ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **18**, Number 1, 2015, 79–92

Efficient Hardware Implementation of Snapshotting Algorithms for NoC Applications

Andrei ZENE[1], Claudiu-Teodor CHIRAP[1],
Octavian CRET,[2], Lucia VĂCARIU[2]

[1]National Instruments Romania, Cluj, Romania
[2]Department of Computer Science,
Technical University of Cluj-Napoca, Cluj, Romania

E-mail: {andrei.zene,claudiu.chirap}@ni.com,
{Octavian.Cret,Lucia.Vacariu}@cs.utcluj.ro

Abstract. The NoC domain has known a big development lately, acquiring an ever growing importance in the context of hardware miniaturization. However at this point it is still hard to observe and / or debug what is going on inside the chip, be it for debugging purposes or for long-running processes like computational biology simulations which could gain a great improvement when ran on an FPGA chip. Therefore, this paper proposes hardware implementations of three of the most important snapshot algorithms: Lai-Yang, Li et al. and Mattern, which could be used in order to achieve observability on a long-running process physically implemented on an FPGA target. The setup is based on the layers architecture, making it easy to separate the snapshot algorithm from the application or the intercommunication network. The intercommunication network was generated using the CONNECT NoC generator and the snapshots are sent to a PC via UART for displaying. The algorithms were compared from three points of view: operating frequency, throughput and resource usage. Based on the obtained results, we show that the Mattern algorithm is the best candidate for an effective hardware implementation (both from the resource usage and speed points of view).

Key-words: FPGA, NoC debugging, network-on-chip, snapshot algorithms, observability, computational biology.

1. Introduction

A few decades ago, Chandy and Lamport came with the concept of snapshot algorithms [1]. The purpose of these algorithms was to capture the global state of a distributed system. The distributed systems in which these algorithms were devised, were "global clock"-less, message-passing systems. At that time, the authors used snapshot algorithms to detect stable properties as "the computation has terminated" or "the system is deadlocked". As the time passed, distributed systems started taking other forms. Nowadays, there have been designed and implemented SOCs that include different IP cores operating at different frequencies. Furthermore, in order to interconnect these IPs, message-passing NoCs were proposed.

This new setup started to recreate a scenario where snapshot algorithms are of great utility. To the best of our knowledge there are surprisingly few papers that highlight the utility of snapshot algorithms in NoC-based applications. One of the most significant is [2], in which a debugging solution in SoCs that used different clock domains was proposed based on a snapshot algorithm.

Debugging in hardware is indeed a great utility that can be brought by taking regular snapshots. Detecting stable properties is also of a great importance. But there is another feature that is more specific to hardware: observability, which is essential for long-running processes. In this direction, computational biology simulations constitute very good examples of applications that need this feature. FPGAs offer a great level of low-granularity (bitwise) parallelism, thus, being suitable for biological simulations. However most of the time when these simulations are run on FPGA, the user ends up with a *start button* and a *results file* without having a clue of how the process arrived to those results. In this scenario having regular snapshots is of great help since they allow the user to better understand the process that is simulated. Furthermore, in case of a failure, the process could be resumed from the last snapshot and not from the beginning.

Ever since biological systems started to be understood in more detail in structure as well as behavior, computer simulations have been of great interest since they would provide a good framework for simulating the living systems, the end purpose being the in-depth understanding of complex organisms (ex. Fig. 1). These areas of research were dedicated to simulating different details of the biological systems, therefore having different levels of abstraction [3]:

- Genetics - where the mechanisms were translated to the behavior of genes and the level of detail required precise knowledge of such mechanisms;
- Molecules - where the emphasis was on the chemical reactions and the impact to one another;
- Cells - that described different types of cell behaviors based on their internal structure or functionality
- Organs - where the internal, highly granular mechanisms were abstracted away in order to just present the understood way of behavior.

A small conclusion that can be drawn from the above categories is that modeling a complex biological system requires an interconnection and communication of such items. These will have to be connected over a network in order to ensure a realistic behavior. As part of the internal behavior, the models' implementation can vary between either a strong mathematical form or rule-based systems [4]. Whereas rule-based systems take away the noise caused by insufficiently understood models, the mathematical systems rely a lot on precise, error-less mathematical models therefore giving a more detailed view. The drawbacks of both methods make the usage of a hybrid model highly valuable.

