IPNoSys III:SDN Paradigm in a non-conventional NoC-based Processor

IPNoSys III: O Paradigma SDN em uma NoC Baseada em um Processador Não Convencional

DOI:10.34117/bjdv7n1-159

Recebimento dos originais: 10/12/2020
Aceitação para publicação: 09/01/2021

Denis Freire Lopes Nunes
Mestrado
Universidade Federal do Rio Grande do Norte
Campus Universitário, Lagoa Nova, Natal/RN
e-mail: denisfreire@ppgsc.ufrn.br

Sílvio Roberto Fernandes de Araújo
Doutorado
Universidade Federal Rural do Semi-Árido
Av. Francisco Mota, 572, Costa e Silva, Mossoró/RN | Campus Leste - CCEN - Sala 35
e-mail: silvio@ufersa.edu.br

Márcio Eduardo Kreutz
Doutorado
Universidade Federal do Rio Grande do Norte
Campus Universitário Lagoa Nova, Natal/RN
e-mail: kreutz@dimap.ufrn.br

ABSTRACT

Dynamic resource allocation has a significant impact on the performance of MPSoCs (Multiprocessors System-on-Chip) based on Networks-on-Chip (NoCs). In this work, we propose the IPNoSys III, an NoC using Software Defined Networks (SDN) paradigm applied to IPNoSys, a parallel non-conventional architecture. IPNoSys III has a 2D mesh topology, that contains in each node four processing cores, connected to a memory and that run packages in the IPNoSys format, and a communication unit. An SDN controller, connected to all nodes, manages the network and has an overview of the network to execute the routing algorithm and to map tasks according to the performance objectives. The results show up to 17% better performance in clock cycles to the SDN controller than a static solution and up to 46% better when comparing IPNoSys III to a conventional NoC.

Keywords: IPNoSys, NoC, MPSoCs, SDNoCs.

RESUMO

A alocação dinâmica de recursos tem um impacto significativo no desempenho de MPSoCs (Multiprocessors System-on-Chip) baseados em Networks-on-Chip (NoCs). Neste trabalho, propomos o IPNoSys III, uma NoC que utiliza o paradigma de Redes Definidas por Software (SDN) aplicado a IPNoSys, uma arquitetura paralela não convencional. A IPNoSys III possui uma topologia de malha 2D, que contém em cada nó quatro núcleos de processamento, que possuem acesso à memória e executam pacotes no formato IPNoSys, e uma unidade de comunicação. Um controlador SDN, conectado a todos os nós, tem uma visão geral e gerencia a rede para executar o algoritmo de roteamento e mapear tarefas de acordo com os objetivos de desempenho. Os resultados mostram um desempenho até 17% melhor em ciclos de clock para a IPNoSys III com o controlador SDN do que com uma solução estática e até 46% melhor quando se compara o IPNoSys III a uma NoC convencional.

Palavras-chave: IPNoSys, NoC, MPSoCs, SDNoCs.

1 INTRODUCTION

Communicating processing cores within MPSoCs is one of the challenges faced by designers. Networks-on-chip (NoCs) integrates high scalability with the possibility of transmitting messages in parallel [1], [2]. With the constant miniaturization of components and the scalability provided by NoCs, MPSoCs appear as state-of-the-art parallel architectures, with chips with dozens or even hundreds of processing cores [3][4]. With many processing elements running different applications, the need to allocate tasks to use the infrastructure better becomes essential to obtain better performance, considering the execution time and energy consumption. In [5], the authors affirm that the complexity in allocating tasks is exponentially proportional to the number of tasks and processing elements available. According to the same authors, poor management of these resources can impair performance and energy consumption by a few tens of magnitude. Thus, MPSoCs need to provide intelligent control of communication and computing infrastructure. This approach allows the architecture to dynamically adapt to the applications, according to the expected objectives (better performance, lower energy consumption, temperature, etc.), with the minimum possible loss in quality-of-service (QoS) [6].

Conventional computer networks also aim to provide efficient communication between the interconnected components and dynamically adjust to the applications' different scenarios, finding the best route for package transmission, traffic balancing, and adaptation in case of the failed link, etc. State-of-the-art works converge towards using the software-defined networking (SDN) paradigm [7]. The concept of SDNs is the separation of control and data layers, that allows the network to overview all nodes and efficiently manage resources [8]. The solutions presented by the SDNs are executed by software, which directly impacts the network's performance because the controller analyzes the packages frequently at run time. However, the better management of the network compensates for the loss of performance, which can be up to 20% more efficient in some cases when compared to networks without SDN [9].

Computer networks were the inspiration of the NoCs. New technologies introduced in networks were also adapted and brought to NoCs, such as wireless routers [10]. Some works such as those by [11] and [12] introduce the SDN paradigm in NoCs. The architectures derived from this paradigm were named SDNoC (software-defined network-on-chip).

The justification for adopting SDN in NoCs is making the architectures support future applications, adapting to the conditions to get better performance [13]. As all management is under a software solution, network management can be updated at runtime and adapts to the applications. With centralized control, routers have their complexity reduced to just forward packages [14]. Although the studies show promising results, some are reaffirming the same conclusions observed in the statistics of conventional SDN networks': the performance improvement is more significant than the losses caused by the software solution.

