Resource-aware Design for FPGA-based NoC

Giovani Bissani Pedroso
Dept. of Computer Engineering
Universidade Estadual do Rio Grande do Sul
Guaíba, Brazil

Abstract— **The concern about the use of resources is always present in a design with FPGAs because the number of available resources is finite. The usage of FPGAs characteristics for the project benefit is important and necessary. Another important issue is the flexibility in the implementation of components described in a hardware description language (HDL). This flexibility comes from the development of component libraries with the same behavior and the same functionalities, but with different hardware representations in FPGAs. This work proposes the use of these concepts to optimize the resources of a Network-On-Chip (NoC) for FPGA. The idea is to use memory elements that are present in the device instead of flip-flops and look-up tables (LUTs) where possible. The logic synthesis tools have a very important role in this process, but it is not enough. The purpose of this work is to use concepts of finite state machines (FSM) mapped into ROMs and this is done through a more specific and targeted HDL coding to characteristics of the target device. Furthermore, it is also proposed to use memories to assist in the calculation of routes in the network, further reducing the use of Adaptive Logic Modules (ALMs). The optimization techniques were applied in two different versions of NoCs and the results were compared. A reduction in the use of basic resources was possible (ALMs) of up to 41% and an increase in the maximum operating frequency of up to 19% compared to versions without optimization.**

Keywords—FPGA; Network-On-Chip; Block RAMs; optimization; resource usage; adaptive routing

I. INTRODUCTION

With the increased number of cores in SoCs (Systems-On-Chip), bus-shaped interconnections or crossbars are at their operating limits [1]. A feasible alternative that allows communication between cores with a higher operating limit is the NoC, network-on-chip [1]. The concept of this alternative form of interconnection is the same as using Ethernet protocol: packets to traffic information [2]. Other important and relevant features of NoCs are their flexibility and scalability.

With the advancement of new technologies, larger FPGAs are being produced. This growth allows large SoC designs to be feasible in FPGAs [1]. Therefore, there is a need to study specific versions NoC, which are sensitive to characteristics of FPGAs. The study of NoCs in FPGAs specifically was chosen because these programmable devices are widely used in embedded systems designs [3]. Considering the development of programmable logic, ASICs (Applicable Specific Integrated Circuits) have better performance, less use of silicon and better performance of power that FPGAs [4]. But still, FPGAs are widely used for their characteristics of rapid prototyping and flexibility, which allow the time-to-market of FPGA-based designs to be shortened [4]. Whereas SoCs are increasingly complex with a very large number of cores that need to communicate with each other, specific NoCs for FPGAs are indispensable.

As an alternative to buses, it is known that the NoC has a better performance at the expense of a greater resource utilization [2]. Therefore, it is essential that your architecture meets many different implementation scenarios. Sometimes projects implemented in FPGA ends up spending many basic features such as flip-flops and LUTs instead of memory elements, which is a common feature. Thus, a sensitive NoC to this becomes very useful.

The communication structure, whatever it may be, must be transparent, imperceptible to the rest of the project. So, the control of the resource type used is important. The logic synthesis tools have an important role in optimization of resources, but does not completely solve the problem. As regards FSMs, for example, there are numerous coding techniques, which do not use of all types of FPGA resources [5]. The description of the NoC must lead to a proper understanding of the synthesis tool generating the most appropriate result for the application.

This work proposes a way to optimize NoCs using memory elements to generate an alternative to SoC designs in FPGAs. The target device is an FPGA from Altera Cyclone V SoC, but all that will be discussed in this work can be used for any other manufacturer. This paper is organized as follows: section II presents the motivation for this work. In Section III, a brief discussion of related work will be done and, later, in Section IV, will be described the methodology used in this paper. In section V, will be described the results and will be made all analyzes. Finally, Section VI summarize all this work with the final considerations.

II. REVIEW OF RELATED WORKS

A. Works related to optimizations

Previous work are references to perform the work described herein. The concern with a conscious use of resources in the FPGA is quoted in the Kwa's paper [6], which proposes

sharing FPGA memory elements for use in input buffers of the routers in a NoC. The so-called BRS, Block RAM Split, uses the concept of true dual port of Block RAMs. Each memory element has two write ports and two read ports and these operations can be done simultaneously. So, in a router construction with five ports (north, south, east, west and local) are used three and not five Block RAMs. This is very important because in an FPGA, resources are finite and used by existing nobler implementations that need a way to communicate with each other.

