An FPGA Implementation of a Scalable Network-on-Chip Based on the Token Ring Concept

Karim Hadjiat, Francis St-Pierre, Guy Bois, Yvon Savaria.
École Polytechnique de Montréal
QC, Canada
{karim.hadjiat, francis.stpierre, guy.bois, yvon.savaria} @ polymtl.ca

Michel Langevin, Pierre Paulin.
STMicroelectronics,
ON, Canada
{michel.langevin, pierre.paulin } @ st.com

***Abstract*— In this paper, we present an FPGA prototype implementation of a Rotator-on-Chip (RoC), a simple and scalable novel network-on-chip (NoC) based on the token-ring concept. The reported prototype design is generic with respect to the number of nodes and data channels. We report synthesis results showing a O(*N* log *N*) area complexity, where *N* represents the number of nodes, with a quasi-linear aggregate bandwidth growth. The slice utilization is less than 25% on a Xilinx VP100 for a 32 nodes version of the RoC, supporting an aggregate bandwidth of about 12 GB/s. Moreover, the number of channels can be easily configured to trade-off area versus performance. These configurations have been validated by simulation and implemented on a FPGA development board.**

I. INTRODUCTION

As the number of transistors in chips continues to grow, designers struggle to take advantage of these available resources to produce improved SoC designs. These systems are complex. To manage that complexity, they must be divided into multiple blocks: processors, memories, I/O controllers, specialized hardware blocks, etc. This creates a need to integrate an efficient network to exchange data between the multiple blocks or nodes composing the SoC [1][2] [3].

We present an FPGA prototype implementation of a Rotator-on-Chip (RoC), which is a simple and scalable novel NoC based on the token-ring concept [4]. We report experimental results for both area and bandwidth, in terms of number of nodes and data-path width. These results can help system architects performing trade-offs between NoC performance and cost.

Our architecture differs from others in the literature, but still uses important basic concepts of NoCs. In Section 2, we first summarize these concepts and review related works. Then, we present the rotator-on-chip functionality and its implementation in Section 3. Section 4 presents results and analysis of different configurations. Conclusion is provided and future works are suggested in Section 5.

II. NETWORKS ON CHIP AND RELATED WORKS

NoCs are communication device transmitting packets between a set of nodes (or IP blocks) [3]. NoC's structures are typically composed of sets of switches linked by communication channels. The way those switches are connected defines the network topology, while the scheduling decision at each switch point determines when and on which path the packets are flowing. Both topology and scheduling algorithm have an impact on the achievable performance of the NoC with respect to latency and aggregate bandwidth [5].

Popular NoC topologies are mesh, ring, and tree [5]. Examples of tree-based topology NoC are SPIN [6], while examples of mesh-based are Cliché [7] and Aethereal [8]. Other NoC topologies are possible, like irregular topologies customized for the application [9]. Some of these NoC topologies have been characterized in terms of ASIC area versus performance [5], while some have been prototyped on FPGA [10][11][12].

These NoC are implemented by interconnecting routers resulting in multiple queuing points in a path between source and destination. This may impact traffic performance, resulting in suboptimal resource utilization.

Recently, high performance products emerged using ring topologies in which only one queuing point exist on a path between source and destination. IBM and its Element Interconnect Bus (EIB) for the CELL processor makes use of four rings to exchange data between 12 processing elements [13]. This NoC implementation is specialized for a specific product.

The RoC implementation we present in this paper is also based on the ring principle. It is based on a simple controller supporting a number of parallel paths that can be configured to support a given bandwidth.

III. THE ROC IMPLEMENTATION

The RoC functionality, detailed in [4], is inspired from a telecommunication switch-architecture [15], but the proposed implementation is totally different since our target is a simple and scalable on-chip solution. In the following discussion, it is assumed that we want to connect together *N* resources. For instance, a resource can be a processor, a memory, or a dedicated-hardware component. Each *resource* must be attached to a node (network) interface.

Fig. 1 presents the top view of a 4-nodes version of the RoC implementation. Each *node core* contains a *node interface* and a *bank*. A bank contains *buffers*. Each buffer from the buffer bank is part of a ring and each ring constitutes a *channel*. Each channel is composed of *N* buffers. Therefore, the RoC of the Fig. 1 is composed of 4 nodes and 2 channels, and each channel is composed of 4 buffers.

