A multiprocessor based packet-switch: performance analysis of the communication infrastructure.

A Multiprocessor based packet-switch: performance analysis of the communication infrastructure

Sergio Tota, Mario R. Casu, Massimo Ruo Roch, Maurizio Zamboni
VLSI Lab
Dipartimento di Elettronica
Politecnico di Torino
{sergio.tota, mario.casu, massimo.ruoroch, maurizio.zamboni}@polito.it

***Abstract*— The intra-chip communication infrastructures are receiving always more attention since they are becoming a crucial part in the development of current SoCs. Due to the high availability of pre-characterized hard-IP, the complexity of the design is moving toward global interconnections which are introducing always more constraints at each technology node. Power consumption, timing closure, bandwidth requirements, time to market, are some of the factors that are leading to the proposal of new solutions for next generation multi-million SoCs. The need of high programmable systems and the high gate-count availability is moving always more attention on multiprocessors systems (MP-SoC) and so an adequate solution must be found for the communication infrastructure. One of the most promising technologies is the Network-On-Chip (NoC) architecture, which seems to better fit with the new demanding complexity of such systems. Before starting to develop new solutions, it is crucial to fully understand if and when current bus architectures introduce strong limitations in the development of high speed systems. This article describes a case study of a multiprocessor based ethernet packet-switch application with a shared-bus communication infrastructure. This system aims to depict all the bottlenecks which a shared-bus introduces under heavy load. What emerges from this analysis is that, as expected, a shared-bus is not scalable and it strongly limits whole system performances. These results strengthen the hypothesis that new communication architectures (like the NoC) must be found.**

I. Introduction

In the recent years always more attention has been dedicated to the development of new solutions for intra-chip communications. This is happening for a number of reasons. Starting from the 130 nm technology node, global wires are becoming a critical issue in the success of a design. Crosstalk analysis is a task that can not be neglected anymore and its computation is getting always more complex at each new node. Current EDA tools give the designer the capability of a complete design exploration, but this is a time consuming operation and problems always appear at a late stage of the design getting the timing closure a dangerous task. The probability of a re-spin of the design is very high, and with the increasing cost of a mask-set, this leads always more companies to go out of the business. Furthermore, clock speed is getting such high [1] that a signal could not be propagated across the chip in a single clock cycle anymore. *Latency Insensitive Protocols* (LIP) [2] have been recently proposed. They aim at de-correlating the functionality of the computational blocks from the clock speed of the communication infrastructure so that any IP, with the introduction of a wrapper called *shell*, will have the capability of working in the presence of signal propagation more than one clock cycle long.

For more complex systems, with an high bandwidth demand and the need for a certain level of reconfigurability, a completely new approach was proposed ([3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]). The idea is to build a number of micro-networks inside the chip and let each IP communicate with the others using a message-passing scheme. Packets will be the basic data unit that will be *routed* from the source to the destination. This Network-On-Chip (NoC) architecture could bring to a number of advantages. First, since interconnections can be dynamically configured, it would be possible to create programmable architectures which each user can customize according to his/her needs. A chip may then contain several IP blocks like processors, memories, FPGAs, DSPs, custom-logic etc, and the final user will decide how to *logically interconnect* them in the final design. This means that a single product can be used for a domain of applications by simply reconfiguring the communication traffic. This concept is similar to the *Sea of gates* one, in which gates are already placed in the chip, and only interconnections must be set. Another advantage of the NoC approach is its regular structure (e.g. a mesh topology). This could lead to early characterize all the physical parameters of the communication infrastructure with a great impact in design productivity.

As mentioned before, programmable elements like microprocessors seem the ideal companion for this infrastructure, and since the multiprocessor arena is recently becoming a hot topic, the development of a scalable, programmable, high-speed backbone for such systems would be the key for next-generation Multiprocessors System-On-Chip (MP-SoC). In order to show the real need for such new communication architectures, which are still in a development stage, it is important to fully understand the limits of current communication systems used in SoC, which are on the contrary well consolidated in the SoC design practice. Moreover, it is crucial to evaluate which are the real-life application domains which could benefit from a high speed communication infrastructure. This application-driven heuristical approach, we believe, though not so appealing from a theoretical point of view, is more adapted to evaluate the real need for a high-bandwidth system then traffic models borrowed from the computer networks domain.