Besides the issue of optimization of input buffers, it is needed to evaluate the use of resources in the rest of the logic that describes the flits routing algorithm that come out of the buffers and are targeted to your specific output ports. These components are described in HDL and use FPGA resources, so it is plausible a carefully look for some form of optimization.

Considering that the hardware description using finite state machines is something very common, mapping it using memory elements results in a saving in the use of other features like flip-flops and LUTs, which become available for use of cores on a SoC. This is discussed in the work of Garcia-Vargas [7], which proposes a method for mapping a FSM in memory elements.

Memory can also be used to optimize the calculation of routes as an instruction table. Routers that assess static conditions to determine the path of flits may have advantage by using this alternative.

B. Routing algorithms

Some adaptable routing algorithms inorder to increase the performance were studied so that they could be submitted to the proposed optimizations to determine the obtained gain. These works were evaluated for the complexity and the ability to become optimizable. The first was the algorithm proposed by Chang [8], which proposes a routing method based on switch and channel congestion levels. In addition, it proposes a predictive calculation of contention that further refines the choice of routing. Because it has a lot of associated arithmetic, it did not prove to be a good candidate for use in this work.

The second proposal to be evaluated was the work of Dehyadegari [9]. The idea behind this algorithm is to use fuzzy logic to find the best path for packets. This process considers the free space in the buffers and the time between flits to calculate a cost for each output of the router. The concepts of fuzzy logic would be passive of memory optimization, but the high complexity of this algorithm prevented its use in the work.

Finally, it was evaluated the Enhanced Dynamic XY (EDXY), which is an evolution of the classic XY routing, making it adaptable to possible disturbances in the network [10]. The routers receive from each of their neighbors a value called "stress value" (SV), which is basically the situation of the input buffer. In addition, there is also another flag that indicates row/column congestion. The combination of these two parameters enhances XY algorithm. This was the algorithm chosen to be used in this work since it presents some advantages such as a very similar FSM description to the XY algorithm and a reduced route calculation that depends on flags and simple comparisons. The EDXY algorithm will be better detailed in the next sections.

III. METHODOLOGY

The methodology applied to the design of this new NoC can be described as follows.

- Implementation of a NoC with XY routing algorithm.
- Optimization of NoC using memory elements.
- Improvement of the original NoC using adaptive routing algorithm.
- Optimization of the enhanced NoC using memory elements.
- Infrastructure generation for functional and performance testing
- Comparative analysis of results in performance tests and resource usage reports.

All development of this work was done with a Cyclone V SoC target architecture. In Helio development kit, there is a Cyclone V SoC FPGA with 41910 ALMs and 5662kb of usable internal memory (Block RAMs).

A. Implementation of a NoC with XY routing algorithm.

The chosen architecture for the development of this work is the mesh type. The network is formed by 16 routers arranged in a 4x4 grid. There are many descriptions of NoCs type mesh with XY routing algorithm available and many work on that too [11]. Nevertheless, it was decided to develop an entirely new description of a NoC so that it could be the most appropriate architecture for further optimization. Considering a classic construction for NoC routers (Fig. 1), a FSM is described for controlling the reading process of the input buffers, and setting the destination of each flit read. In this version, there is no kind of priorities and routing intelligence. The process is based on round-robin where all buffers are treated equally in a circular polling system.

The flits have a two-bit header to make the indication of the beginning and end of each packet. That is, the total data width is the width of valid data plus 2. Fig. 3 shows the structure of the packet used in this work. Flit is a contraction of the term “flow control unit” and represents the small pieces of the packet in a network-on-chip [2].

For each new incoming packet is made the calculation that defines to which output port will be sending the related flits. The value of the flit header for packet start is 3. The route is set only at that time, so all flits of this packet use the same output port. This port remains locked until the last flit of the packet pass. After that it is released to meet the next demand. The XY described algorithm makes the displacement of flits in the X axis first and then adds shifting on Y axis. The router receives only the usage information of the neighbor buffer to decide to send the flit or not, once the route has been calculated. If more

than one arrived packet has the same destination, which came first has priority in shipping. Fig.2 shows the resulting finite state machine.