Integrated Processing NoC System (IPNoSys) is a new model of NoC presented in [15]. This architecture brought a new component that joined processors and routers in a single element called RPU (Processing and Routing Unit). This architecture has a high processing power compared to a conventional MPSoC, mainly in processing applications with high parallelization capacity. The number of published works shows that IPNoSys architecture is gaining prominence in the academic environment. Due to its unique execution model and differentiated architectural components, it is possible to create variations of the architecture to adapt to new scenarios. As an MPSoC, IPNoSys presents a good platform for a parallel single application, including more parallelism was archived with IPNoSys II [16] Thus, in this paper, we propose a new version of this architecture (IPNoSys III) through the SDN paradigm, which allows the execution of many applications keeping high performance and balance control of the entire network.

2 RELATED WORKS

2.1 SDNOCS

Works that directly deal with SDNoCs are quite recent, with the first work published in 2014 by [17]. The presented architecture differs from conventional NoCs with the separation between data and control.

In [12] were the first to deal directly with resource management in SDNoC, which describes a task mapping algorithm that runs on routers. Like other purposes, this SDNoC implements the data and control layers via software, over the infrastructure (physical) layer. In this architecture, the tests carried out in a simulation tool described in SystemC with the network's size varying among 4x4, 5x5 and 6x6. The benchmark was the applications H.263 and MP3 multimedia encoder and decoder. They compared the proposed dynamic mapping with two other strategies: a genetic algorithm and a heuristic. The results showed that SDNoC achieves a better performance with the application's execution time and has a lower mapping cost than the others. However, the SDNoC router has a larger area and power consumption than a conventional router.

In [13], a manycore with a physical topology composed of a 2D mesh, but through software configuration, a dynamic virtual topology mapped node adapts to the application. The logical topology is configured based on the application at the beginning of execution. The authors presented an example of the network being reconfigured for a ring topology in which 37.5% of the links are disconnected, saving energy. Each router has a memory that maintains a routing table. The special instructions added to the ISA (Instruction Set Architecture) do the configuration. To assess performance, they compare the SDNoC to a conventional NoC with mesh topology. Both were executed with a random traffic pattern and varying the network's size between 16 and 1024 PEs (Processing Elements). The results showed that latency in conventional NoC increases at a higher rate than in SDNoC. For 1024 PEs, the latency in conventional NoC is almost double for SDNoC (425 and 220 cycles, respectively). Also, SDNoC latency increases at a more linear rate, proving its better scalability. The paper does not compare area and power with other architectures.

In addition to communication management, the software control can also provide other resources in an NoC as in [18], when it uses to provide security in the communication among cores of the same MPSoC. The authors argue that the SDN paradigm was adopted because the amount of memory needed to store the routing tables grows exponentially as the NoC increases. Using SDN, the switches are now managed by the controller, reducing the amount of memory required.

2.2 IPNOSYS

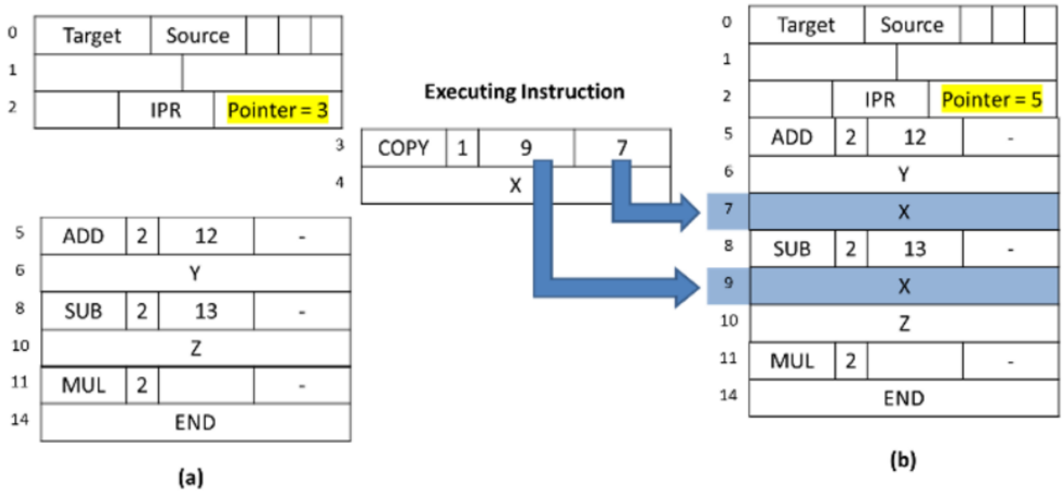
IPNoSys was created after the observation that data transmission between the source and the destination routers, in a conventional NoC, only access the package header to perform the routing. The idea is to take advantage of the package's transmission time by carrying out to process of a program's instructions during the path between the origin and the destination [19]. IPNoSys consists of an interconnection network in the 2D grid format of NxN nodes, XY routing, distributed arbitration, storage at the entrance, credit-based flow control, at least two virtual channels, VCT, and wormhole switching [15]. The RPU (Routing and Processing Unit) replaces the router-processor sets of conventional NoCs and the Memory Access Unit (MAU) handler the memory access.

The execution in IPNoSys happens by injecting packages in the four corners of the network through the MAUs. Each package consists of 36-bit words, the four most significant bits define the type of the word and the other 32 bits carry the actual information. The words can be of four different types: header, instruction, operand, and end-of-package. Each package has header words, a sequence of instructions and operands, and an end-of-package word.