0-7803-9333-3/05/$20.00 ©2005 IEEE

Currently, the most used communication typology for SoC systems is the *shared-bus* which has been implemented in several on-chip communication protocols like AXI [17], OCP [18], DTL [19], CoreConnect [27], STBus [20], etc. Since this scheme is so popular in SoCs, a correct knowledge of its characteristics, performances, fields of applications and limitations is essential as a preliminary analysis before developing new communication architectures like the NoC.

Communication requirements have been analyzed in several previous works. In [21], [22], [23] and [24], traditional parameterized statistic traffic generators were used, that, in spite of their generality, prevent designers from assessing performance in presence of real-life workloads and make it difficult to account for dynamic effects such as bus contentions. In [25] a simulation environment that models all hardware and software components of a multiprocessor system is proposed, but the applications that were mapped on such system do not represents a *real-life* case study. Our work is based on a 8-processor system mapped on a FPGA, in which a distributed ethernet packet-switch application was implemented, in order to provide realistic performance estimates and statistics. The aspect of packets managing inside the chip also represents a first outline of a future network-on-chip multiprocessor environment.

The article is structured as follows. Section II outlines the problem and makes some theoretical considerations. Section III describes the hardware and the software system used for the test. Section IV presents the results. In section V the results are commented, conclusions are drawn and a perspective of our future work is this field is given.

II. Point-to-point channels vs. Shared-buses

We can divide communication channels mainly into two opposite fields:

- Point-to-Point links
- Shared links

Each of them has advantages and drawbacks. Point-to-Point (P2P) links are fast, they do not require any arbitration protocol and permit full I/O concurrency (in the presence of other P2P links). But they require that each peripheral has as much I/O interfaces as the number of other peripherals to talk with. If there are $2N$ peripherals and each needs a link with each other, a total of $2 \times N \times N$ P2P unidirectional links are required, as it shown in Fig. 1.a. This situation can easily bring to wire routing congestion and so it is inapplicable when the number of blocks to be connected becomes significant.

On the other hand, *shared-buses* are easy to use, and each block, with just one I/O interface, can virtually communicate with every other block connected to the same shared- bus as shown in Fig. 1.b.

Wire routing of such infrastructure is also definitely easier. But in this case only one resource can drive the bus at a time and an *arbiter* is required to handle contentions. For this reason an handshake protocol is also required (Request-Grant scheme) and this introduces an extra overhead. Since the bus

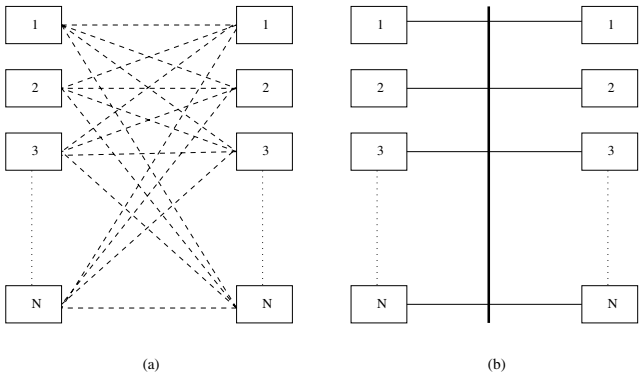


Fig. 1. *a)* Point-to-Point configuration . *b)* Shared-Bus configuration.

is now shared with other N peripherals, each one uses the bus N times less frequently than the P2P case, as long as a fair bus arbitration is performed. So, the right choice of a P2P bus versus a shared-bus depends on several aspects, primarily cost and performances, which strongly depend on the application at stake.

If W is the parallelism of the bus, f_{clock} the clock frequency, and $\alpha \leq 1$ is a corrective factor which takes into account the bus protocol overhead (e.g. arbitration, handshake for write/read cycles, et cetera), when all the resources want to access to the same shared-bus, the bandwidth B in bit/sec of each resource is:

$$B = \alpha(W \times f_{clock})/N \tag{1}$$

So for example, if we consider a read cycle, we would likely have a valid data every 4 clock cycles (Request-Grant-Address-Data) and this leads to a value of $\alpha = 1/4$ which strongly cuts the real bandwidth down.