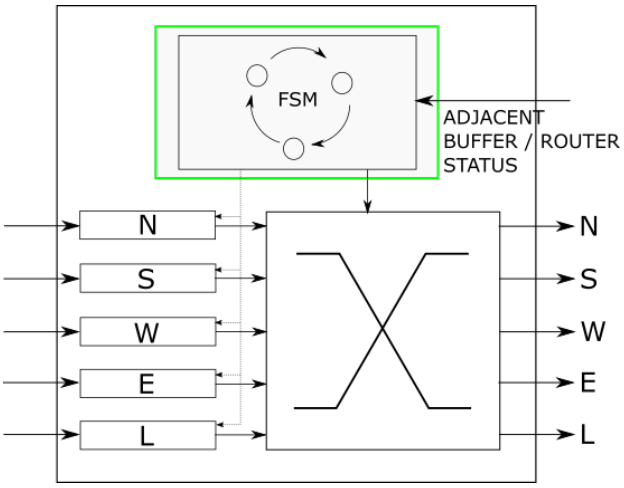

Fig. 1. Router architecture

In RR_ST state, the counter that defines which buffer will be read is incremented. In READ_ST state, a flit is read from the selected buffer. The DV_ST state has no action; it is just a time to stabilize the buffer data read for later use. In CHK_FLIT_ST state is made the analysis of the flit: if it is a header, it has the target information and from this is done the route calculation. If a flit data, it is forwarded to the output. If the last flit, releases the output port to be used by a different input buffer. In ROUTE_ST state is done calculating the routing and in SEND_FLIT_ST state, flit is routed to its respective output.

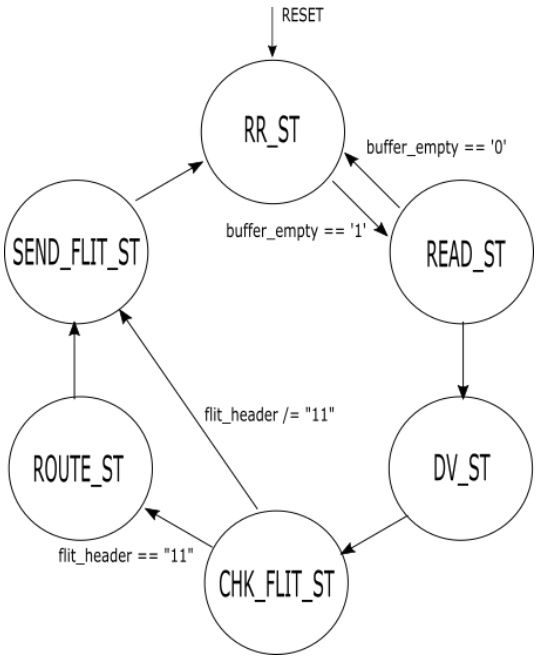

Fig. 2. XY resulting finite state machine

B. *Optimization of NoC using memory elements*

Since the primary NoC is described, we have started the development of the design optimization process. According to what was discussed in [7], it is possible to map a FSM in a read-only memory. Each address corresponds to a state. The contents of the memory address must have: state outputs, part of the address indicating the next state and operators along with the entries make the calculation of the next state. For a very complex FSM, the width of data becomes too large, and a modification of the state machine to be simplified is necessary. Basically, the number of states is increased to reduce the complexity of operators and outputs. As memory elements in FPGAs have sizes of the order of kilobits, is not a waste to increase the number of states often if it entails a reduction in the complexity in calculating the next state.

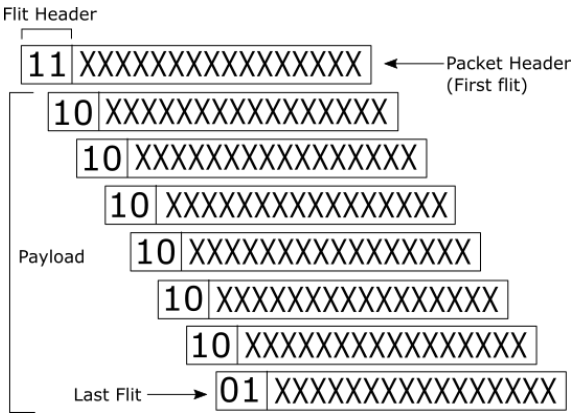

Fig. 3. Packet structure example (16 bits)

This process of simplification should be done considering the bit-oriented operations. In the case of the router to be optimized, the FSM has 3 inputs: the status of the input buffer selected by a round-robin counter, the status of the input buffer of adjacent router and a flag indicating that all flits correspondents to a packet have been processed; and 2 outputs: the read control of the input buffers and enabling the sending flits to the next router.