The header words indicate the package's properties, such as origin and destination of the package, number of instructions, type of package, and program to which it belongs. The instruction word contains the OPCODE of the instruction to be executed and the addresses for inserting the operation results. The operand word uses all bits to indicate a value or a memory address, depending on the instruction. The end-of-package word does not carry useful information, only indicating the end of the package. Regular packages carry instructions to be executed within the network by RPUs. Regular packages generate control packages with specific instructions that must be executed by the MAUs.

The IPNoSys assembly language is called Package Description Language (PDL), which includes information about each package, how they are related among them in a program, the conventional programs instructions, furthermore, synchronization instructions to parallelization. Each program can have one or more packages stored in the memories located in the four corners of the network and inserted into the network word by word by the MAUs. As the package enters the network, each RPU in its path executes at least one instruction from the package, removes it, and reinserts the result in the locations indicated by the instruction word. The number of instructions to be executed by each RPU is indicated in the package header and after execution, the rest of the package is transmitted to the following RPUs. Figure 1 details this process.

Figure 1 - IPNoSys Execution Model

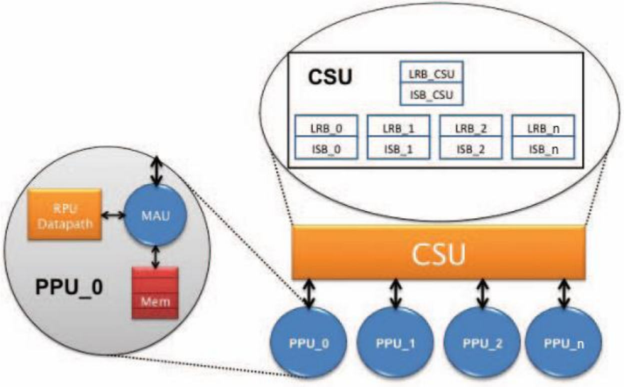


Source: [20]

In Figure 1a, the words 0, 1, and 2 are the header. Words 3 and 4 are the COPY instruction and its operand. Words 5 to 14 are the rest of the instruction and the terminate word in the package. According to the example, the COPY instruction is executing, with the result of the operation going to addresses 7 and 10 of the packages (Figure 1b). After execution, the RPU transmits the rest of the package to the next RPU. This process repeats until all instructions within the package execute. The best IPNoSyS performance is achieved through thread-level parallelism. Thus, an application can be built using packages such as threads that are injected in parallel by the four MAUs.

In [16] the authors developed IPNoSys II, a new architecture that inherits characteristics of IPNoSys. The new architecture maintains the execution model through new components created from the original RPU and MAU modifications. Figure 2 shows the basic architecture diagram.

Figure 2 - IPNoSys II



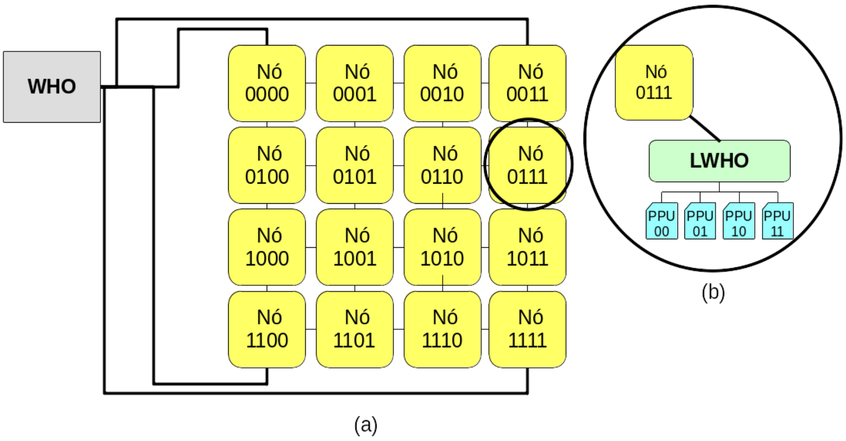
Source: [16]

In the IPNoSys II, the network of RPUs of the version I is replaced by a Packet Processing Unit (PPU), which can execute an entire package without the need to transmit it, thus improving more yet the performance. An IPNoSys II containing four PPUs, linked by CSU (Communication and Synchronization Unit), is equivalent to the version I to execute for parallel threads. The experimental results showed the version II speed up to 5.9 times compared to version I and up to 19.1 times compared to NoC-based MPSoC [16]. Furthermore, the connection among several CSUs is perfectly possible to increase the parallelism and consequently performance.

The PPU works as a "mini" IPNoSys. It can read packages in memory, decode and execute all instructions, send instructions to be executed on remote PPUs (via CSU), and execute instructions sent by remote PPUs. It can also send the processing result back to the PPU that requested execution in the latter case.

In [21], the author analyzes the feasibility of using PPUs as processing elements in MPSoCs and shows a theoretical analysis of performance. The author designed an architecture with square mesh NoC (Figure 3a), in which each node has 4 PPUs and a communication element called LWHO (Figure 3b), similar to CSU of IPNoSys II. A central component, called WHO controls communication between network nodes.