In terms of the interconnection network, emulating time is a complex feature. Some models can be used following a time-driven execution where modules would execute only at specific moments in time. However, when emulating items such as biological cells, their reaction is asynchronous and usually event-driven - therefore time needs to be determined through a more complex mechanism. In order to see the state of the network system at one point, a snapshot mechanism can be used to ensure coherency. Based on the operating parallelism for the simulated cells, the implementation on general purpose CPUs would be tightly coupled to the number of units available, since any execution of more than one cell would use a lot of processor sharing. However, while operating on FPGAs, the modeled biological units will operate independently in parallel such that the network and interconnection would be more similar to real biological systems.

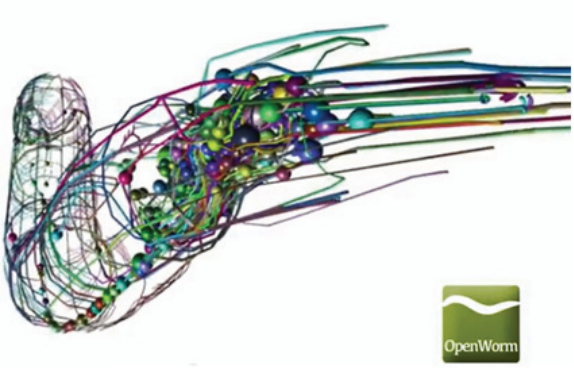

Fig. 1. C. elegans, a microscopic worm of $\sim$ 1000 cells. [Source: openworm.org].

2. Theoretical Background and Previous Work

2.1. Snapshot problem

Let S be a distributed system, whose nodes (or processes) are linked through channels: SC_{ij} (channel between nodes i and j). As way of communication, nodes only use messages. For each message in the system we define the following two events:

- $send(m_{ij})$ happens when message m is sent by node i towards node j

- $recv(m_{ij})$ happens when message m sent from node i is received at node j

If we denote the local state of a node by LS_i, the state of a channel between two nodes by SC_{ij} and the number of nodes by n, then we can define the global state of the system as [5]:

$$GS = \{\bigcup_{i}^{n} LS_i, \bigcup_{i,j}^{n} SC_{ij}\}, \tag{1}$$

otherwise said, the global state of the system is the set that contains each node's local state and the states of the channels that link those nodes.

We say that the global state of a distributed system is consistent iff for each message the following two conditions are met:

$$send(m_{ij}) \in LS_i \longrightarrow recv(m_{ij}) \in LS_j \oplus m_{ij} \in SC_{ij}, \tag{2}$$

$$send(m_{ij}) \notin LS_i \longrightarrow recv(m_{ij}) \notin LS_j \wedge m_{ij} \notin SC_{ij}. \tag{3}$$

The above conditions require that (1) any sent message has to be either on the channel between nodes i and j, either in node's j state, but not both at the same time and (2) there is no message received by node j from node i or on the channel between them that was not sent by node i.

The reason for that is very simple: if a message sent by node i appears both in node j's local state and in SC_{ij} channel's state, the global state of the system contains message m twice, being therefore *inconsistent*. On the other hand, if a message does not appear as sent by node i (or state LS_i contains message m_{ij}, and the message was not yet transmitted), and at the same time the message appears on the channel or received by node j, that message appears in the global state twice making again the global state inconsistent.

In [5], the author splits the snapshotting problem in two sub-problems which need to be solved in order capture a consistent snapshot:

- distinguish the messages that should be included in the snapshot from those that should not be included in the snapshot;
- find the moment when each node should record its state.

2.2. Snapshot algorithms

The first algorithm, proposed by Chandy and Lamport, [1] required the communication channels between nodes to be FIFOs. The algorithm used marker messages to separate the messages included in the snapshot from the other messages, and the state of a node was recorded right before receiving the marker message. However when using NoCs the above assumption is not available anymore. Flits are delivered out of order making therefore the communication channels non-FIFO.

