient Network Interface solution, a scalable programming model based on the MPI paradigm and an EDA tools for automated architecture generation, have been all proven in the design of a graphic parallel application, the NoCRay MP-SoC Raytracer. Results show the feasibility of the proposed design flow achieving good results when compared with a general-purpose high-speed processor.

7. ACKNOWLEDGMENTS

We would like to thank Tensilica, Mentor Graphics, and Xilinx for the precious technology support.

8. REFERENCES

[1] Jerraya A. and Wolf W. (editors), "Multiprocessor Systems-on-Chip", Elsevier Morgan Kaufmann, San Francisco, California, 2005

[2] Held J., Bautista J, Koehl S., "From a Few Cores to Many: A Tera-scale Computing Research Overview", Intel Development Forum, September 2006

[3] Weber C. M., Berglund C.N., Gabella P, "Mask Cost and Profitability in Photomask Manufactoring: An Empirical Analysis", Transaction on Semiconductor Manufactoring, pp. 465-474, Nov. 2006

[4] K. Goossens *et al.*, "AEthereal network on chip: concepts, architectures, and implementations," IEEE Design & Test of Computers, Vol. 22, No. 5, Sept.-Oct. 2005, pp. 414-421

[5] Radulescu A. *et al.*, " An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration", in Proceedings of the Design, Automation and Test in Europe 2004

[6] S. Vangal, N. Y. Borkar, and A. Alvandpour, "A Six-port 57GB/s Double-Pumped Nonblocking Router Core", Dig. Symp. VLSI Circuits, pp.268-269, June 2005

[7] Sriram Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS", ISSCC 2007

[8] Zhonghai Lu, Raimo Haukilahti, "NOC Application programming interfaces: high level communication primitives and operating system services for power management", Kluwer Academic Publishing, "Networks on chip", pp. 239-260, 2003

[9] Zhonghai Lu et al., "Evaluation of On-chip Networks Using Deflection Routing", GLSVLSI, April 30-May 2, 2007 Philadelphia

[10] S. V. Tota, M. R. Casu, L. Macchiarulo, "Implementation Analysis of NoC: A MPSoC Trace-Driven Approach", pp. 204-209, in Proceedings of the 2006 ACM Great Likes Symposium on VLSI, Philadelphia, April 30-May 2

[11] Tensilica Inc., http://www.tensilica.com

[12] The Eclipse open development platform, http://www.eclipse.org

[13] "Tensilica Extensible Instruction User Manual"

[14] F. Borgonovo, "Deflection Routing," in [17] pp. 263-305.

[15] S. Cameron Woo *et al.*, "The SPLASH-2 Programs: Characterization and Methodological Considerations", In Proc. of the 22nd Symposium on C. A., p. 24-36, Santa Margherita Ligure, June 1995

[16] The OpenCores Project, http://www.opencores.org

[17] M. Steenstrup, ed., *Routing in Communication Networks*, Prentice Hall, 1995.