Figure 3 - NoC IPNoSys



Source: [21]

The WHO component allocates tasks in the nodes, creating clusters that execute the applications. The WHO connects only to the corner nodes. All communication between the WHO and other nodes must be retransmitted between the intermediate nodes. Static algorithms that create a cluster for each application manages the NoC. At first, only one node forms each cluster, but it can be added more nodes according to the application's need to instantiate more tasks until there are no more nodes available. LWHO manages the execution of applications within the cluster, providing communication between PPUs and defining which PPU executes each task. All management is done statically, through tables that store the location of tasks in each PPU.

The architecture was simulated in high-level software, to ascertain to validate the allocation task algorithm. Specific characteristics inherent to IPNoSys and other NoCs were abstracted, such as the communication cost and the number of cycles for executing each instruction. The results showed that the architecture scales well even in scenarios where many requests for allocating and deallocating nodes to the clusters.

The new architecture proposed by this paper is called IPNosys III, it expands [21], changing some architecture properties and adding the SDN paradigm to the allocation of tasks.

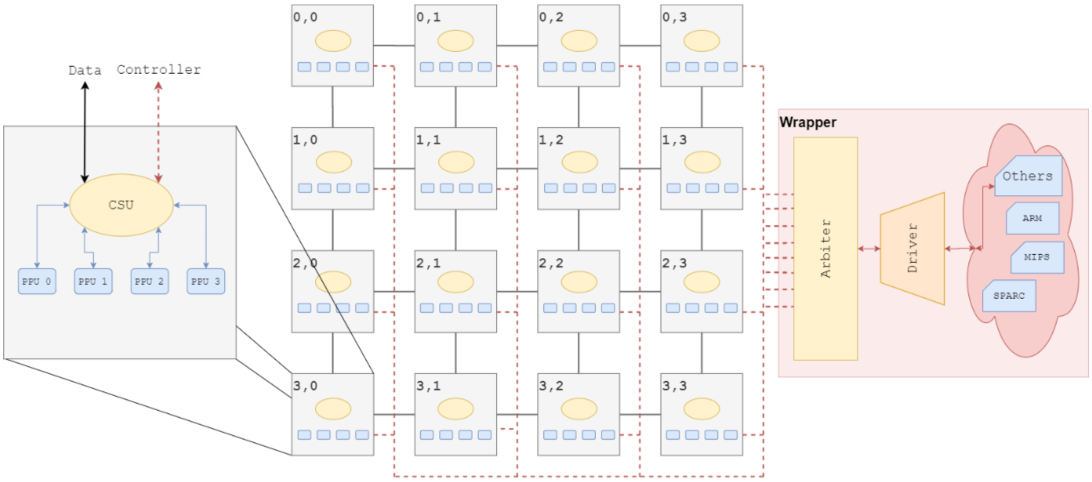
3 IPNOSYS III

The work of [21] shows the feasibility of an NoC using IPNoSys II processing elements. The study was done analytically and showed positive results. Thus, in this paper, we expanded this idea and will present a SystemC RTL implementation with a network central controller with SDN characteristics, it is the IPNoSys III.

3.1 OVERVIEW

IPNoSys III is an NoC with a 2D mesh topology, where each node has a processing element like IPNoSys II, with four PPUs and one CSU, performing up to four concurrent threads. Figure 4 shows the simplified scheme of the proposed architecture. The CSU manages the communication between the PPUs inside the node without the network controller's intervention, while the controller manages the communication between nodes.

Figure 4 - IPNoSys III



The proposed architecture uses SDN (Software-Defined Networking) concepts, adding a controller element that communicates with the nodes through its own protocol. The controller has an overview of the entire network, creates clusters for applications, indicates on which node should instantiate each task and calculate the routes to tasks communications.

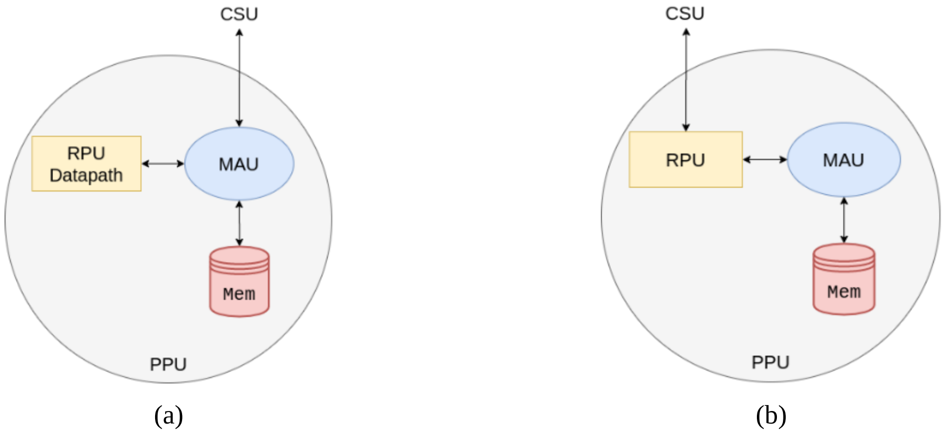
Each PPU has a physical memory that forms a unique address space, equally shared for the entire architecture. For each memory module, an associated MAU is in charge of injecting the packages and executing instructions that read or write data in memory.

Like the previous version, the IPNoSys III uses the original IPNoSys ISA, so the programs run in previous versions are compatible with our proposal, thus maintaining the idea of the IPNoSys processor family. We add only a new control package, with a slightly different format, which carries out communication between the controller and the network components.

3.2 PPU, CSU, AND CONTROLLER

In IPNoSys III we change the interface of PPU to the CSU through RPU instead of MAU, as the previous one, as seen in Figure 5.