A. System Architecture

Our aim is to provide realistic performance estimates and statistics on the communication infrastructure of SoCs based on shared bus. Traditional parameterized statistic traffic simulator do not offer an adequate solution for this purpose because it is difficult to account for dynamic effects such as bus contentions. Instead, we believe, a realistic test has to be performed on the field with realistic applications.

For these motivations we built a multiprocessor architecture to perform an ethernet packet-switch. All the ethernet controllers and the microprocessors were linked to the same bus.

When a packet arrives at the port i of the switch, the ethernet controller i generates an interrupt to the microprocessor i which reads the packet. At this point the destination address of each packet is analyzed to calculate the output port j and the packet is sent to the ethernet controller j.

To be as much flexible as possible, we made use of processors also to generate the traffic for the switch and to monitor the packets which were coming out from the switch. With this environment we had the possibility to test the performances

of a real-life application and at the same time to control and monitor each parameter of the whole system.

The architecture is organized as follows.

- *4x4 ports ethernet packet-switch.* It is composed of 4 processors and 4 ethernet controllers (10/100 Mb/s). Each resource is connected to the same shared-bus as it is shown in Fig. 2.
- *4 traffic generators.* Each traffic generator is composed of a processor and an ethernet controller. It also acts as a traffic monitor for the incoming packets (Fig. 2).
- *A timer.* It has been configured to generate an interrupt each second so that the number of transmitted and received packets per second for each port is easily computed.
- *An interrupt-controller for each processor.* A total of 8 interrupt-controllers are presents. Each controller receives interrupts from the timer and from the ethernet controller (a packet has arrived or a packet has been sent).

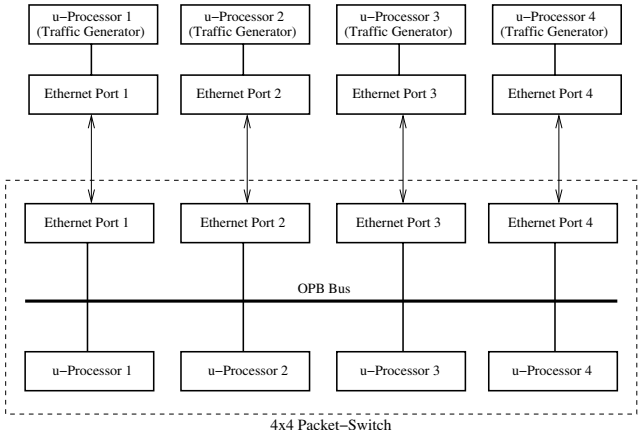


Fig. 2. System architecture

A first method to evaluate the system performances consists in sending a number of packets to each port and monitor the number of packets that exit from each port after one second of operation. If the total number of output packets is less than the total number of packets that were injected into the switch, apart from small differences due to the abrupt interruption when one second of count is reached, it means that a performance loss occurred somewhere.

The first cause of packets loss is the bandwidth of the bus that must be at least equal to the aggregate bandwidth of the ports. In our case the speed of each port was set to 25 Mb/s so with a maximum of four ports the aggregate bandwidth is 100 Mb/s which is far below the theoretical bandwidth of the bus, calculated according to equation (1).

Another possibility of loosing packets occurs when the *inter-frame gap* between two consecutive incoming packets is less than the time of the interrupt service routine to read the packet and make the input buffer ready to receive the next incoming packet. This situation could lead to a systematic loss of packets even without any bus contention.

Considering the ethernet channel when the traffic generator is continuously sending packets; we define T_{send} as the time in which the channel is busy transmitting a packet, and T_{idle} as the time where the channel is not used.

To fully use the bandwidth of the channel we should have $T_{idle} = 0$ but this is not possible because the ethernet protocol requires an *inter-frame gap* which depends on the ethernet speed. We have also to consider another aspect. If we call T_{Tx} the time to write a packet in the ethernet Tx buffer (which is equal to T_{Rx}, the time to read a packet from the ethernet Rx buffer), this time is composed of the time spent to run the *Interrupt Service Routine* and all the others software routines, let us call it $T_{Tx,Sw}$, and the time spent to write or read the entire packet through the bus, which we define as $T_{Tx,Hw}$.