Therefore, Lai and Yang proposed an algorithm [6] that was able to capture a snapshot in distributed systems that did not require channels between nodes to be

FIFO. Their method used two colors for the messages (*white* and *red*) to distinguish between the messages that should be included in the snapshot and those that shouldn't. Besides this, the algorithm needed local memories at each node to save all the messages sent and received by all nodes. The saved messages were further used to recompute the state of the channels.

As an improvement to Lai's algorithm, Li et. al [7] used tags instead of colors to separate messages, making it possible to capture multiple snapshots. Additionally, Li's algorithm used less memory since his algorithm only saved the messages between two snapshots.

An algorithm that does not need any memory at all and also allows an unlimited number of snapshots was proposed by Mattern [8]. In order to solve the problem of multiple snapshots, this algorithm used three colors to separate the messages from the last snapshot, the current snapshot and the next snapshot. The problem of computing the channels state was solved by keeping counters at every node with the difference between the number of messages sent and received. When a snapshot is taken, the node sends along with its state the recorded counter. The initiator computes then the sum of all counters which is equal to the number of messages in transit (on channels) during this snapshot. Since all messages that have to be included in the snapshot have the same color, the nodes can identify those messages and send a copy of them to the initiator. The initiator knows how many messages are in transit and can detect when all messages in transit were received and the snapshot capture ended.

However in our research we noticed that most of the snapshot algorithms were only theoretically stated. We did not find any study or comparison of any hardware implementations of these algorithms. Therefore, in this paper we chose to implement three algorithms: Lai-Yang, Li et. al and Mattern. These algorithms were used since they do not require the communication channels to be FIFOs, and therefore can be used in a NoC without implementing a transport layer that reorders out-of-order flits.

2.3. Other theoretically possible solutions

One would, however, say that it is not important to use a NoC as an underlying structure for a biological simulation system, because in a biological system, usually only neighbours exchange data and there is no need to send data to entities other than neighbours. In other words, we could implement only FIFO's between every two neighbours and therefore also use FIFO snapshot algorithms. Although it would be interesting to see, how FIFO snapshot algorithms perform comparing to non-FIFO algorithms, we didn't choose to do that, because this model would not scale well, especially if a cell has many neighbours.

Another possible technical solution that needed to be analyzed was using the FPGA-specific configuration readback mechanism (called ICAP for Xilinx FPGAs) to obtain a snapshot of the biological simulation system. In this case, we found several arguments that made us find this method unsuitable for our purpose. First of all, in order to read the state of a logical resource through ICAP, logic has to be read on a per-column basis [9]. This introduces a significant overhead, as we read much more than we need. Secondly, in order to read the state of a logical resource through ICAP,

we have to first stop the clock [9]. This assumes that all the snapshotted elements share the same clock. We might have a single shared clock in some cases, but not anymore when the nodes are in different clock domains, or even on different FPGAs. This can happen, because: (1) the biological systems are heterogeneous by nature, therefore the frequency at wich the various groups of cells communicate differs, and (2) depending on the complexity a biological system might need to be implemented on more than one FPGA. Thirdly, in order to protect the intellectual property, the ICAP capability might be disabled as a security measure (post-configuration anti-tamper policy) to prevent reading the content of the FPGA chip. Other manufacturers like ALTERA don't even provide a configuration readback mechanism for their FPGA chips.

3. Proposed Working Setup

The setup we used in order to implement the three algorithms stated above is highlighted in Fig. 2.