Figure 5 - IPNoSys II PPU vs IPNoSys III PPU

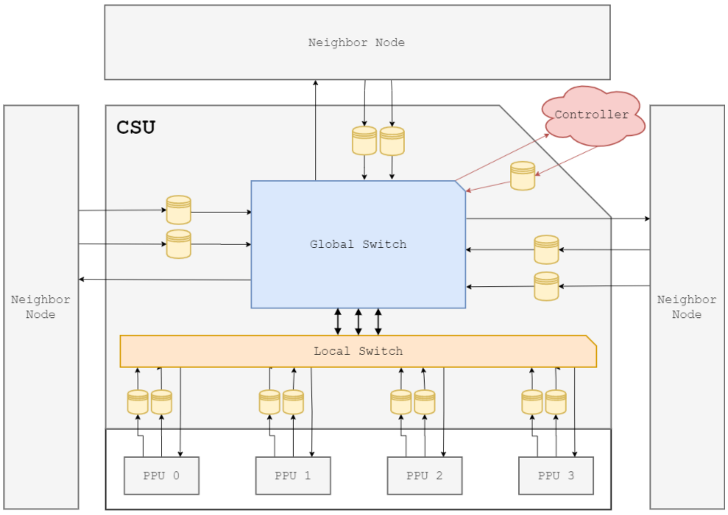


(a) (b)

The choice for changing the internal organization of the PPU is because RPU executes control instructions used for communication between nodes, decreasing the number of jumps to reach the destinations. Only read and write memory instructions need to go through the MAU.

The CSU (Communication and Synchronization Unit) of IPNoSys III, in addition to connecting a node's PPUs, also is connected to the network manager and with the neighboring nodes' CSUs. It consists of two switches, one local (SL), which interconnects all PPU ports, and one global (SG), which connects the SL with the network controller and neighboring nodes, as shown in Figure 6.

Figure 6 - IPNoSys III Comunication and Sinchronization Unit



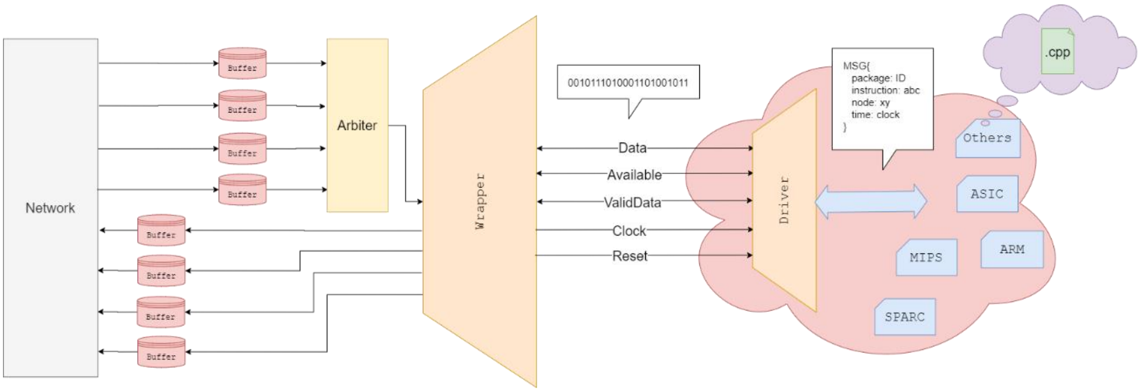
The SL works by connecting the PPUs and the SG via a crossbar, implementing the Round-Robin algorithm to attend to requests of each port. The network controller executes the routing algorithm and CSUs only store the routes, characterizing the SDN paradigm.

According to the SDN paradigm, software that usually runs on one node manages the network. This component can have an overview of the status of all nodes and act to maximize performance according to the conditions pre-established by the programmer. Although this is the standard adopted by the architectures, there are no defined rules or official norms for implementing an SDNoC. As long as some features are implemented, such as the control being done at a high level by some element with an overview of the network, this architecture can be referred to as having SDN characteristics.

Unlike other SDNoCs, IPNoSys III's designed a control to be independent of the network, executing the control software in a separate processor. An interface (wrapper) was implemented, with a communication protocol to connect a network with an ASIC or any other processor such as an ARM, MIPS, SPARC, etc., which will run the control software.

The interface is an arbiter connected to all CSU of the nodes. Each link has a buffer that stores the packages from the CSUs towards the controller and vice versa. The arbiter uses the Round-Robin algorithm to select one of the requested buffers and transmit the package to a wrapper. Figure 7 shows a summary diagram of the controller components for a network with four nodes.

Figure 7 - Diagram of the Controller



The wrapper converts the messages exchanged between the network and the control software between digital signals and high-level data and stores the data in memory to the software can access them. The data arriving at the wrapper over the network are digital signals made up of words (or packages) from IPNoSys. It extracts the instruction, program and package IDs and the operands (if exists) present in the control packages (or signals) and the node to send to the control software can access them.

