Generic Crossbar Network on Chip for FPGA MPSoCs

David Bafumba-Lokilo, Yvon Savaria, Jean-Pierre David
Département de Génie Électrique, École Polytechnique de Montréal,
Montréal, CANADA

Abstract— **Networks-on-Chip (NoCs) have emerged as a new design paradigm to implement MPSoCs that competes with the standard bus approach. They offer more scalability, flexibility, and bandwidth. Nevertheless, FPGA manufacturers still use the bus paradigm in their development frameworks. In this paper, we study the complexity and performances of a FPGA implementation for a crossbar NoC. We propose a generic architecture and characterize its complexity, maximum frequency of operation, and global throughput for NoCs supporting 2 to 8 nodes. Results show that FPGA-based designs would benefit from such architecture when high throughput must be reached. Finally, we present a fully functional 3x3 NoC interconnecting a PowerPC and 2 Xtensa processors implemented in a VirtexII Pro FPGA.**

Keywords—**Network-on-Chip, NoC, FPGA, Crossbar**

I. INTRODUCTION

The complexity in circuit design grows rapidly, still validating Moore's Law. Therefore, the ability of implementing complex architecture in a single chip always presents new challenges. One of the issues met by designers when implementing large SoCs is the communication between their (numerous) components. Buses are an increasingly inefficient way to communicate, since only one source can drive the bus at a time, thus limiting bandwidth.

NoCs are increasing in popularity because of their benefits: *larger bandwidth,* thanks to high performance switching networks, and *lower power dissipation* through shorter wire segments. Communications in large SoCs are so important that many researchers have adopted the NoC approach. They are drawing inspiration from computer networks, thus most SoC are usually an adaptation of previous work on networks for parallel and distributed systems [1][2].

Many topologies and architectures have been investigated. The challenge consists in offering the best connectivity and throughput with the simplest and cheapest architecture or methodology. This is very well illustrated in [3], where researchers propose a two-level FIFO approach in order to simplify the design of the arbitration algorithm and improve the bandwidth. Unfortunately, this method tends to be expensive in terms of hardware.

FPGA manufacturers such as Xilinx and Altera offer embedded tools to help their customer design complex Multi-Processor Systems-on-Chip (MPSoCs). Nevertheless, their environments only offer the bus paradigm or point-to-point connections. More complex MPSoCs may require higher bandwidths than a bus can offer, or may need to be more area efficient than point-to-point connections.

Strangely, if several NoC generators are reported in the literature, very few FPGA implementations and performance evaluations are available. In [4], researchers have described a NoC generator, which is used to create a synthesizable NoC description. NoCGEN uses a set of modularized router components that can be used to form different routers with a varying number of ports, routing algorithms, data widths, and buffer depths. A way to design MPSoCs in FPGAs in a short time is proposed in [5]. The authors use a set of predefined components and a high-level language to build the MPSoC in their tool environment: ESPAM.

In this paper, we propose a generic crossbar based NoC with two parameters: the number of nodes and the data width. The architecture relies on CASM [6], an intermediate level HDL which enables a very simple implementation of the Round-Robin algorithm to prioritize some transfers and avoid deadlocks (compared to CASM, VHDL is considered to be a low level HDL). Synchronization is implicit in the language. We study the area, frequency, and throughput achievable for NoC implementations ranging from 2x2 to 8x8 nodes on Altera and Xilinx circuits.

The rest of the paper is organized as follows. Section 2 presents the architecture of the generic crossbar NoC. Section 3 details its implementation. Section 4 reports the implementation results. Section 5 discusses the design of a fully functional implementation of an MPSoC with a 3x3 crossbar based NoC. The last section concludes this work.

II. ARCHITECTURE

With the crossbar topology, each source can send data to each target in a straight way. This topology is illustrated in Figure 1 in the case where there are 4 sources and 4 targets. In this paper, we only consider NxN topologies, since most of the time, a processor requires read and write accesses to the NoC. Nevertheless, the approach is easily extensible to NxM

978-1-4244-2332-3/08/$25.00 © 2008 IEEE

or even partial crossbars, when it is not mandatory that each device on the network be connected to all the other devices.