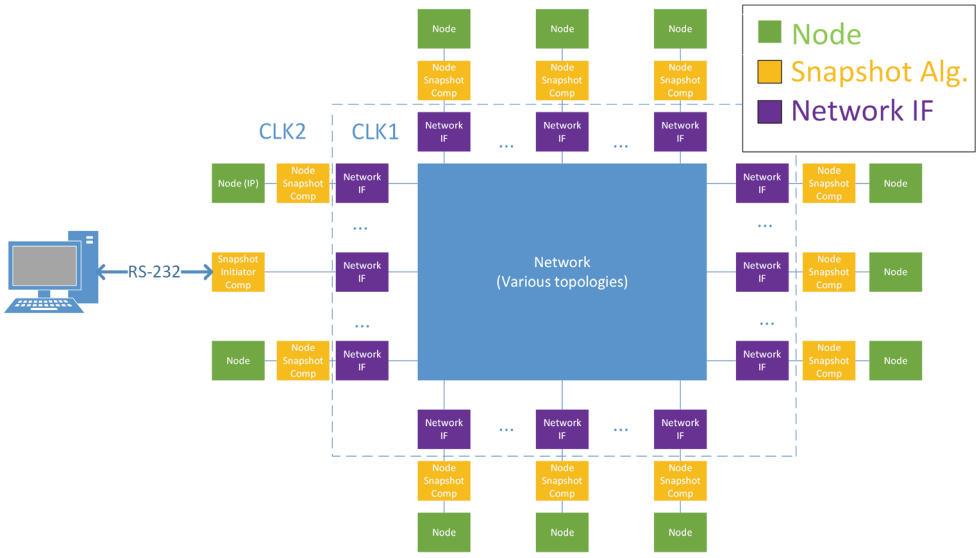

Fig. 2. The general architecture of the proposed setup.

To implement the interconnecting network we used CONNECT [10] which is a NoC generator built by researchers from Carnegie Mellon University. The developers of CONNECT have also built a web interface through which users can generate networks of different topologies, sizes, variable flit width, variable number of virtual channels, different flow control mechanisms, flexible configurable routing, etc. As can be seen in Fig. 2 the network has the same interface regarding the topology that is used inside the component. The generated network is described in entirely synthesizable Verilog code. Additionally the user can build its own topologies through the network editor available together with the network interface or through simple configuration

files. The reason for choosing CONNECT was because of the small amount of logic resources the generated NoC occupied on the FPGA chip compared to the NoCs generated by other NoC generators.

On the next layer, we have implemented an interface with the network whose purpose was to separate the end points from the network and also to offer a simplified interface to the upper levels. This way, the snapshot algorithms are not necessarily coupled to the CONNECT NoC architecture. They could be actually used in any intercommunication network, even a bus-based one, as long as the network interface is re-implemented and the messages are flushed at write. If the bus supports bursts too, the problem gets slightly more complicated, since a message marked as sent is not necessarily sent. However, the snapshotting will still work if the burst control is implemented above the snapshot algorithm. At the network interface level, flow control (Fig. 3) had to be implemented and we choosed to use credit-based flow control due to its superior performance. We also used two virtual channels, one being responsible for the application, and the other one for snapshot messages. The network interface was built to operate at the same frequency as the routers that build the NoC component.

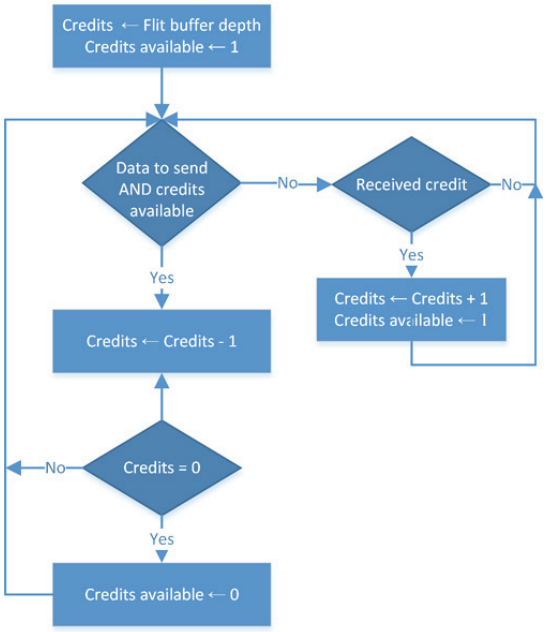

Fig. 3. Flow control at Network IF level.

