BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 64 (68), Numărul 1, 2018
Secția
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

A MODULAR SIMULATION FRAMEWORK FOR NETWORK-ON-CHIP SYSTEMS USING PYTHON

BY

PAVEL IONUȚ CĂTĂLIN[*] and VASILE MANTA

"Gheorghe Asachi" Technical University of Iaşi
Faculty of Automatic Control and Computer Engineering

Received: February 19, 2018
Accepted for publication: March 23, 2018

Abstract. The need to incorporate more features on silicon chips requires new means to evaluate the system behaviour. Simulation tools become indispensable in analyzing performance during the design phase. A closer look at the available tools reveals some existing solutions that are unfortunately hard to deploy and adapt for all the situations encountered. This paper proposes a new tool to aid in the simulation of such systems, in particular systems that are using Network-on-Chip technologies. The tool is written in Python and it offers a framework to analyze different aspects involved in the design of a Network-on-Chip, such as node placement and routing algorithms. The proposed framework allows adding more features on the way due to a modular design approach.

Key words: System on Chip; Network on Chip; Python; Simulation; Router; Port; Multi-core; Many-core.

2010 Mathematics Subject Classification: 68N01, 68Q99.

1. Introduction

1.1. *Insight Into the Topic*

Since the need for miniaturization, increased processing power and efficiency in modern computational devices, including phones, tablets and desktop computers is ever growing, more and more ways to achieve and sustain these demands are constantly developed. One of these methods consists in the

[*]Corresponding author: *e-mail*: ionut.catalin.pavel@webmail.tuiasi.ro

integration of all the computing subsystems on the same chip (memory, I/O, special adapters, etc...), the resulting product being known as a "System on Chip" (SoC). Since more and more dedicated blocks are required in these devices, blocks that include many microprocessors, graphics processors, various memories, sound processors and others, the usual way of making interconnections between these blocks, which consisted of bus based interconnection and crossbar switches, fall short when scalability and future improvements come into play. Also when we are dealing with billion transistor chips, it becomes impossible to send signals across the chip within real time bounds (Jantsch & Tenhunen, 2003). If we use a global clock signal to synchronize different signals we become more prone to electromagnetic interference (EMI). The traditional way was to design critical paths and clock distribution trees (Kumar, et al., 2002). This makes the current implementations power hungry and difficult to manage due to clock skew issues (Arteris, 2005).

The solution for these scalability problems comes in the adaptation of network systems to work on a small scale, and connect the various blocks inside a SoC. These networks also bear the name "Network on Chip" (NoC). However, with the adaptation of such technologies, new and challenging problems come into play, problems as network topology and routing algorithms.

1.2. *Network on Chip structure*

In a NoC interconnection, units are connected via a scalable homogenous system. Units communicate with each other using messages (usually called packets). In Fig. 1 we can see a simplified schematic of a NoC type interconnect inside a SoC.

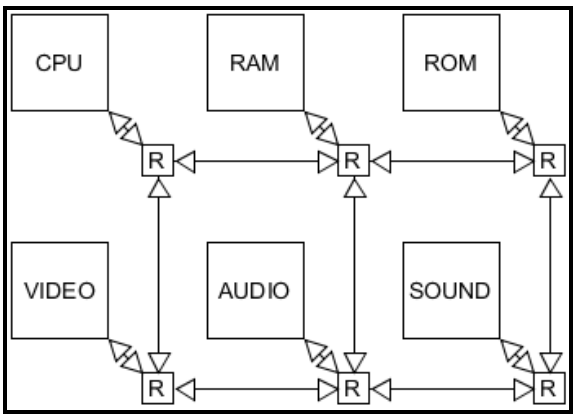

Fig. 1 – Simplified view of a NoC system.

Because in a NoC system there can be more than one simultaneous data path, it is usually required to have dynamic routing and decision making for determining the traffic direction. One key feature of these kinds of systems is

the scalability and the layering. The scalability ensures compliance with the changes in the silicon manufacturing processes and the layering assures a higher level of abstraction between different parts of the network protocols. These systems are basically a scaled down version of wired computer networks, and usually employ the same terminology for describing various parts of the network such as:

1º Routers: Redirect the incoming packets based on a set of rules that are dependent on the network topology.

2º Ports: Are attached to the router and usually contain buffers to aid in the packet transmission scheme.

3º Links: Are the physical wires that make the connection between ports.

4º Processing Units: These are the actual masters or slaves present inside the SoC. These are also attached to the Routers using some sort of local connection scheme that may be the same as in the general case.

1.3. *Comparison Between Classical Bus and NoC Systems*

In Table 1 we can see the main benefits and downfalls regarding the 2 technologies (Bjerregaard & Mahadevan, 2006).