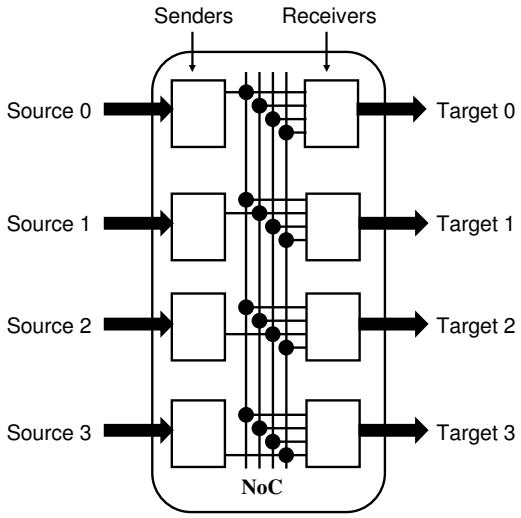

Fig 1: Example of a 4x4 crossbar NoC

The architecture is based on two main building blocks. Blocks on the left hand side are called *senders*, and blocks on the right hand side are called *receivers*. The sender has the role of getting a token from its (left) input port and sending it to the right receiver according to its address. Inversely, the receiver must check all its input ports to detect if one sender attempts to send data to this port. When this is the case, a transfer occurs from the sender to the receiver.

A common problem arises when several senders are ready to send their data to the same receiver. The round robin approach offers an elegant solution to fix this issue [7]. The receiver assigns a priority token to one sender. As soon as a transfer occurs, the token goes to the next sender in a circular way. If the sender that has the priority token is not ready, its nearest neighbor (in the round) that is ready to send data will complete its transaction. In this way, a transfer can occur at each clock cycle for each receiver that has at least one sender that is ready.

Other researchers have concentrated their efforts on designing a Round-Robin arbiter generator tool [8] to accelerate the implementation of efficient Round-Robin arbiters. Our work takes advantage of the high level features of our CASM HDL to implement the Round-Robin algorithm with a state machine, as described in the next section.

III. IMPLEMENTATION

Our NoC is generated from an API (Application Programming Interface) written in Java. The API takes two parameters (network size N and data width w) and generates a network of dedicated senders and receivers written in CASM.

CASM (Channel-based Algorithmic State Machine) is an intermediate level HDL developed in our research group [6]. This language enables an easy description of Algorithmic State Machines (ASM) that process and exchange data tokens over channels in a self synchronized way. Moreover, compared to Verilog or VHDL, the language has higher-level constructs enabling "state calls" and even recursion. The language is aimed at describing circuits at an algorithmic level and let the CASM compiler manage all the hardware, logical and electrical details.

For brevity, implementation details will be illustrated on a very basic example depicted in Figure 2: a 32-bit, 2x2 crossbar NoC.

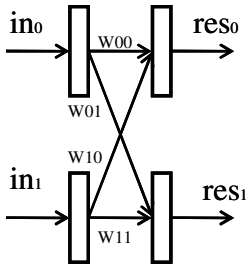

Fig. 2: 32-bit 2x2 module

In this context, there are two senders, two receivers and 4 virtual communication channels. Our Java API generates the following code for <u>one</u> sender:

```
unsigned input in0{protocol="FS"}[33];
unsigned post W00{protocol="FS"}[32];
unsigned post W01{protocol="FS"}[32];

ASM  {
      unsigned register xa[33];
      unsigned register xb[33];

s1:   xa:=in0; goto s2;

s2:   if (xa.[32..32] == "0"b)
            W00:=(unsigned)xa.[31 .. 0];
            xb:=in0; goto s3;
      else
            W01:=(unsigned)xa.[31 .. 0];
            xb:=in0; goto s3;
      end;

s3:   if (xb.[32..32] == "0"b)
            W00:=(unsigned)xb.[31 .. 0];
            xa:=in0; goto s2;
      else
            W01:=(unsigned)xb.[31 .. 0];
            xa:=in0; goto s2;
      end;
}
```

The input in0 is 33-bit wide (1 bit for the address and 32 bits for the data). "FS" stands for "Fully Synchronized", which means that the compiler will generate all the synchronization logic required to manage the data as tokens.

The ASM has 3 states (s1, s2 and s3). Two 33-bit registers (xa and xb) are defined to temporarily store the data coming from the input port in0. In each state, the transfer symbol ":=" guaranties that one and only one transfer will occur.