3.3 EXECUTION MODEL

The IPNoSys programming model divides programs into regular packages that correspond to threads and control packages that correspond communications among network components. In previous versions of IPNoSys, the package traveled between the RPUs and needed to carry some information for the correct communication and processing of its instructions, for example, origin, destination, and which the RPU will execute the instruction. This information was divided into ten fields distributed in three words and formed a header. For IPNoSys III, most of these fields are irrelevant and the three header words could be replaced by just one with four fields: program ID (8 bits), package ID (14 bits), number of instructions (8 bits), and type (2 bits). The simulator loads the program into memory in the traditional way, as translated by the assembler, with the three header words and the MAU of IPNoSys III changes the headers when reading and injecting each package on the network. This makes it possible for IPNoSys Original programs to run with compatibility.

As in the previous format, the Program ID field identifies which program belongs to, and the Package ID field identifies that package within the program. The Program ID and Package ID tuple is unique (among regular packages), and it is through it that the controller can manage communication. The control packages generated from the regular packages contain the same IDs as the regular packages that instantiated them.

We define a protocol to operationalize the communication between the network components and the controller, consisting of new control instructions. The nine new instructions, called SDN instructions and listed in Table 1. Of the nine instructions, eight are transparent to the programmer and are created directly by the PPUs, CSUs or controller.

Table 1 - SDN Instructions

Instructions	**Source**	**Destination**	**Function**
SDN_START	PPU	Controller	The instruction used to configure the environment.
SDN_INIT	PPU	Controller	PPU tells the controller the start of the execution of a package.
SDN_END	PPU	Controller	PPU informs the controller that it has finished executing a package.
SDN_REQ	CSU	Controller	CSU has all PPUs occupied and requires the controller to address a node to instantiate another task.
SDN_RESP	Controller	CSU	Controller answer CSU with the address of a new node to instantiate a new task.
SDN_SEND	CSU	Controller	CSU requests a route to communicate with another node.
SDN_RT	Controller	CSU	Controller answer CSU which port send the package.
SDN_INT	Controller	CSU and PPU	Controller requests interruption of package processing and transfer to another node.
SDN_STOP	PPU	Controller	PPU informs the controller of the execution of the last program package.

The first instruction, SDN_START, is the only one that can be optionally created explicitly by the programmer. The programmer uses it to define how the network runs the application, limiting cluster size, or if it should use all nodes in the cluster. This instruction, if used, must always appear in the first package of the application. In the case of running programs of old versions of IPNoSys or if the programmer omits this instruction, the controller uses the default configuration, not limiting the cluster size or the use of the nodes.

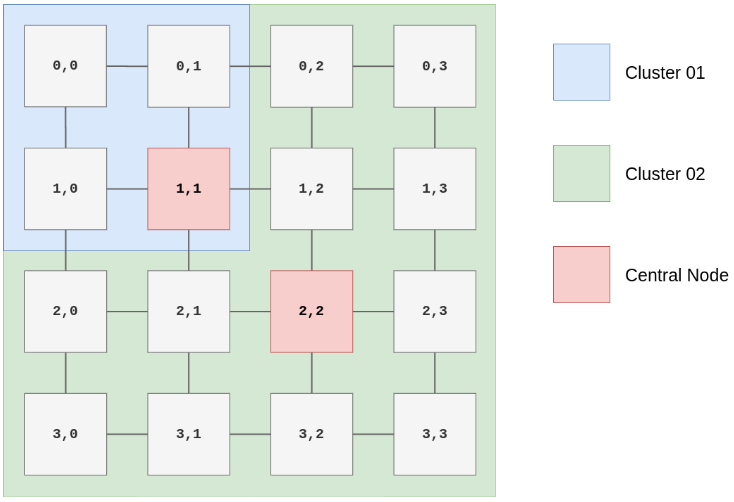
The remaining instructions are used only by the network components (CSU, PPU, Controller). Because they have very specific functionalities, some of these instructions are converted into signals, carrying all the necessary information in just one word, reducing the network traffic. The PPU creates and sends instructions SDN_INIT e

SDN_END to inform when each package's start and finish execution on it. The controller uses these information to know where each package is executing. If a new task needs to be instantiated, the CSU can allocate it in an idle PPU. If there's no idle PPU inside the node, CSU uses SDN_REQ to ask the controller to allocate a new node. The controller sends an SDN_RESP with the new node address. If tasks inside the same node need to communicate, the CSU can transmit control packages between them. But if they both are in different nodes, the CSU sends the controller a SDN_SEND signal, and the controller answers with a SDN_RT signal with the port to route the message. At least, when the PPU finishes the execution of the application's last package, it sends a SDN_STOP signal to update the controller about the finish of execution.

The beginning of an application's execution always starts at one node closest to the center of the network, as there is a greater density of nodes in this region, allowing the tasks that are more to be closer to each other. The first instruction executed by the network is SDN_START to configure the controller with the parameters defined by the programmer. The next instantiated tasks will always be allocated within their respective cluster by the controller.

As an example, we can imagine a scenario of running two applications. In the first one, the programmer defines, by the SDN_START instruction, that his application could run in a cluster containing four nodes. In the second application, the programmer did not define any parameters and the network configure it automatically. The first application starts to run on one of the central nodes and the controller defines a cluster containing four nodes in which all the tasks for that application will instantiate. In the second application, the controller, as there was no initial parameter, uses all available resources and instantiates the first task on the most central node available in this cluster. Figure 1 shows this example.

Figure 8 - Example of a Cluster with Two Applications



We implement the control software using the Floyd-Warshall algorithm to calculate the network's central nodes, the closest available node (to allocate new tasks), and the shortest path between the source and the destination (when two nodes need to communicate). The Floyd-Warshall algorithm has a complexity of O(n^3). The algorithm can be replaced by a more efficient one when necessary since the control executes in a software solution.