Immediately above comes the snapshot algorithm. As it can be seen in Fig. 2, the snapshot does not necessarily operate at the same frequency as the network interface. Instead it operates at the frequency of the end point. The snapshot algorithm is separated in two components: a snapshot initiator (Fig. 4) and a snapshot node each component being implemented as a state machine. The initiator, iterates through all

addresses in the network (each node has an address) and sends to each one of them a GET_STATE message. Of course, the initiator doesn't send a message to itself, but it could ignore other nodes too, if needed. After having sent all the messages, it waits for $STATE$, $COUNTER$ and $TRANSIT$ messages from the nodes. The snapshot ends when all the messages have been received, i.e. is when the sum of the received counters equals the number of received transit messages. The structure of a flit when using the Mattern algorithm can be seen in Fig. 5. It is important to note here, that the flit structure varies between the algorithms, depending on what control information it has to hold. By example, Lai-Yang's algorithm uses less control bits (1 for color and 1 for type), therefore having more *state* bits, but it can't capture multiple snapshots. To have more *state* bits, one must either increase the flit size, or use at the end-point level a protocol to fragment and reassemble flits.

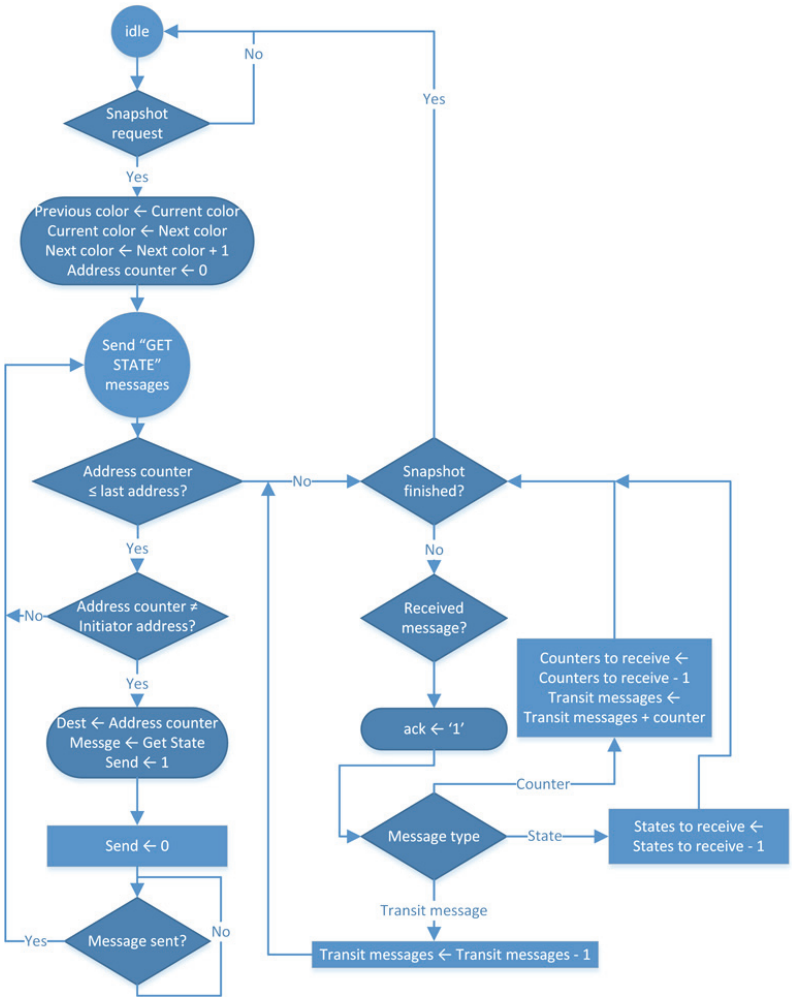

Fig. 4. ASM Chart of Mattern initiator.

The snapshot node stays between the network interface and any end-point and identifies when a snapshot request was made. In that case, its role is to record the state of the end-point and send this information along with any other information specific to the snapshot algorithm to the initiator. Because the snapshot node is separated from the end-point, it is possible to capture the state of any end-point. The endpoint doesn't even need to know that a snapshot algorithm is running and it works in the same manner with any snapshot algorithm. However the end-point has to make the state available to the other components in the system through *state* pins, and to implement the communication protocol (Fig. 6) with the snapshot node. If this is not possible, an adapter block can be created to adapt the interface of the end-point to the interface of the snapshot node at the expense of an extra layer.