In state s1, the state machine waits until a first data token is available at input port in0. As soon as this occurs, the data is written in register xa and the control is passed to state s2.

In state s2, depending of the address bit, two transfers must occur before leaving the state: sending the data stored in xa and acquiring a new data in xb. These transfers may occur in any order and possibly in a single clock cycle. The control is then passed to state s3. State s3 plays a role similar to s2 except that xa and xb are swapped.

The code generated by our API for one receiver is the following:

```
unsigned output res0{protocol="fs"}[32];

ASM  {
s0:   if (W00.dout.sync.rts)
            res0 = W00; goto s1;
      elsif (W10.dout.sync.rts)
            res0 = W10; goto s1;
      else goto s0;
      end;

s1:   if (W10.dout.sync.rts)
            res0 = W10;
            goto s0;
      elsif (W00.dout.sync.rts)
            res0 = W00;
            goto s0;
      else goto s1;
      end;
}
```

Basically, each state maps a different position of the priority token. State s1 prioritize the channel W00 (coming from input in0) while state s2 prioritize the channel W10 (coming from input in1). As soon as a transfer occurs, the control is passed to the next state.

Since all the transfers occur in the "Full Synchronized" mode, each channel is composed of three signals:

- Channel_data (driven by the source)
- Channel_rts (activated by the source when ready)
- Channel_rtr (activated by the target when ready)

A transfer occurs when both Channel_rts and Channel_rtr are activated at the same clock rising edge, which enables a transfer at each clock cycle when sender and receivers are continuously ready. Such protocol is very close to a FIFO protocol and can interface with it almost directly.

In conclusion, our implementation only requires two registers for each sender, one multiplexer for each receiver and a few logic gates to implement the control ASMs. An important feature to notice is that the whole description is generated by our Java API from only two parameters and automatically translated into VHDL by our CASM compiler.

IV. RESULTS

A set of NoCs ranging from 2x2 to 8x8 ports with the proposed architecture has been implemented. A VHDL simulation allowed verifying that all packets were correctly sent to their target. The test-bench sends 5000 random vectors (address + data) at each input and check that they reach their destination. This test also gives statistical values for global throughputs which are reported in Table 1. Note that the test-bench attempts inserting one vector at each cycle and that vector remains at the input if the insertion is not successful.

Table 1: Throughput and latency

NxN	Data sent	Cycles	Latency (cycle)	Global throughput (word/cycle)
2x2	10000	6713	1,34	1,49
3x3	15000	7337	1,47	2,04
4x4	20000	7663	1,53	2,61
5x5	25000	7853	1,57	3,18
6x6	30000	7969	1,59	3,76
7x7	35000	8046	1,61	4,35
8x8	40000	8149	1,63	4,91

Results demonstrate that the throughput of this NoC is much higher than that of a bus, where the maximum throughput is 1 data/cycle. All the modules have been synthesized by the Synplify Pro 8.9 tool to fit in Virtex-5 and Stratix-3 FPGAs. We measured the performances in terms of area and frequency. The results are reported in Tables 2 and 3 for NoC with 32-bit data paths.

The resulting frequencies of operation are all above 100 MHz and up to 350 MHz, which is quite good for a non pipelined architecture. The area is very small too since the 8x8 NoC would only consume 1.5% of the combinational resources available in the xc5vlx330ff1760-1.

Table 2: Area and frequency (xc5vlx330ff1760-1, Xilinx)

NxN	Estimated freq. (MHz)	LUTs	Registers
2x2	243,4	228	146
3x3	187,3	553	292
4x4	139,5	1604	336
5x5	131,2	2278	557
6x6	126	3467	734
7x7	118,2	3676	843
8x8	111,1	4956	986

Table 3: Area and frequency (ep3sl150f1152c2, Altera)

NxN	Estimated frequency	ALUTs	Registers
2	377,9	215	148
3	284,3	442	255
4	244	764	363
5	197	1593	535
6	199	2072	670
7	160,4	2860	808
8	170,6	3554	952

V. APPLICATION

We have developed a SoC architecture consisting of two Xtensa processors, one PowerPC processor, and a 3x3 NoC module. The system is aimed at prototyping video algorithms, but this is still a work-in-progress. The PowerPC will manage the I/O streaming while the Xtensa processors and their dedicated instructions (TIE) will perform video processing. Figure 3 describes this architecture.