A buffer contains a packet, and the transfer of information follows a clockwise rotation. That is, packets make hops between nodes in a ring channel which is primarily a chain of buffers that are interleaved with multiplexers. Parallel ring channels allow minimizing internal blocking. At the extreme, using as many ring channels as the number of nodes, internal blocking can be totally avoided, but this is the most costly solution.

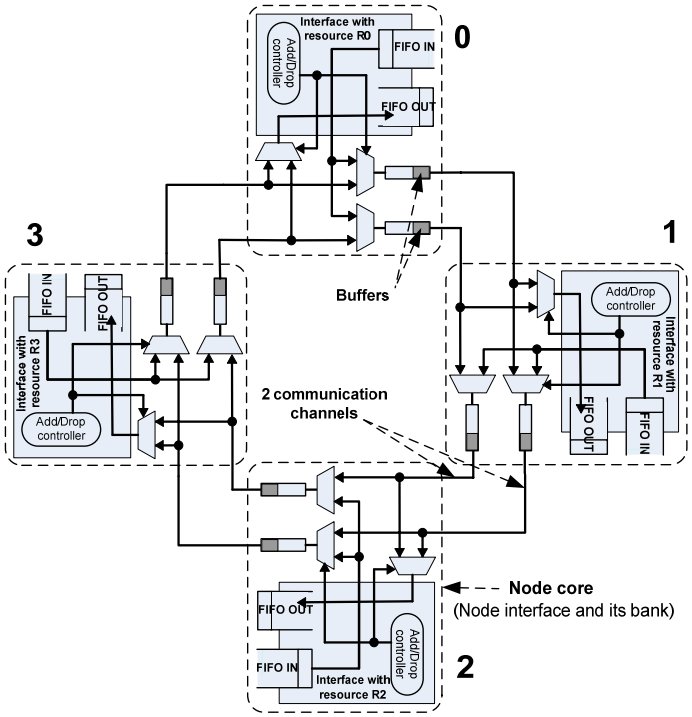

Figure 1. The RoC top-level RTL model based on shift registers for *N*=4 .

A node core, composed of a node interface and its bank, is associated with each resource for interfacing the communication with the network channels. For simplicity, the current implementation is including a node interface with separate dedicated FIFOs for input and output traffic.

Packets are composed of four fields: a source address, a destination address, a payload (data) and an information field. The packet size is configurable in our implementation.

Each buffer is associated with a control bit indicating if the buffer is *occupied* or *free*. Based on this information, a node interface can send (i.e., add) a packet in the core switch, while a packet can be delivered to the node interface (i.e., drop) from the core switch. This is accomplished by adding and dropping packet from the bank of buffers associated with the nodes under the direction of the add-drop controller (Fig. 1).

- *Drop operation* This operation is achieved when a buffer transports a packet *P* destined for the node itself. Then, *P* is written in the *FIFO_OUT* (through the multiplexer) and the buffer is now considered free.
- *Add operation* This operation is achieved when a node interface must send a packet *P* to the destination node *D*. The operation succeeds if two conditions are met: 1) there is a free channel (including the one released during the drop operation) and 2) there is no buffer transporting already a packet for destination *D*. Then, *P* is transferred from the top of the *FIFO_IN* to the buffer (through the multiplexer) and the buffer is now considered occupied.

The second condition for the add operation is important, since it guarantees that for any destination, there is at most one buffer in a bank containing a packet for this destination. Then, packet ordering is guaranteed since there is no possible contention during the drop operation.

The detailed implementation of the node core supporting a configurable number *C* of channels and packet data width *W* is shown in Fig. 2, highlighting in particular the structure of the add-drop controller.

The *rx_bitmap* input (Fig. 2) is a bit-vector of length *C*, one bit per channel, indicating if the associated channel is free or not. This bit-vector is processed by the add/drop controller to detect a packet to be dropped and to select on which channel to add a packet. This functionality is implemented using 5 combinatorial components, connected as shown in Fig. 2, and described as following:

1) check_for_drop: This component produces a bit-vector drop_select where at most one bit is set. This bit corresponds to the channel that drops the packet. If such a channel is selected, the wr_fifo_out signal will be set to enable the push of this selected packet inside the FIFO_OUT.

2) update_bitmap_drop: This component simply updates the rx_bitmap into bitmap_after_drop to capture the fact that a channel has been freed.