The content of each address of the ROM should contain the outputs as most significant bits, a portion of the next state as the least significant bits and operators which result in operation with inputs to generate the complete address of the next state. In Fig.4, is highlighted the portion of the read data that matches the address of the next state of the FSM.

Thus, it was made the FSM mapping to memory. In addition, the route calculation has also been optimized in memory. In a NoC which uses XY routing, route calculation considers only the destination of the flit and the position of the actual router. So, this can be described in an instruction ROM as follows: the address is formed by the destination of the flit and the corresponding data indicates the output port to be used. For this work, we used a 4x4 grid, then 4 bits are required considering a cartesian plane - 2 bits for the horizontal axis and 2 bits for the vertical axis.

To further optimize the implementation, each router has an individual route instruction ROM. So, the necessary addressing of the memory consists of only 4 bits and not 8, if it was considered the position of the current router. Because of this additional memory, the FSM had to be changed and a few states have been added. Basically, they are passing states that allows the information read from the route instruction ROM to be stable for use in the routing of flits.

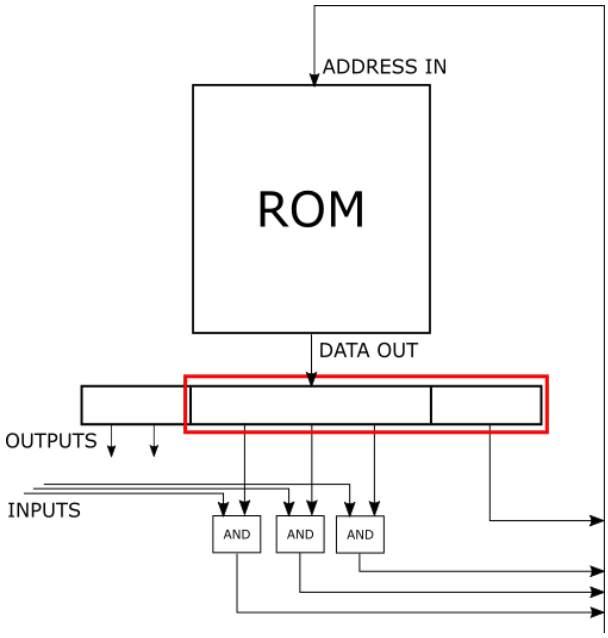

Fig. 4. FSM mapping technique

C. *Improvement of the original NoC using adaptive routing algorithm*

Once validated the finite state machine optimization method proposed earlier, it is possible to advance in improving the NoC. There are many proposed adaptive routing algorithms such as Dehyadegari's work [9] that proposes the use of fuzzy logic for calculating routes or Chang's work [8], which proposes a routing based on switch and channel congestion levels. They are very interesting ideas, but use a lot of arithmetic in their algorithms and thinking of the proposed optimization style in this work do not fit as candidates.

There is another algorithm presented by Lotfikamran [10], which proposes the use of basic information of routers so that the route is calculated in the simplest way. The EdXY, which means Enhanced Dynamic XY, is an evolution of DyXY proposed by Li [12], which in turn comes from an update on the XY classic, which makes it adaptable to possible disturbances in the network.

The DyXY, Dynamic XY, uses information from neighboring routers to calculate the route of flits. Instead of the packet go all one axis and then the other to get to your destination in DyXY, the decision of which output port to use in a situation where two ports are possible router choices, considers the input buffers situation of adjacent routers connected to the output ports. Thus, a path with less congestion is chosen and the packet arrives faster to their destination.

In the evolution proposed by Lotfikamran [10], the EDXY, in addition to the status of the input buffers of the adjacent routers, there is another information between routers called Congestion Flag (CF). This flag indicates congestion of lines/columns that helps refine further routing. This is important when the destination router is a hop from the row/column of the current router. When the flit arrives in the same row/column of the destination router, the path becomes a straight line to the end. Then, the CF indicates whether the path is congested or not so that the best choice is made. In the example of Fig. 5, the current router (0,0) is 1 hop from the row/column of the destination, the router (1,1). So, in addition to evaluating the occupation of input buffers in (0,1) and (1,0), you need to assess the congestion of the column between (1,0) and (1,1) and of the line between (0, 1), and (1,1). This is the use of Congestion Flag. An important feature is also that being an evolution of the XY algorithm, EDXY is deadlock-free. This is an algorithm that does not create routes that loop do packet is lost on the network.