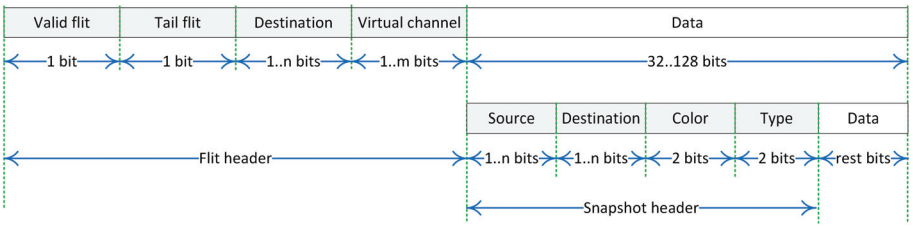

Fig. 5. The structure of a mattern flit.

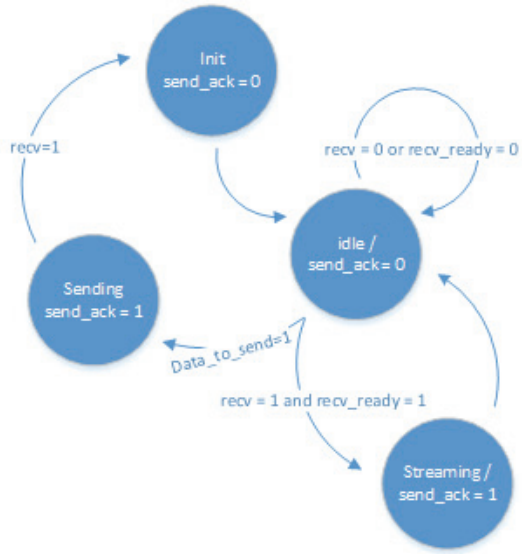

Fig. 6. Communication protocol between levels.

The communication protocol (Fig. 6) between the snapshot node and the end-point (Fig. 7) uses 3 *pins* (one for send and acknowledge signals, one for the receive signal, and one for flow control). This protocol is used between all levels in the hierarchy making it possible to remove a layer (e.g. the snapshot layer) when needed. The end-point is the place where the behavior of a biological level of abstraction (molecule,

cell, organ) can be simulated. The snapshot initiator is fixed, and this component is the one who has the role of initiating a snapshot, collecting the states from the nodes, reassemble them and send the full state to the PC. The protocol used to send the states to the PC is UART. All the layers, (except the NoC component which was generated using the CONNECT tool) were implemented in VHDL language. On the PC a small C# console application was implemented to prove the functionality of the snapshot algorithm.

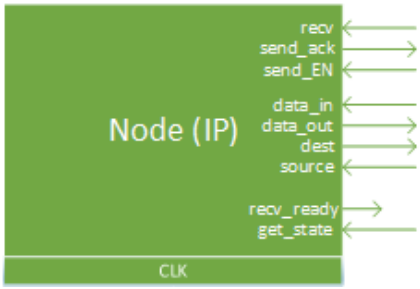

Fig. 7. Blackbox of an end-point.

4. Technical Results

In order to compare the three proposed algorithms we measured the following three performance indicators: throughput without taking a snapshot, throughput during a snapshot, resource usage and operating frequency. All the testing was done using the Xilinx Kintex-7 KC705 evaulation kit that comes with a XC7K325T-2FFG900C FPGA. As a comparison element we used a 16 nodes, mesh network (Fig. 8) with 32-bit wide channels from which 11 bits were used by the snapshot algorithm to store information as source or destination address, color or tag of the message, or other algorithm-specific information. It is important to state here, that we tested the algorithms with networks of different topologies with sizes of up to 128 nodes, but we had to choose a 16 nodes network for comparative testing due to the limitations of Lai-Yang and Li et al. algorithms.

The throughput when a snapshot is not taken was computed theoretically as the maximum messages that can be sent in a second in the network. It can be noticed in Table 1 that when using algorithms Lai-Yang and Li, even when no snapshot is taken, the throughput is very small. This happens because algorithms Lai-Yang and Li need memories to store the messages sent and received between nodes. This is problematic since the memory available on a FPGA is small. For example at a throughput of 34 MB/s it would suffice 1 second to fill the whole BRAM of a Kintex 7 FPGA. Therefore the rate at which nodes can send messages had to be limited for these two algorithms. On the other hand, Mattern's algorithm does not suffer from this disadvantage because it does not need to save any messages at all. Thus, it is able to support much higher throughputs than the former algorithms.