Table 1
Benefits and Downfalls

Bus		NoC	
Every unit adds parasitic capacitance, therefore electrical performance degrades with growth.	-	+	Performance is not degraded with scaling.
Bus timing becomes difficult as the feature size decreases.	-	+	Links can be pipelined.
Bus arbitration can become a bottleneck. If there is more than one master, the others must wait for the active one to complete its transaction.	-	+	Distributed routing.
The bus arbiter is specific for the instantiated block.	-	+	The same router can be re instantiated in other part of the network.
Testability is slow.	-	+	Locally placed boundary test and scan is fast and offers good coverage.
Bandwidth is shared by all the units attached.	-	+	Bandwidth scales with network size.
Latency is the same as the wire speed once control is taken by the bus master.	+	-	Internal network congestion may cause latencies.
Compatibility with a vast amount of existing IPs.	+	-	Bus oriented IPs need wrappers.
Simple concept.	+	-	Needs reeducation.

To facilitate the development of such systems, several dedicated analysis and simulation tools are available. These are often developed by the scientific community. The industry also participates in the development of such

tools. By classification there are usually 2 types of tools used for this task: (*i*) *synthesizers* and (*ii*) *simulators*. Synthesizers are used to generate the hardware system given a set of design constraints or rules. These are usually commercial solutions and are used to model all the stages in the design of a SoC. These are usually characterized by the performance of the output they give (gate count, energy consumption) and the level of abstraction required in the design. The higher the level of abstraction, the easier the design task becomes. Some commercial synthesizers are: "*FlexNOC*" from Arteris (Arteris), "*INOC*" (iNoCs) and "*The Tool Suite Works CHAIN*" from Silistix.

Simulators are usually used to estimate the system performance before actual design. These estimations often include power consumption estimations, gate count estimations, routing algorithm performance and others. Because these systems usually have very restrictive constraints, simulation is a must in the design of NoCs. More information about the available simulators can be observed in Table 2 (Achballah & Saoud, 2013).

Table 2
Available Simulators

Tool	Year	Developer
NS-2	1995	DARPA and later Contributors
Noxim	2010	Catagne University
DARSIM	2009	MIT
SunFloor – 3D	2006 - 09	EPFL (swizerland)
ORION 1 and 2	2003 - 09	Princeton University
INSEE	2005	Basque University (Spain)
ATLAS	2005	Federal University of Brazil
NOCIC	2004	Massachussetts University
Pestannna Environment	2004	Phillips Research Laboratories
PIRATE	2004	Polytechnique School of Milan
SUNMAP	2004	Stanford University
xpipesCompiler	2004	Bologne University – Stanford University
µSpider	2004	Bretagne Sud University
OCCN	2004	ST microelectronics
NoCGEN	2004	University of New South Wales
FlexNoC	–	ARTERIS
iNoC	–	iNoCs
The CHAIN works tool suite	–	Silistix

2. A NoC Simulation Framework Written in Python

2.1. *Motivation*

Since NoCs are fairly complex, pre design analysis is a must, and, as explained in the previous chapters, there are many topics that could have a great

effect on the performance and the potential of the network. Simulating the network pre design has become the norm. The simulation can be accomplished using various tools, as the ones presented above, however most of those tools lack some certain features and prove hard to reconfigure to support new features. They also are written in various programming languages that require a greater deal of knowledge in using, and they usually have hard to fix dependency issues. Because of this, in the following pages a new model for a simulation tool is proposed. The tool is completely written in Python without any external dependencies, the only prerequisite is having a working version of the Python interpreter installed. This also facilitates multi platform development since Python code is portable across platforms. At the current moment, the tool supports only a small subset of the NoC methodologies, mostly deterministic routing, dynamic routing and packet routing.

The whole idea is to provide a scalable framework that is easy to extend to various architectures and methodologies while providing a basic platform for the most common used elements. The simulation is at a basic implementation state, only some basic simulations could be performed. The features supported are the following:

a) single clock domain;
b) multi network simulations;
c) various topologies (created by the developer);
d) various routing algorithms (created by the developer);
e) processing element simulation (created by the developer);
f) multiple router types;
g) multiple routing algorithms (provided by the developer).

2.2. *Test Case and Simulation Results*

After the base framework realization, a small test script was required to proof the functionality of the simulator. To realize this test, a network had to be defined. In Fig. 2, the proposed network and the used routing algorithm can be seen. The algorithm routes the packets based on the coordinates inside the router matrix. At first a delta is performed and the packet is moved in the direction that decreases this delta value. This type of algorithm is known as the XY algorithm. After the network was created, it was simulated for a number of 10 clock cycles, exchanging packets between the MEM and the CPU node.

In Table 3 we can observe a list of events that took place during the 10 cycles of the simulation. We can observe the latency introduced by the network. The delay is exactly 6 cycles which corresponds with the number of links the message has to go from the source port to the destination port. For better visibility, the nodes send different positive and negative integers.