So, after a packet is sent from the ethernet controller, before sending a new packet, some time is spent by the software routine to execute the firmware and some time is spent by the processor to write the entire packet into the ethernet buffer. This means that in first approximation:

$$T_{idle} = T_{Tx,Sw} + T_{Tx,Hw} \tag{2}$$

Lets now suppose to have a number N of data as 32 bits words to send through the channel. If we want to send this payload in M packets, the duration of each packet will be:

$$T_{send} \propto N/M \tag{3}$$

If we now define $M^{'}$ a number of packets so that $M^{'} < M$, we have that:

$$T^{'}_{send} \propto N/(M^{'}) \tag{4}$$

With $T^{'}_{send} > T_{send}$. Since $T_{idle} = T_{Tx,Hw} + T_{Tx,Sw}$, if we consider that the time spent to run the software is independent from the value of N and M, and that the time spent to write or to read a packet is considerably less in comparison with the software time which requires thousands more clock cycles to be executed, if we express the bandwidth usage B_u like:

$$B_u = (T_{send})/(T_{send} + T_{idle}) \tag{5}$$

we see that if the packet length increases (T_{send}), with T_{idle} nearly constant in first approximation, the bandwidth usage increases. So, if we have $T_{Rx} < T_{idle}$, we are in the condition of no packet loss.

Now, let us define the condition in which packets are lost. If we have $T_{idle} < T_{Rx} < 2T_{idle} + T_{send}$, this means that we are loosing $1/2$ of the packets systematically. This happens because when a packet arrives, the time spent to read it is longer than the idle time, so when the packet source sends another packet, the Rx buffer of the ethernet controller is still busy due to the reading procedure of the previous packet. This means that for each packet we read, the following packet will be inevitably lost.

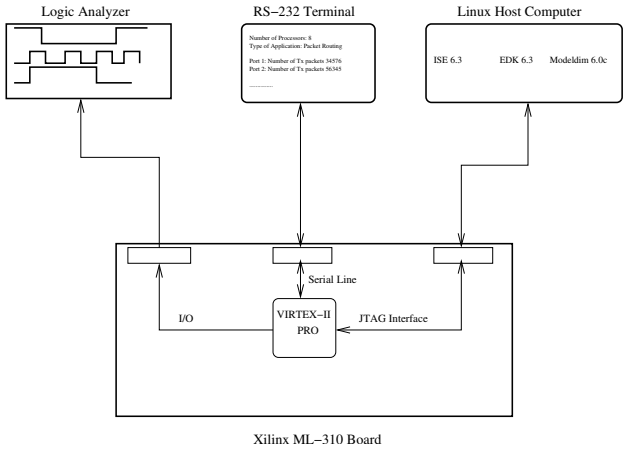


Fig. 3. Configuration of the test environment.

III. Test Platform

The test system is composed of two parts, the hardware one and the software one.

A. The hardware subsystem

All the hardware was mapped in a *XilinX* FPGA, the *Virtex-2-Pro* [26], in order to build a prototype architecture and have the possibility to run all the tests at full speed instead of spending much time simulating the design. The development board was the *XilinX ML-310* [26]. The only feature we were not able to use in this board is the possibility of fully exploit its big I/O pin count. There are two high-speed Tyco connectors but we were not able to find the Tyco adapters all across Europe and in the US. We then limited our probing to a limited number of I/O's that, fortunately, were sufficient for our purposes. As for the other features, we found it a good development platform.

We used a logic analyzer to debug the system during the development process bringing out all the necessary signals. Fig. 3 explains how we set-up the experiments.

The Xilinx *MicroBlaze* was chosen as soft microprocessor, and Xilinx Ethernet evaluation core as the ethernet controller. Microblaze uses the OPB Bus (OnBoard-Peripherals-Bus) of the *IBM CoreConnect* standard [27]. Each Microblaze has a dual port local memory for Instructions and Data. Also a timer and 8 interrupt controllers were added. An RS-232 controller was also instantiated so that all the results were printed on a console. The equivalent gate count reported by XST was of about 2.8 million. The clock frequency of the whole system was 60 MHz and the Ethernet speed was 25 Mb/sec. These values have been chosen to obtain an adequate comparison of the impact that a variable number of master between 1 and 4 may have on the shared-bus performance. Differences on the system clock speed will only lead to a move of the threshold of performances loss, but all the considerations can be made independent from the system frequency.