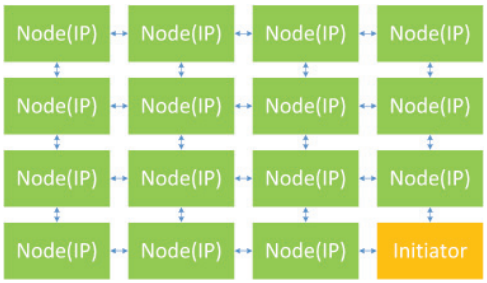

Fig. 8. Snapshotting in a mesh topology. The initiator is placed in the bottom-right corner. Note that moving the initiator position might slightly increase the performance during a snapshot, but decrease the performance when not taking a snapshot.

Table 1. Throughput with and w/o snapshotting

Algorithm	Throughput w/o taking a snapshot	Throughput while taking a snapshot	Overhead
Lai-Yang	30 KB/s	0 KB/s	100%
Li et. al	30 KB/s	0 KB/s	100%
Mattern	600 MB/s	280 MB/s	53%

In order to measure the throughput during a snapshot, we implemented a test scenario where each node sent messages continuously towards all other nodes in the network. Then, we chose the state of each node to be the number of received messages. Afterwards we started capturing consecutive snapshots to measure the number of messages sent during a snapshot. We then compared the results to the throughput when a snapshot is not taken and measured the overhead induced by the snapshot algorithm. The results in Table 1 show that during a snapshot is taken using algorithms Lai-Yang and Li, the whole network capacity is used by the snapshot algorithm and nodes are blocked until the snapshot ends. In contrast, when using Mattern's algorithm, nodes are still able to communicate even when a snapshot is taken but at a lower rate.

The throughput was, however, also limited by the operating frequency of the end-points. We noticed that if the nodes send messages at a very high rate, bringing the network to a 100% load, the network (the NoC component) stops working. This is why we had to reduce the operating frequency of the end-points in order to make sure that they can send messages continuously to all other nodes. A scenario like this where nodes would continuously send messages, could theoretically happen in a biological, even if just for short periods. Table 2 shows the operating frequencies for a few network configurations when using Mattern's algorithm. An improvement that could be added here would be to use DCM to dynamically control the operating frequency in the following way: lower the frequency while a snapshot is taken, increase the frequency when no snapshot is taken.

Table 2. Nodes operating frequency for a 100% sustained traffic when using Mattern's snapshot algorithm

Topology	Nodes	Nodes operating frequency
Ring	16	7 MHz
Line	16	10 MHz
FatTree	16	10 MHz
Mesh	16	10 MHz
Ring	64	625 KHz
Line	100	625 KHz
Ring	128	< 100 KHz

Afterwards, the resource usage of each algorithm was measured by separately synthesizing the snapshot initiator and the snapshot node and writing the number of LUTs and FFs each occupied. The results for the initiators can be found in Table 3. It can be seen that Mattern's algorithm surpasses the other two at resource usage too. This happens because Mattern's algorithm does not need memory to store sent and received messages. The results for the snapshot nodes are similar excepting the fact that the snapshot nodes for Lai-Yang and Li et. al algorithms also use 128 BRAMs, while the snapshot nodes for Mattern algorithm don't.

Fig. 9. Blackbox of an initiator.