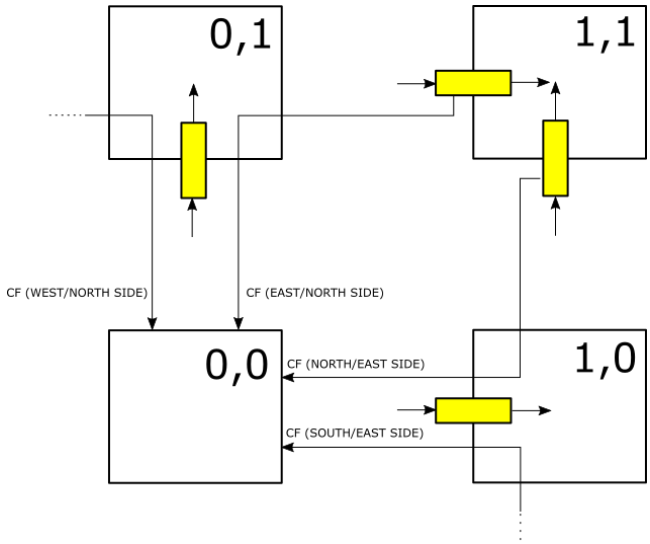

Fig. 5. EDXY Congestion Flag

From the point of view of HDL description, the evolution of the XY algorithm for EDXY is simple. It takes a few extra wires between routers so that can be used to CF. The route calculation is more complex, but the main control FSM remains the same. The read access routine to the input buffers remained made in the form of a circular pooling. There is no definition of priority levels between the router input ports, which is guaranteed using round-robin.

D. *Optimization of the enhanced NoC using memory elements*

For reasons of similarities described above, it was not necessary effort to change the main FSM in the evolution of the XY algorithm for EDXY. Therefore, in this implementation we used the same ROM described for the previous NoC. The

big challenge for this release was the optimization of the route calculation, which considers many more variables than the first algorithm.

For the Enhanced Dynamic XY, the route calculation depends on the destination of the packet, the status of the input buffers of adjacent routers and corresponding CF values for each output port. As the number of variables is too large it has not been possible to generate a single route instruction ROM. The solution was to divide the route calculation in two stages. In the first stage is defined from the flit allocation if the CF should be considered or not. It depends on the position of the current router and the destination of the flit. If not required, the route is calculated only considering the occupation of the input buffers of adjacent routers. Thus, it was possible to create a route instruction ROM for that first step.

The EDXY is an evolution of DyXY algorithm [10] as has been said previously. The separation of the calculation in 2 steps can be described as: the first step corresponds to DyXY algorithm and the second stage as the complemement granted by the EDXY: use of congestion Flags, as shown in the Fig. 6.

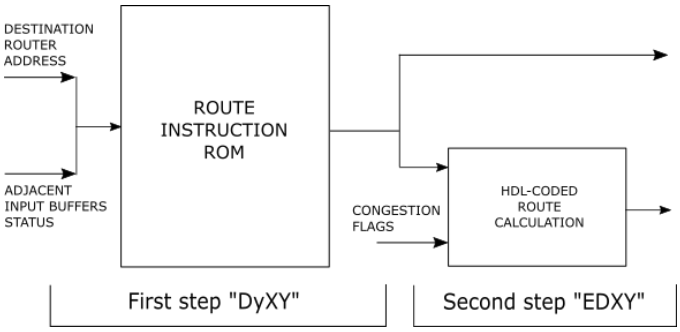

Fig. 6. EDXY steps

The addressing of this memory is composed of the packet destination address and status of adjacent routers. To the routers addressing 4 bits are needed, as discussed above. The status of the buffers of the adjacent routers are provided by 1 bit per output port, then are 4 bits corresponding to 4 router output ports (north, south, east, west). Thus, the addressing of the ROM route instruction is composed by 8 bits. Again, we use the technique to generate individual ROMs for each NoC router, so the position of the current router is not necessary.

For each memory address, there is data with 2 separate information: an output port indication to flit, when the destination router is not on the same row/column of destination router or 1 hop away from it and the relative position of destination router from the current. The latter is important for the EDXY algorithm, because it indicates the need to use the Congestion Flag for route calculation.

Considering the example of Fig. 7, the current router is in position (2,1). Routers position (0,1), (1,1), (3,1), (2,2), (2,0) and (2,3) are in the same row or column current router, then destinations are not dependent on any other variable in question. For these destinations, route instruction ROM only indicates the corresponding output port to them. The router position (0,3) is not in the same row or column of the current router and your row/column is not one hop away from the current router. Therefore, the route to a flit destined for (0,3) depends only on the status of the adjacent buffers - in this case: (1,1) and (2,2). For destinations described here, only the ROM route instructions are necessary.