B. The software subsystem

The entire system was developed using the Xilinx ISE 6.3i and EDK 6.3i, both for the Linux OS. EDK is a co-design tool which helps to quickly develop both the hardware and the software part of a system. Even if we experienced some instability, our system was very complex, so we believe that in more "common situations" its behavior should be more reliable.

On a P4-2.8GHz running Linux, the synthesis, P&R and mapping phase took about 1 hour. As for the development time, we found that the availability of a high number of IPs, even if in a evaluation version, really helps to quickly develop complex systems in weeks.

IV. Test results

We ran two type of applications, a level two ethernet *packet-reply* and a level two ethernet *packet-routing*.

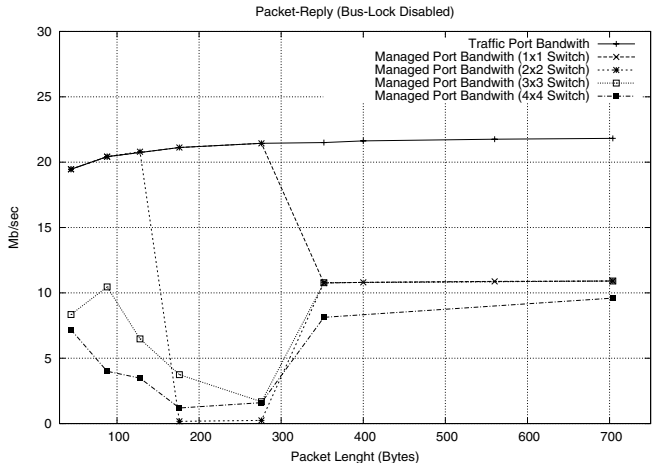
The packet-reply application consists in sending back the packet just read to the same port it was coming from. This means that in the packet-reply application each processor always reads and writes from and to a fixed ethernet controller. Therefore *no resource contentions* were present because in no case two (or more) processors try writing or reading from the same ethernet port.

In the packet-routing application each traffic generator sends a packet to an uniformly random generated destination address. This means that in this case resource contentions are possible. Apart from a possible performance degradation, the contention in accessing an ethernet peripheral may result in malfunctions. This might happen when a processors has started filling an ethernet input buffer and eventually another processor, which aims at writing to the same buffer, receives a grant to access the bus. If the second one starts writing *before* the first one has completed its packet write operation, the packet in the input ethernet buffer will be jammed. Therefore a mechanism of resource locking was introduced using the *Bus-Lock* feature of the OPB bus which is also supported by the Microblaze processor [28]. For this purpose we wrote two small assembler routines to enable and disable bus-locking. When the bus-locking is enabled, the master that was granted will always own the bus until it put the bus-lock down. Therefore the problem of packet jam is avoided as long as each processors leaves the bus grant when it has finished writing an entire packet.

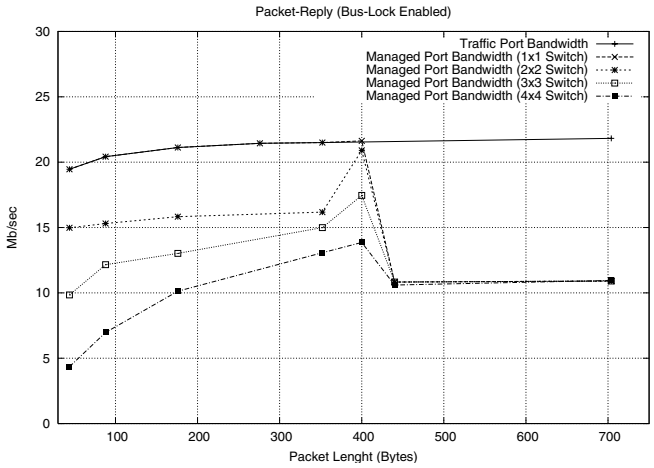
Even if not necessary because contentions are avoided by construction in the packet-reply case, we also made use of this feature in that application to see if there were some performance changes.

For each application we varied the size of sent packets from 44 bytes to 704 bytes. Every test was performed with 1, 2, 3 and 4 processors *contemporary active* on the bus. The results are reported in Fig. 4 for the packet-reply case, where two different situations, bus-lock disabled and enabled, are reported, in Fig. 4.a and Fig. 4.b respectively.

With one processor (Fig.4(a), $\times$ dotted line) we noticed the presence of missing packets starting from a packet length of



(a) Packet-Reply with Bus Lock Disabled



(b) Packet-Reply with Bus Lock Enabled

Fig. 4.