4 RESULTS

To validate the proposed architecture, we describe a simulator for the network in SystemC RTL with cycle precision. We made a high-level abstraction for the controller that makes it possible to connect the control software to the network, so requests need one clock cycle. The simulated scenarios aim to evaluate the impact of using the central controller in conjunction with the network and compare the proposed architecture with a conventional NoC.

4.1 EXPERIMENTS SETUP

In the first experiment, we evaluate the impact of the network communication with the controller. To make this experiment possible, we created a modified version of IPNoSys III without a controller, so that it was possible to use the IPNoSys II packages, with three header words that carry the source and destination information. The IPNoSys III without controller needs the programmer to manually inform which node processes each package as a compilation-time mapping, and it is possible to CSU use XY routing algorithm to communicate with other nodes. We simulate both NoCs (the IPNoSys III and IPNoSys III without a controller) using synthetic applications with logical and arithmetic instructions. We use two synthetics benchmarks: an application with a few packages and many instructions and an application with many packages and a few instructions. The network's size was varied (4x4, 5x5, and 6x6), adding more nodes to instantiate more tasks. The metrics evaluated was the execution time in number of clock cycles.

The second simulated scenario was the performance evaluation of IPNoSys III running real applications. As a benchmark, we use a matrix multiplication algorithm and the DCT-2D algorithm. We use the MPSoCBench framework [22] to generate four NoCs integrated with some ISA simulators available on the market: ARM, MIPS, SPARC, and PowerPC and compare the execution times (in clock cycles) to the IPNoSys III.

4.2 EXPERIMENTS RESULTS

The first scenario evaluated was the impact of network communication with the controller. Figure 9 shows the comparison between IPNoSys III connected to the controller vs. IPNoSys III with manual mapping done by the programmer, running an application with few tasks and many logical and arithmetic instructions. It is possible to notice that the impact of communication between the network and the controller remains constant, between 4,16%, 4,32%, and 4,40%, even when the network's size varies. Because of the small number of packages, the controller has little influence on the execution, setting the communication cost close to the cost of using extra words in the header of each package.

Figure 9 - Comparison of the Application with Few Tasks and Many Instructions

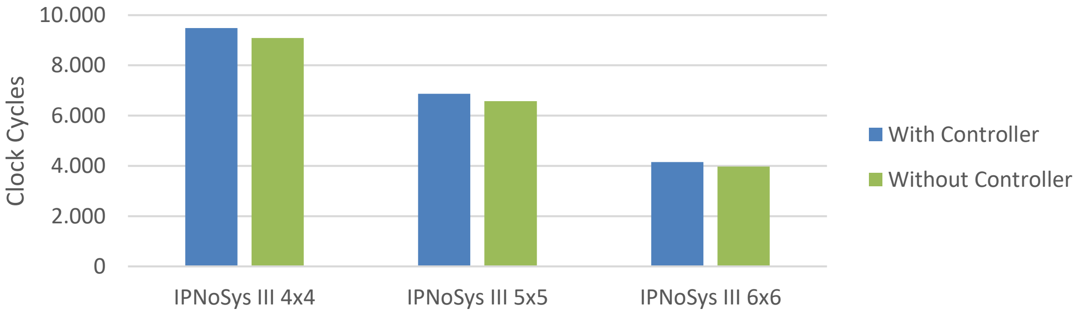
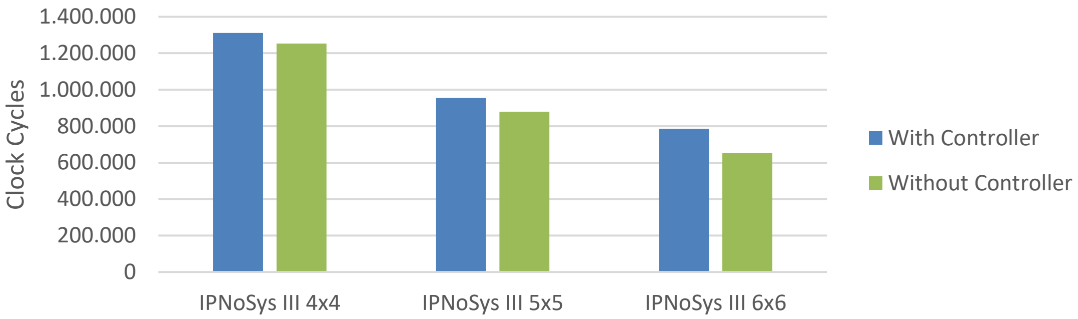