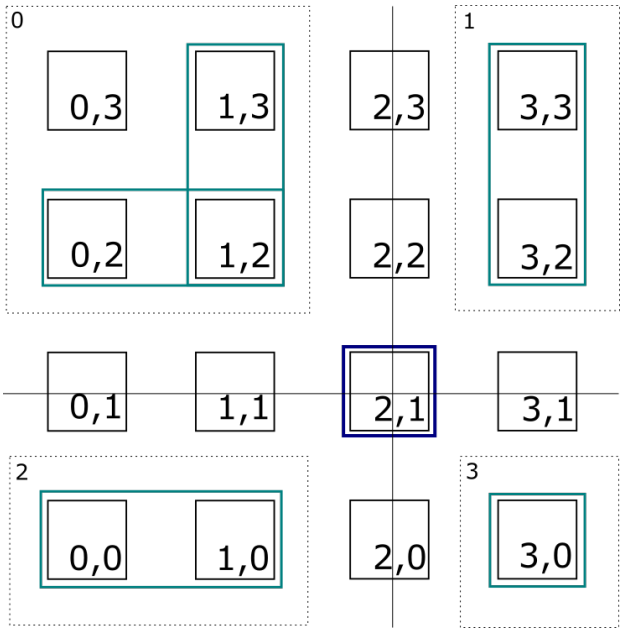

Fig. 7. EDXY routing instruction ROM mounting process example

Routers position (0,0), (0,2), (1,0), (1,2), (1,3), (3,0), (3,2) and (3,3) are part of the group whose rows or columns are one hop away from the current router (2,1). Routes to these destinations need the information provided by the CF to be defined. Then the route instruction ROM provides position information on these routers to the second stage of EDXY algorithm. Quadrants centered on the current router are defined and numbered from 0 to 3 as shown in Fig. 7. From this information and the associated CF is possible to calculate routes to these destinations.

This second stage was described in HDL and is implemented usually through the synthesis tool using essential resources such as flip-flops and LUTs. In the end, the route calculation implementing this EDXY algorithm is hybrid with a portion described in memory and partly described in HDL.

E. Infrastructure generation for functional and performance testing

After the development of the 4 versions of NoC, it was necessary to create a test infrastructure to qualify the network at first. Usually in a NoC, routers are connected to elements called NIC (Network Interface Controller). These components make the interface between cores and routers. Each access

requested by a core is converted into a packet by the NIC and reaches the router.

To play the role of the cores and its NICs were developed generators and receivers of packets. These elements can generate packets with configurable destination information and known content that can be checked at the end of its route network. Then, there were added 16 Packets generators/receivers on the tests network - one for each router. Thus, it was possible to generate configurable network traffic so that all versions of NoC could be validated.

To measure network performance, these components were developed as follows: except the first packet sent, packets were only sent as others were received. So, when creating fixed mappings between routers on the network, you could reach the packet traffic limit. Moreover, each Packet generator/receiver can measure the number of clock cycles between received packets to define the network latency. With that feature, it was possible to measure network performance. This infrastructure allowed flexibility in testing enabling the creation of numerous scenarios as exemplified in Fig. 8.