3) check_dst_conflict: This component is verifying if the packet at the head of FIFO_IN can be injected without creating a destination conflict with any destination packet already in the channels. Notice that if the FIFO_IN is empty (i.e., if ack_fifo_in is 0), adding a packet is trivially not possible.

4) select_free_channel: This component produces a bit-vector free_select where at most one bit is set. This bit corresponds to a free channel on which a packet could be added.

5) update_bitmap_add: This last component produces the rd_fifo_in signal to pop the packet from the FIFO_IN on the condition that add_possible is true and there is a free channel. At the same time, it generates the bitmap_after_add, which is the resulting bitmap after the drop and add actions. Before being forwarded as the rx_bitmap of the next node (after being buffered as new_bitmap), bitmap_after_add is used to enable the load of the packets, i.e., those going through (not dropped) as well as, potentially, the added one.

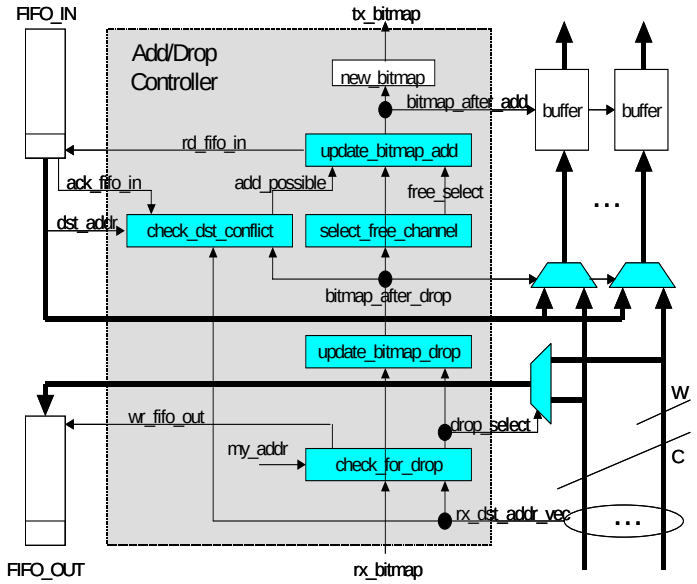

Figure 2. Node core implementation of Fig. 1

Thus, at each clock cycle, the *add-drop* controller implementation permits to add and drop a packet from the core switch. The combinational critical path is to generate the signal *bitmap_after_add* from the value of *rx_bitmap*.

Finally, each resource is connected to an entry point of the RoC (node) through a wrapper interface, so that different communication protocols can be used. That consists in managing the communication between the resources and the input/output FIFOs of the nodes.

IV. RESULTS & ANALYSIS

This section presents synthesis results of the RoC RTL implementation described in the previous section. They are obtained with ISE tool targeting a Xilinx Virtex-II Pro FPGA [14]. We concentrate on implementation using packet size *W* of 41 bits and FIFO size of 15 packets. Default synthesis has been experimented only, without attempting RTL code restructuring and optimization either for area or speed. When applicable, these results will be compared with those of a 2D mesh FPGA implementation reported in [12].

The first set of reported results concentrate on the configurable RoC itself without its processing resources. Table I presents such results for N = 4, 8, 16 and 32 nodes, while the number of channels is fixed to $C = \log N$. It shows the obtained area in term of number of slices and flip-flops, as well as the clock frequency. Table I also reports the resulting minimal round-trip latency, as well as the maximal bandwidth supported by a node and the maximal aggregate bandwidth supported by the complete RoC. These extreme latency and bandwidth values are traffic dependent, and may be worst in particular for the case of high-conflicting traffic for the same resource.

From the 4 RoC configurations of the Table 1 (N/C = 4/2, 8/3, 16/4 and 32/5), we can observe that the average number of slices per node is respectively 576/4 = 144, 1544/8 = 193, 3800/16 = 238 and 9217/32 = 288 slices. The growth complexity of each node depends mainly on the number of channels *C*. Each new channel increases the node complexity by approximately 50 slices; in particular, there will be about 350 slices used per node for a 64 nodes version of the RoC with 6 channels.

For comparison, a 2D mesh router FPGA implementation described in [12] consumes 352 slices and 478 flip flops for each node. This complexity is independent of the network size. Thus, for about 64 nodes or less, hardware complexity of the RoC is lower than the cited mesh implementation; for instance, a 4x4 mesh network takes 5632 slices of a Xilinx XC2VP30, while the 16 nodes RoC takes only 3800 slices. Notice that this mesh implementation used an 8-bit datapath, while the RoC implementation has 32-bit datapaths (not including control).