276 bytes, as predicted in Section II. For packet lengths from 0 to 276 bytes, as the packet length increases, the bandwidth usage of the channel increase as predicted in equations (3), (4) and (5).

We also noticed that in the same configuration, using the *bus-lock* mechanism, the threshold above which we begin losing packets increases up to 400 bytes. This is shown in Fig. 4(b). This behavior was expected since in this case the bus arbitration protocol occurs only one time per packet instead of every word.

With two processors things start changing. With the *bus-lock* disabled, the bus subsystem is capable of supporting the traffic until a packet length of 120 bytes. Slightly above this value and until a packet length of about 350 bytes, the packet loss rate increases dramatically. This was not an expected behavior and must be further investigated. What we suspect is that a kind of synchronization occurs between packets arrival and bus request and this leads to that high packet loss. These behavior also manifests with three and four processors as Fig. 4(a) clearly reveals.

With *bus-lock* enabled the same application shows a high improvements in the usage of the bus bandwidth, with a peak when the packet length is around 400 bytes (Fig. 4(b)). This case would correspond to the minimum level of synchronization between the masters of the bus.

The *packet-routing* application gives the possibility to make further considerations on the fact that bus usage strongly depends on traffic statistic and synchronization. The results of our tests on this application are reported in Fig. 5. In theory, the same performances should emerge between packet-routing and packet-reply (with bus-lock enabled) with a shared-bus configuration. However, this would be true if no resource contentions appear. In the packet-reply application, when a packet arrives in the ethernet controller, the probability to find the Tx buffer of the same ethernet controller free is almost 1, so the packet can be successfully sent back. On the contrary,

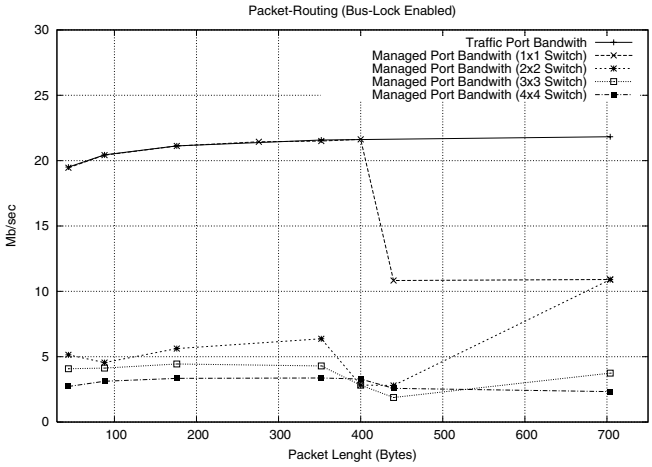


Fig. 5. Packet-Routing Application

in the packet-routing application, this is not true anymore. Even if a processor wins the bus arbitration and is capable of reading an incoming packet from the ethernet controller, now it is possible to find the Tx buffer of the destination ethernet controller busy and this leads to a high performance loss in the bus bandwidth usage.

As a consequence, the results in Fig. 5 compared to those in Fig. 4(b), both with bus-locking enabled, show the relevant performance degradation of the packet-routing case. Clearly, when there is not contention, and this is the case of 1 microprocessor, the behavior is identical.

V. Conclusion

This work intended to evaluate the performance of a shared-bus configuration in a Multiprocessor-SoC configuration when a real-life application is run. As a case study, we mapped a packet-switch application on our system. Our intention was to

stress as much as possible our communication infrastructure and not to propose an efficient networking-application.

What emerges from this set of tests is that the theoretical supported bandwidth of a bus is not the only parameter to be considered in the development of a multiprocessor system based on shared-bus. This is evident in our test environment where, with a system frequency of only 60 MHz and a bus width of 32 bit, a theoretical throughput of 1.92 Gb/s was available. Nevertheless, with a traffic of 25 Mb/s per port, which gives a maximum aggregate bandwidth of only 100 Mb/s with 4 ports, the communication infrastructure was unable to sustain the traffic rate.

This means that a correct statistics of the bus access of each resource is crucial in the development of a complete system. In some case this is possible, especially when the complexity of the whole architecture is not so high. But in future multimillion gate designs, with the need of reconfigurable systems, where hundred of Processing Elements (PEs) will work in parallel, each executing different tasks depending on the state of others PEs, it will be not so easy anymore to accurately predict the intra-chip communication statistics.