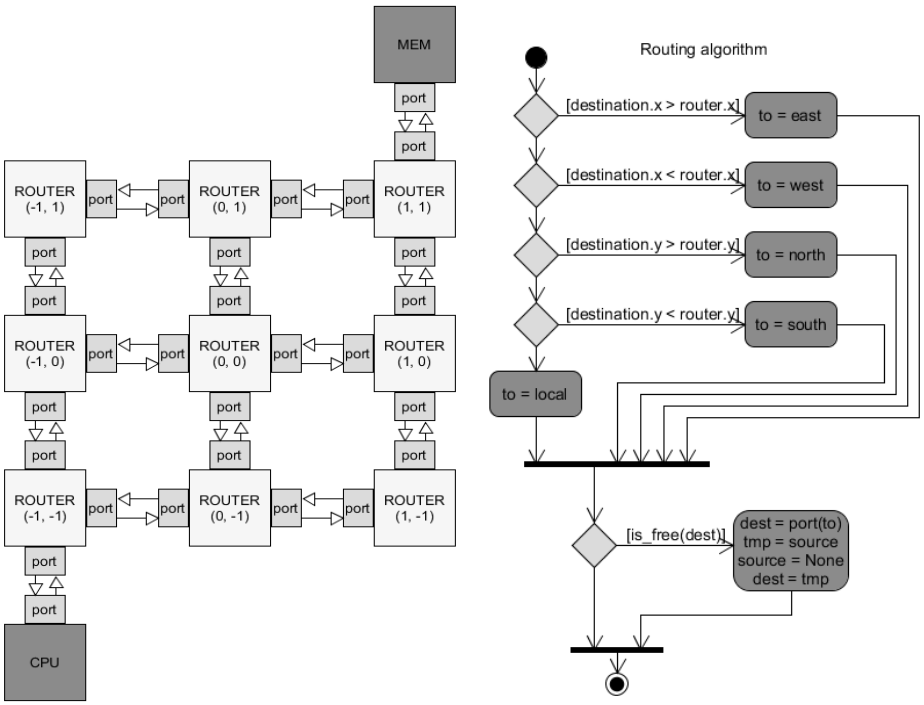

Fig. 2 – Test network and routing algorithm.

Table 3
Simulation Results

Network cycle	Event 1	Event 2	Event 3	Event 4
1	CPU sends to MEM value 0	MEM sends to CPU value 0		
2	CPU sends to MEM value 1	MEM sends to CPU value -1		
3	CPU sends to MEM value 2	MEM sends to CPU value -2		
4	CPU sends to MEM value 3	MEM sends to CPU value -3		
5	CPU sends to MEM value 4	MEM sends to CPU value -4		
6	CPU sends to MEM value 5	MEM sends to CPU value -5		
7	CPU receives from MEM value 0	CPU sends to MEM value 6	MEM receives from CPU value 0	MEM sends to CPU value -6
8	CPU receives from MEM value -1	CPU sends to MEM value 7	MEM receives from CPU value 1	MEM sends to CPU value -7
9	CPU receives from MEM value -2	CPU sends to MEM value 8	MEM receives from CPU value 2	MEM sends to CPU value -8
10	CPU receives from MEM value 3	CPU sends to MEM value 9	MEM receives from CPU id 3	MEM sends to CPU id 9

3. Conclusion

Although the simulation is still in the development phase, the partial results obtained in the simple test simulation show that the idea can be expanded to many situations encountered in the development phases of the system. The usage of a popular and relatively easy to learn programming language as Python, makes the implementation of new features very easy because of the modular design, and removes any external module dependency issues that are present in other simulators. More features and improvements are planned for the future, some of these will include, but not limited to: *time based simulation*, *predefined collection of routing algorithms*, *network generator based on classic topologies* and *a better event logger*.

REFERENCES

Achballah A.B., Saoud S.B., *A Survey of Network-On-Chip Tools*, International Journal of Advanced Computer Science and Applications, Vol. **4**, No. *9* (2013).

Bjerregaard T., Mahadevan S., *A Survey of Research and Practices of Network-on-Chip*, ACM Computing Surveys, 2006.

Jantsch A., Tenhunen H., *Network on Chips*, Kluwer Academic Publishers, Boston, 2003.

Kumar S., Jantsch A., Soininen J.P., Forsell M., Millberg M., Oberg J. et al., *A Network on Chip Architecture and Design Methodology*, IEEE Computer, 117-124, 2002.

* * * *A Comparison of Network-on-Chip and Buses*, Arteris, 2005.

* * * Arteris. (n.d.). Retrieved November 15, 2016, from www.arteris.com: http://www.arteris.com/flexnoc

* * * iNoCs. (n.d.). *www.inocs.com*. Retrieved November 15, 2016, from www.inocs.com

UN ECOSISTEM MODULAR PENTRU SIMULAREA REȚELELOR INTEGRATE FOLOSIND LIMBAJUL DE PROGRAMARE PYTHON

(Rezumat)

Această lucrare prezintă un ecosistem software pentru simularea sistemelor de interconexiuni pe chip de tipul Noc (Network-on-Chip). Acest simulator este creat folosing limbajul de programare Python. Acest lucru vine ca o alternativa la sisteme de simulare deja existente, dar care sunt afectate de anumite probleme, în special legate de disponibilitate şi de funcțiile suportate. Sistemul propus este uşor modificabil având în vedere structura modulară pe care este construit.

In finalul lucrării este prezentat un exemplu simplu de simulare obținut cu ajutorul utilitarului. Rezultatele obținute încurajeaza dezvoltarea in continuare şi adaugarea de noi funcționalități.