Table 3. Resource usage results

Algorithm	Snapshot initiator		Snapshot node	
	LUTs	FFs	LUTs	FFs
Lai-Yang	5,729	1,504	338	193
Li et. al	4,285	1,519	334	181
Mattern	396	279	334	181

5. Conclusion

This paper compares the performance of the FPGA-based implementations of three snapshot algorithms suitable for NoCs adapted to biological simulations that as far as we know were not implemented in hardware: Lai-Yang, Li et. al and Mattern. Although these algorithms are known for quite some time, they have only been used in distributed computing and software simulations of NoCs. This paper presents a comparative study and implementation of the three algorithms in hardware (FPGAs),

showing their direct applicability in the field of biological simulations. A generic infrastructure has been developed to allow a fast and automatic implementation of snapshotting for any network-on-chip; in addition, this infrastructure can be used as a powerful debugging tool for NoCs. The technical results show that the best of these three algorithms is Mattern. It was surprising to see that although Mattern proposed this algorithm as an alternative to Lai-Yang's and Li's, with the purpose of reducing the necessary memory, the final conclusion shows that removing the memory brought also to reducing communication overhead needed in order to capture a snapshot. The synthesis results show also that the usage overhead added by the snapshot algorithm is not significant.

We noticed however that at this point a big bottleneck in the actual setup is UART transmission which is very slow. We would like therefore, as a future work, to replace this transmission with Ethernet transmission, improvement that would allow the algorithm to finish faster. Besides this, it would be interesting to see if using pipeline techniques would improve the performance of the algorithm, because at this moment the algorithm was implemented as a FSM. As we stated in the introduction, this setup could be used for simulating biological processes on a FPGA. Therefore, we look forward into using these framework into speeding-up the mentioned simulations. Even though the biggest network we could synthesize was of only 128 nodes, it is possible to create a network of NoCs located on multiple FPGAs interconnected using MGTs as in [11] in order to simulate bigger processes like the C. elegans worm (Fig. 1), [12] (artificial life) or [13] (biological simulations).

References

[1] CHANDY K. M., LAMPORT L., *Distributed snapshots: determining global states of distributed systems*, ACM Transactions on Computer Systems (TOCS), Vol. **4**, No. 1, pp. 63–75, 1985.

[2] VERMEULEN B., GOOSSENS K., *Obtaining consistent global state dumps to interactively debug systems on chip with multiple clocks*, High Level Design Validation and Test Workshop (HLDVT), 2010 IEEE International, pp. 1–8, 2010.

[3] POLITOPOULOS I., *Review and analysis of agent-based models in biology*, University of Liverpool, Technical Report, 2007.

[4] BARNES D. J., CHU D., *Introduction to modeling for biosciences*, 2010, Springer.

[5] KSHEMKALYANI A. D., RAYNAL M., SINGHAL M., *An introduction to snapshot algorithms in distributed computing*, Distributed systems engineering, Vol. **2**, No. 4, p. 224, 1995.

[6] LAI T. H., YANG T. H., *On distributed snapshots*, Information Processing Letters, Vol. **25**, No. 3, pp. 153–158, 1987.

[7] LI H. F., RADHAKRISHNAN T., VENKATESH K., *Global State Detection in Non-FIFO Networks*, ICDCS, pp. 364–370, 1987.

[8] MATTERN F., *Efficient algorithms for distributed snapshots and global virtual time approximation*, Journal of Parallel and Distributed Computing, Vol. **18**, No. 4, pp. 423–434, 1993.

[9] LogiCORE IP XPS HWICAP (v5.01a) Product Specification, Xilinx, June 2011, Rev. 1.7.1 [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap/v5_01_a/xps_hwicap.pdf

[10] PAPAMICHAEL M., HOE J. C., *CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs*, FPGA, 2012, pp. 37–46. [Online]. Available: http://dblp.uni-trier.de/db/conf/fpga/fpga2012.html

[11] CHIRAP C.-T., CREŢ O., *A flexible communication method for multi-FPGA based designs*, Proceedings of the 10th FPGAworld Conference, ser. FPGAworld 13. New York, NY, USA: ACM, 2013, pp. 4:1-4:6. [Online]. Available: http://doi.acm.org/10.1145/2513683.2513687

[12] NEDEZKI C. JULEAN, D., KEREKES G., *Study of the workspace for parallel manipulator 3RUU*, Annals of DAAAM & Proceedings, 2009.

[13] DARABANT L., CRETU M., DARABANT A., *Magnetic stimulation of the spinal cord: Experimental results and simulations*, IEEE transactions on magnetics, Vol. **49**, No. 5, pp. 1845-1848, 2013.