TABLE I. RESULTS OF SYNTHESIS FOR POSSIBLE ROCS WITH XILINX ISE: N = 4, 8, 16 AND 32 NODES, AND THE NUMBER OF CHANNELS EQUAL TO LOG N

Number of N/C	Slices	Flip Flops	Freq. (MHz)	Latency L (cycles)	Node Bandwidth (MBps)	RoC Bandwidth (GBps)
4/2	576	724	138.2	8	552.8	2.21
8/3	1544	1784	125.8	12	503.2	4.02
16/4	3800	4240	112.3	20	449.2	7.19
32/5	9217	9824	100.8	36	403.2	12.90

For the case of the minimal round-trip latency, compare to a mesh implementation on the Xilinx XC2VP30, a 9 nodes RoC provides a round-trip latency of 13 cycles, to be compared to about 70 cycles on average for the 3x3 mesh [12]. Therefore, the RoC latency is about 5 times smaller. Notice that the reported clock frequency for the mesh implementation in [12] is about 33MHz, while for the RoC implementation, the clock frequency is about 100MHz.

For the case of the RoC maximal aggregate bandwidth, this value depends on the clock operating frequency and the number of nodes. For instance, as shown in Table I, the clock frequency obtained for a 32 nodes RoC configuration with 5 channels is 100.8MHz; this results in a maximal node data bandwidth of 403.2MBps (4Byte data payload x 100.8MHz), and a maximal aggregate bandwidth of 12.9GBps (32 x 403.2MBps).

Fig. 3 shows the percentage of slices used for a complete SoC including N resources while considering a Xilinx XC2VP100 as FPGA device (maximum of 44,096 slices and 444 BRAMs). Each resource is composed of a Microblaze microprocessor [14], OPB wrappers and peripherals, requiring 800 slices and 8 BRAMs. For different sizes of a network (4, 8, 16... 32 nodes), it is showing the portions used by the RoC itself, and by the resources, respectively. For instance, a 16 node system requires a total of 16600 slices (37.6%). Note that only 3800 of these slices (8.6%) are required for the configurable RoC itself.

We have performed experiments to characterize the effective utilization rate of the NoC interface, referred to as the supported load. Fig. 4 shows the maximal load observed for different configurations of the 32-node RoC, from 1 to 16 channels, when injecting packets at each source for destination randomly selected using a uniform distribution.

It can be observed that the effective load is growing up almost linearly from about 8% to 55% when the number of channels is increasing from 1 to about 10; the effective load in this experiment is saturating at about 58% when 12 channels is reached.

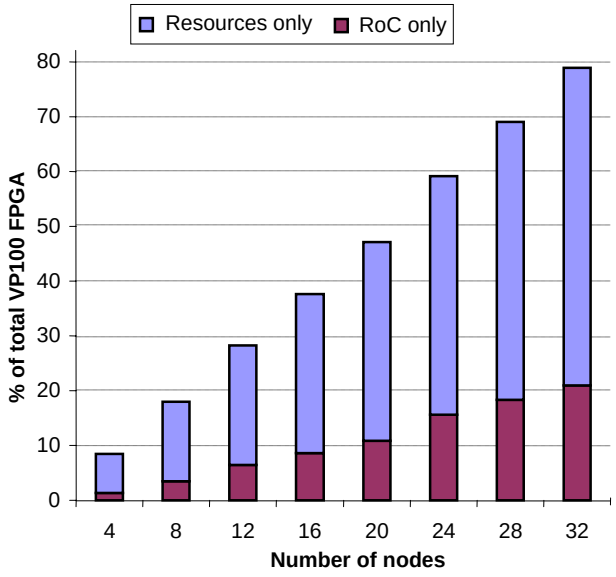

Fig. 3. Percentage of use on a XC2VP100.

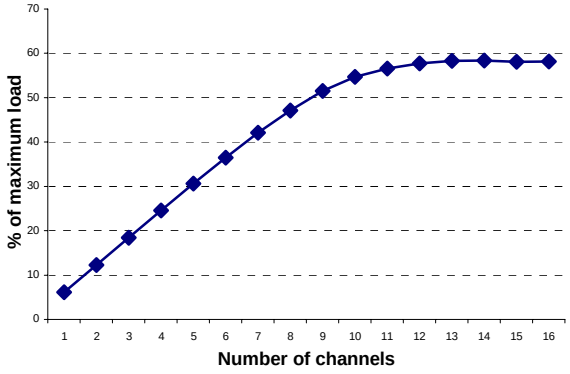

Fig. 4. Maximal load supported for an 32-node RoC.