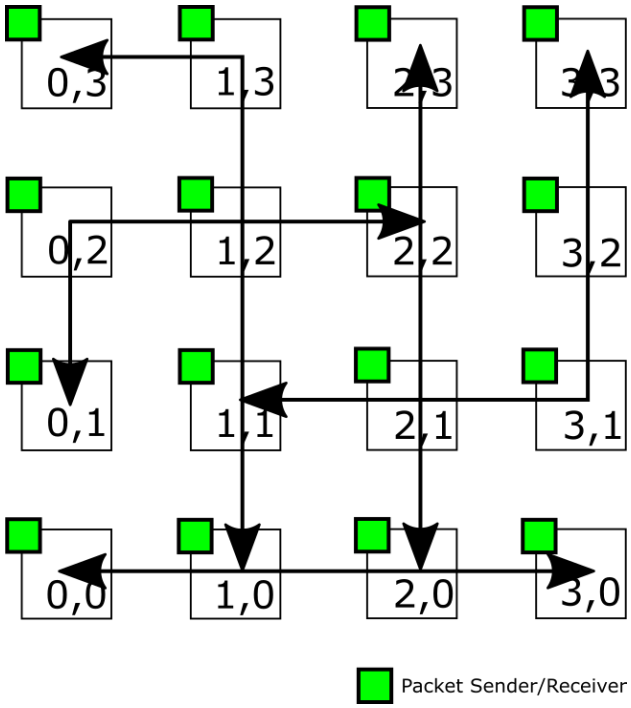

Fig. 8. Tests infrastructure

IV. ANALYSIS AND RESULTS

The initial focus was the functional tests to ensure that all versions NoC implemented did not have any problems. Then it was generated a first scenario with mappings between the network routers such that all were equidistant. All origins and destinations were spaced by 3 hops in the network. Considering the layout of the routers in the network illustrated by Fig.8, the following mappings between Packet generators/receivers were made:

- (0,0) – (1,2);
- (0,1) – (1,3);
- (1,0) – (0,2);
- (1,1) – (0,3);
- (2,0) – (3,2);
- (2,1) – (3,3);
- (3,0) – (2,2);
- (3,1) – (2,3).

The test was performed in Cyclone V SoC FPGA present in Helio Board and through the Altera SignalTap tool was observed that the packets travel correctly in 4 versions of NoC that have been proposed for this work and the result of the measure latency between packets received by the router origin and destination is described in table I. The average latency is measured in clock cycles and it is an average between the measurements of all 16 network routers.

For this test was used a traffic packet with 8 flits each: one containing a header with the destination information and the other 7 with a defined standard. As discussed previously, the packet injection rate is variable and it depends on the network traffic. The sender/receiver components send a packet to its peer shortly after the arrival of a packet from its peer.

TABLE I. LATENCY MEASUREMENTS

NoC Version	Latency (avg.)
XY without optimization	66,2
EDXY without optimization	57,0
XY optimized (ROM)	121,2
EDXY optimized (ROM)	110,7

After the functional tests with the above scenario, it was made implementations of all 4 versions of NoC through Altera's Quartus II 15.0 tool for the same FPGA Cyclone V SoC and the following results of the synthesis were observed:

- Number of used ALMs
- Maximum operating frequency
- Number of memory bits used / number of Block RAMs

All the testing infrastructure was removed from the implementation, leaving only the components related to the four versions of NoC proposals for this work. This was done so that the tool could provide more realistic possible results. Table II and Table III show comparisons between versions. The number of ALMs used per router is an average of which was used in the implementation by the 4 routers that are in the center of the NoC as Fig.7: (1,1); (1,2); (2,1) and (2,2).

For each version were generated 2 NoC implementations: 16-bit valid data and 32-bit valid data. That is, are versions with data buses between routers of 18 and 34-bit respectively. This is because each flit is marked by 2 bits indicating the beginning, middle and the end of packet.

TABLE II. IMPLEMENTATION RESULTS (16 BITS)

NoC Version	ALMs	Fmax	Memory bits
XY without optimization	82,12	94,3 MHz	0
EDXY without optimization	97,37	81,0 MHz	0
XY optimized (ROM)	55,30	100,3 MHz	5680 (2 M10K)
EDXY optimized (ROM)	57,17	96,4 MHz	7424 (2 M10K)

TABLE III. IMPLEMENTATION RESULTS (32 BITS)

NoC Version	ALMs	Fmax	Memory bits
XY without optimization	122,35	92,6 MHz	0
EDXY without optimization	134,95	89,7 MHz	0
XY optimized (ROM)	90,02	98,6 MHz	5680 (2 M10K)
EDXY optimized (ROM)	94,52	92,7 MHz	7424 (2 M10K)

The ALMs, Adaptive Logic Modules are basic units within Altera FPGAs. Inside them there are the basic features such as LUTs, flip-flops, full-adders and carry chains. The number of ALMs used by component is information that the Quartus II tool provides and from that information that is made the estimation of the use of resources in the FPGA for each logic synthesis held.

The results of the functional test showed that improved versions through EDXY routing algorithm had better performance than the versions with XY routing, but there is a difference between versions with and without optimization with memory elements. In addition to the increase in the number of states in the FSM of optimized versions, which explains this difference is the latency of read accesses to the ROM. There is a time between the address is stable on the input memory and the data is stable in output. This in practice makes the result of memory access can only be used in the next clock cycle. Therefore, the optimized ROM FSM has a worse performance than the classical implementation in HDL.