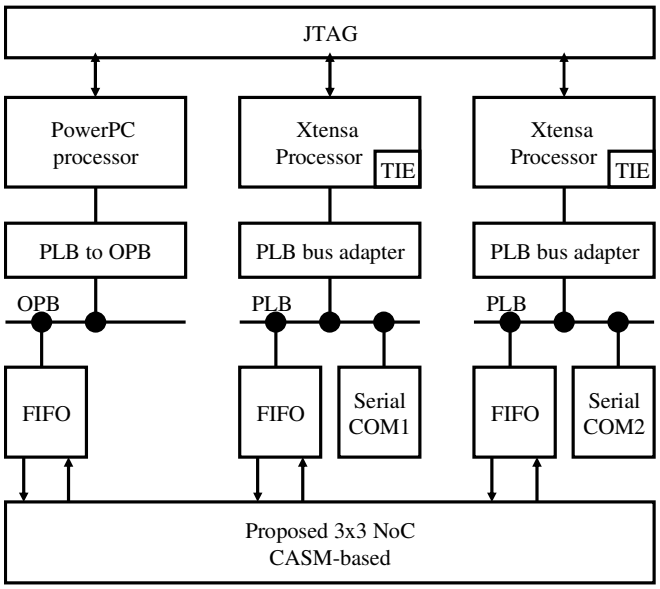

Fig 3 : SoC system

The whole system has been implemented in a FPGA XC2VP100. Many tests have been conducted with parallel data transfers between all the processors. The architecture is fully functional but no video algorithm has been tested at this time. It is of interest that we lost many weeks in attempting to integrate the Xtensa processor in the standard environment (PLB and OPB bus) because of tools and IP issues. However, once we were able to add a standard FIFO to each processor, it took just half a week to complete the network thanks to our generic NoC.

VI. CONCLUSION

In this paper, we described a generic NoC template and discussed its implementation on FPGA technology. Compared to the standard bus paradigm or point-to-point connections, this approach proved to be an easy and efficient way to rapidly interconnect modules in a System-on-(Programmable) Chip. A 32-bit 8x8 NoC only requires 1.5% of the available resources of a Virtex xc5vlx330ff1760-1 FPGA while a point-to-point or FIFO-based approach would be very costly to implement since 64 FIFOs should be instantiated in addition to data routing and synchronization logic. As chips size still continue to increase exponentially, we think that such approach will be more and more useful for future designs and we suggest that FPGA manufacturers should integrate such technology in their standard development flow.

REFERENCES

[1] L. Benini and G. De Micheli, "Networks on chip: a new SoC paradigm", IEEE Computer, vol. 35, no. 1, Jan. 2002.

[2] K. Shashi, J. Axel, M. Mikael, O. Johny, S. Juha-Pekka, F. Martti, T. Karia and H. Ahmed, "A network on chip architecture and design methodology", IEEE Computer Society, Washington, DC, USA, 2002, pp.117.

[3] H.Po-Tsang, H. Wei, "2-Level FIFO Architecture Design for Switch Fabrics in Network-on-Chip", Circuits and Systems, 2006, ISCAS 2006. Proceedings. 2006 IEEE International Symposium on**.**

[4] J. Chan, S. Parameswaran, " NoCGEN: A Template Based Reuse Methodology for NoC Architecture", VLSI Design, 2004. Proceedings. 17th International Conference on, 2004 Page(s): 717 – 720.

[5] H. Nikolov, T. Stefanov, E. Deprettere, "Efficient Automated Synthesis, Programing, and Implementation of Multi-Processor Platforms on FPGA Chips**",** Field Programmable Logic and Applications, 2006. FPL '06. International Conference on, pp.1-4.

[6] J.P. David, E. Bergeron, "An Intermediate Level HDL for System Level Design", Forum on specification and Design Languages (FLD), Lille, France, September 2004.

[7] X. Gao, Z. Zhang and X. Long "Round Robin Arbiters for Virtual Channel Router", Computational Engineering in Systems Applications, IMACS Multiconference on, 4-6 Oct. 2006, pp. 1610-1614.

[8] E. Shin, V. Mooney, G. Riley, "Round-robin Arbiter Design and Generation", Georgia Institute of Technology, 2002.