These results demonstrate an important property of the proposed RoC architecture and implementation in which a designer can easily select a NoC configuration appropriate for the design requirements, investigating trade-offs between area and performance.

Notice that as for any NoC, the achievable performance is dependent on the traffic type. On one hand, it is worst with traffic having high level of contention for the same destination, since the bottleneck will be at this destination. On the other hand, it is much better with traffic having low contention. For the RoC, we experimented with many different one-to-one mapping between source and destination, and observed that a traffic load of 70% and beyond can be supported with a 32 nodes version using 12 channels.

V. CONCLUSION

We have presented an implementation of a Rotator-on-Chip, a parameterized and scalable NoC architecture based on the token ring concept. The implementation has been validated by simulation and implemented on a FPGA. This implementation is generic with respect to the number of nodes N and channels C, and have a complexity of $O(N \log N)$ when restricted the number of channels to $\log(N)$. For 64 nodes or less, hardware complexity of the RoC is lower than the mesh implementation. We have also discussed the tradeoffs between the number of communication channels and supported traffic load. In terms of latency, the RoC latency was shown to be 5 times smaller than a mesh.

Currently, we are investigating extensions of the proposed RoC implementation, some of them being described in [4]: using multiple FIFOs in the node interface, supporting the transfer of packet per smaller segments, supporting different traffic classes and using a bidirectional or two-level hierarchical configuration.

REFERENCES

[1] A. Jantsch, H. Tenhunen, Eds., “Networks-on-Chip”, Norwell, MA: Kluwer, 2001.

[2] W. Dally, B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, in Proc. of DAC, pp. 684-689, 2001.

[3] L. Benini, G. De Micheli, “Networks on chips: A new SoC paradigm”, IEEE Computer, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[4] F. Deslauriers, M. Langevin, G. Bois, Y. Savaria, P. Paulin, “RoC: A Scalable Network on Chip Based on the Token Ring Concept”, Proc. of NEWCAS, Gatineau, Canada, June 2006.

[5] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, “Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures”, IEEE Trans. on Computers, vol. 54, no. 8, pp. 1025-1040, Aug. 2005.

[6] A. Adriahantenaina, H. Charlery, A. Greiner and L. Mortiez, “SPIN: a Scalable, Packet Switched, On-Chip Micro-network”, DATE’2003, p.70-73.

[7] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, A. Hemani, “A network on chip architecture and design methodology”, in Proc. IEEE Computer Society Annual Symposium on VLSI, Apr. 2002, pp. 117-124.

[8] K. Goossens, J. Dielisson, A. Rãdulescu, “Aethereal Network on Chip: Concepts, Architectures, and Implementations”, IEEE Design & Test of Computers, vol. 22, no. 5, pp. 414-421, Sept./Oct. 2005.

[9] D. Bertozzi, L. Benini, “Xpipes: A Network-on-Chip Architecture for gigascale Systems-on-Chip”, IEEE Circuits and Systems Magazine, vol. 4, no. 2, pp. 18-31, 2004.

[10] G. Brebner, D. Levi, “Networking on chip with platform FPGAs”, in Proc. IEEE International Conference on Field-Programmable Technology, Dec. 2003, pp. 13-20.

[11] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, R. Lauwereins, “Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs”, in Proc. 12th International Conference Field-Programmable Logic and Applications, Sep. 2002, pp. 795-805.

[12] B. Sethuraman, P. Bhattacharya, J. Khan, R. Vemuri, “LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip”, in Proc. ACM Great Lakes Symposium on VLSI, 2005, pp. 452-457.

[13] T. Chen, R. Raghavan, J. Dale, E. Iwata, “Cell Broadband Engine Architecture and its first implementation”, Nov. 29, 2005. http://www.ibm.com/developerworks/power/library/pa-cellperf/

[14] Xilinx Inc. http://www.xilinx.com.

[15] M.E. Beshai, E. A. Munter, “High Capacity ATM Switch”, U.S. Patent 5745486, Apr. 28, 1998.