Nevertheless, the gain in optimizations with memory elements in the use of resources was quite significant. In versions with 16-bit valid data come to a saving of 32.66% for the NoC with XY algorithm and 41.28% for the NoC with EDXY algorithm. In versions with 32-bit valid data, the economy was 26.42% for the NoC with XY algorithm and 29.95% for the NoC with EDXY algorithm. Only 2 memory elements were needed for router.

This reduction in the number of elementary features (ALMs) used causes the step of placement and routing tool be made by faster. With fewer resources to route, the maximum operating frequency rises. In the version of NoC EDXY 16-bit was a 19% increase in the maximum frequency of operation. This facilitates the synthesis of any project that uses this type of NoC.

V. CONCLUSION

This paper presented a NoC optimization proposal using memory elements in FPGAs. Often the fight for resources on a FPGA-based design is very intense, therefore having the ability to adapt is the key. The optimization techniques were applied to two different models NoC: XY routing algorithm and the EDXY routing algorithm.

Map a FSM in a ROM is not a straightforward operation. They are needed some changes and not all the features are preserved. As can be observed in experimental tests, due to the latency of access to Block RAMs FPGA, the FSM lost performance causing higher latency values. Still, the gain regarding to the use of resources made up for the loss of performance. In 16-bit version EDXY was possible a saving of 41% using elemental resources (ALMs), and a 19% increase in the maximum operating frequency.

The objective of this work was achieved, but there is still plenty to do to evolve components with conscious use of FPGA resources. Consider optimizations in the input buffers in addition to what has already been done in the core of the router are a future possibility. Another aspect to be worked on is to reduce the impact of latency access the Block RAMs in the memory-mapped FSM performance and evaluate other traffic scenarios.

REFERENCES

[1] W. J. Dally and B. Towles, “Route packets, not wires: On-chip interconnection networks,” in *Proceedings of the 38th Annual Design Automation Conference*, Jun. 2001, pp. 684–689.

[2] S. Pasricha, N. Dutt, On-Chip Communication Architectures. Morgan Kaufmann Publishers, 2008.

[3] J. J. Rodriguez-Andina, M. J. Moure and M. D. Valdes, "Features, Design Tools, and Application Domains of FPGAs," in *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810-1823, Aug. 2007.

[4] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203-215, Feb. 2007.

[5] K. Kuusilinna, V. Lahtinen, T. Hamalainen and J. Saarinen, "Finite state machine encoding for VHDL synthesis," in *IEE Proceedings - Computers and Digital Techniques*, vol. 148, no. 1, pp. 23-30, 2001.

[6] J. Kwa and T. M. Aamodt, "Small virtual channel routers on FPGAs through block RAM sharing," *2012 International Conference on Field-Programmable Technology*, Seoul, 2012, pp. 71-79.

[7] I. García-Vargas, R. Senhadji-Navarro, G. Jiménez-Moreno and A. Civit-Balcells, “ROM-Based Finite State Machine Implementation in Low Cost FPGAs,” *IEEE International Symposium on Industrial Electronics (ISIE)*, Vol. 1,pp. 2342-2347, Vigo (Spain), 2007.

[8] E. J. Chang, H. K. Hsin, S. Y. Lin and A. Y. Wu, "Path-Congestion-Aware Adaptive Routing With a Contention Prediction Scheme for Network-on-Chip Systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 113-126, Jan.2014.

[9] M. Dehyadegari, M. Daneshtalab, M. Ebrahimi, J. Plosila, and S. Mohammadi, "An Adaptive Fuzzy Logic-based Routing Algorithm for Networks-on-Chip", in *Proceedings of 13th IEEE/NASA-ESA International Conference on Adaptive Hardware and Systems (AHS)* pp. 208-214, June 2011, USA.

[10] P. Lotfi-Kamran, et al., “EDXY - A low cost congestion-aware routing algorithm for network-on-chips”, *Journal of Systems Architecture*, v.56, I.7, 2010.

[11] Chandravanshi, Ruchika, and Vivek Tiwari. "Network on Chip Router Architecture Performance Analysis by using VHDL." *International Journal of Computer Applications*, vol. 140, pp.13 , 2016.

[12] M. Li, Q.-A. Zeng, W.-B. Jone, “DyXY – a proximity congestion-aware deadlock-free dynamic routing method for network on chip”, in: *Proceedings of the Design Automation Conference*, 2006, pp. 849-852.