We believe that there is and there will be always more need of having flexible, scalable and programmable communication infrastructures, and the NoCs seem attractive for these aspects. This is the first of future works which aim at better clarifying the domains in which current communication solutions are adequate, and subsequently individuate the domains where instead new solutions, like NoCs, are required. We believe that the multiprocessor arena, which already is a hot topic in the SoC domain, is among the applications which require to develop new high performance on-chip communication systems.

VI. Acknowledgment

We would to thank Kaushik Ravindran from Berkley University and Mark Harvey from Xilinx for their kindness in helping us to solve some problems which emerged during the building of our system.

References

[1] International Technology Roadmap for Semiconductors, http://public.itrs.net/

[2] Luca Carloni at al., "A Methodology for Correct-by-Construction Latency Insensitive Design", IC-CAD 1999.

[3] William J. Dally and Brian Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", DAC 2001.

[4] A. Adriahantenaina, H. Charley, A. Greiner, L. Mortiez, and C.A. Zeferino, "SPIN: A scalable, packet-switched, on-chip micro-network", in *Proc. Design Automation Test Eur.*, 2003.

[5] N. Banerjee, P.Vellanki, and K,S,Chata, "A power and performance model for networks-on-chip architectures", in *Proc. Design Automation Test Eur.*, Feb. 2004.

[6] L. Benini and G. De Micheli, "Powering networks on chips", in *Proc. ISSS.*, 2001.

[7] —, "Networks on chips: a new SoC paradigm", *IEEE Comput.*, vol. 35, no.1, pp. 70-80, Jan. 2002.

[8] E. Bolotin, I. CIdon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip", *J. Syst. Architec.*, vol. 50, pp. 105-128, Jan. 2004.

[9] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage, "Networks on silicon: combining best-effort and guaranteed services", in *Proc. Design Automation Test Eur.*, 2002.

[10] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections", in *Proc. DATE*, 2000.

[11] F. Karim, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Oberg, J. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology", in *Proc. ISVLSI*, 2002.

[12] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A acalable, single-chip communication architecture",in *Proc. PACT*, 2000.

[13] J. Liu, L.-R. Zheng, and H. Tenhunen, "Interconnect intellectual property for network-on-chip (NoC)", *J. Syst. Architec.*, vol. 50, no.1, pp.65-79, 2004.

[14] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip", in *Proc. Design Automation Test Eur.*, Feb. 2004.

[15] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade offs in design of a router with both guaranteed and best-effort services for networks on chip", in *Proc. Design Automation Test Eur.*, 2003.

[16] D. Wiklund, and D. Liu, "SoCBUS: Switched network on chip for hard real time embedded systems", in *Proc. IPDPS*, 2003.

[17] ARM, AMBA AXI Protocol Specification, Mar. 2004

[18] OCP International Partnership, Open Core Protocol Specification. 2.0 Release Candidate, 2003.

[19] Philips Semiconductors, Device Transaction Level (DTL) Protocol Specification. Version 2.2, Jul. 2002.

[20] www.st.com

[21] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic performance characteristics of system-on-chip communication architectures", 2001.

[22] V. Lahtinen, E. Salminen, K. Kuusilinna, and T. Hamalainen, "Comparison of synthesized bus and crossbar interconnection architectures.", *ISCAS*, pages V433-V436, May 2003.

[23] Y. Zhang and M. Irwin, "Power and performance comparison of crossbars and buses as on-chip interconnect structures", *Asilomar Conference on Signals, Systems and Computers*, 1:378-383, Oct. 1999.

[24] K. K. Ryu, E. Shin, and V.J. Mooney, "A comparison of five different multiprocessor soc bus architectures", *EUROMICRO*, pages 202-209, September 2001.

[25] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, " Analyzing On-Chip Communication in a MPSoC Environment", in *Proc. Design Automation Test Eur.*, 2004.

[26] http://www.xilinx.com.

[27] IBM CoreConnect Bus Architecture, http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture.

[28] Xilinx MicroBlaze resource page, http://www.xilinx.com/ipcenter/processor_central/microblaze/microblaze_user_resources.htm.