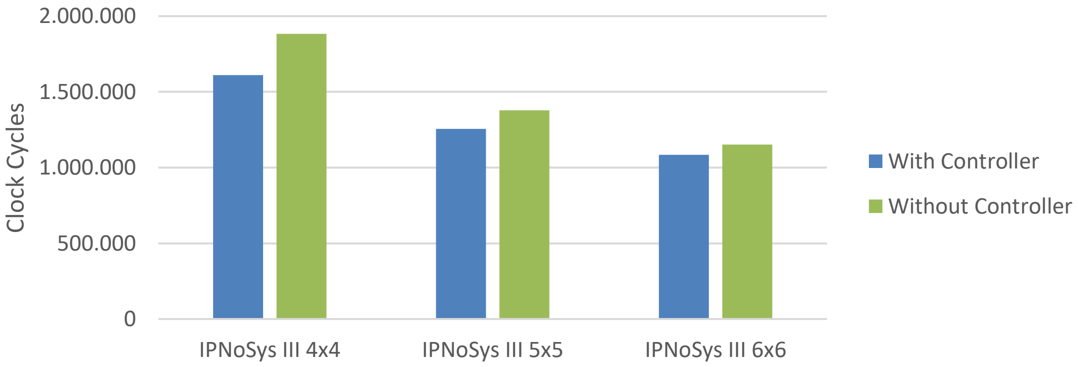
Figure 10 presents a setup similar to the previous one, but with both architectures running an application with many packages (tasks) and that each package has few instructions. In all simulations, it is possible to notice that the compilation-time mapping was more efficient than at runtime mapping, using the controller, varying between 4,34%, 7,98%, and 16,93%. Since there is no communication between tasks, there is no benefit in adding the controller. Moreover, with more tasks to be instantiated, the controller becomes the network bottleneck.

Figure 10 - Comparison of the Application with Many Tasks and Few Instructions



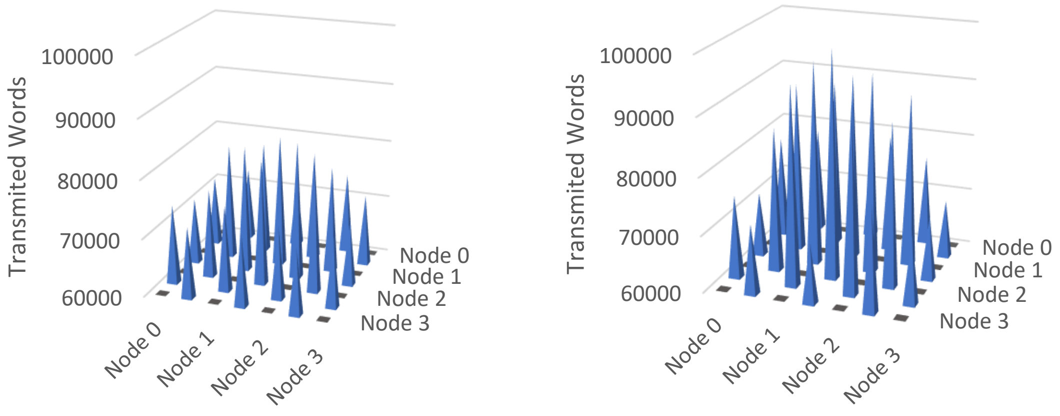
In the before simulations, the controller acts only to instantiate a new task in a new node. In the next simulation for this scenario, we evaluate the IPNoSys III performance when the application tasks communicate. We use the application with many tasks and a few instructions and add some new instructions in each package to send results to different tasks, in order to generates more traffic in the network. The results in Figure 11 shows that IPNoSys III without a controller needs 16,96%, 9,87%, and 6,17% more clock cycles compared to version with controller. In this experiment, we notice a better performance of IPNoSys III with the SDN controller. It happens because the controller acts to find a better route to communicate nodes, despite IPNoSys without controller use XY routing algorithm. The difference decreases when the number of nodes increases because more nodes send a request to the controller, and it becomes the bottleneck.

Figure 11 - Comparison the Application with Many Tasks and Few Instructions with Communication



The Figure 12 shows a graph with the number of transmitted words of each link in a IPNoSys 4x4 for the before simulation. The black dots in the ‘x’ and ‘z’ axis represents the networks nodes, and the ‘y’ axis represents the number of transmitted words of each link. In IPNoSys III with controller, each package has less words, and the controller spreads the communication in the network, how we can see in Figure 12a. The IPNoSys III without controller, has more words in each package and the number total of transmitted words is 10,51% more than the IPNoSys III with controller and the communication stay concentrated in the center of the network, causing congestion in these nodes.

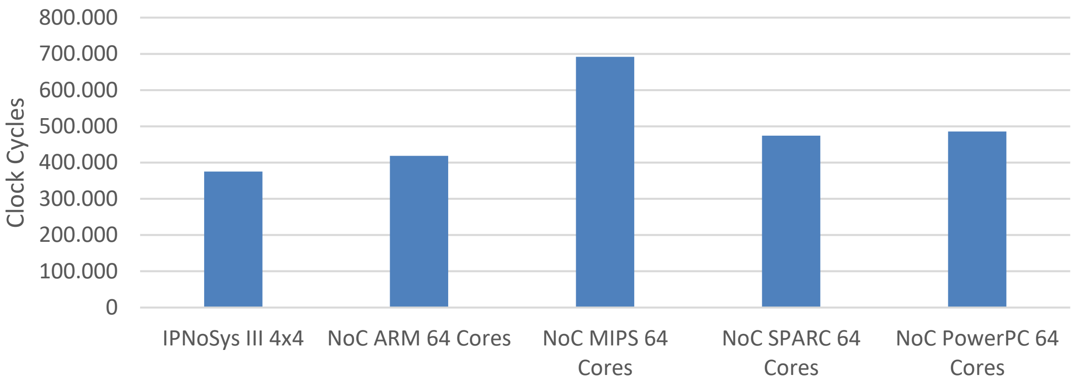
Figure 12 - Transmitted words in a IPNoSys 4x4 with controller (a) and without controller (b)



(a) (b)

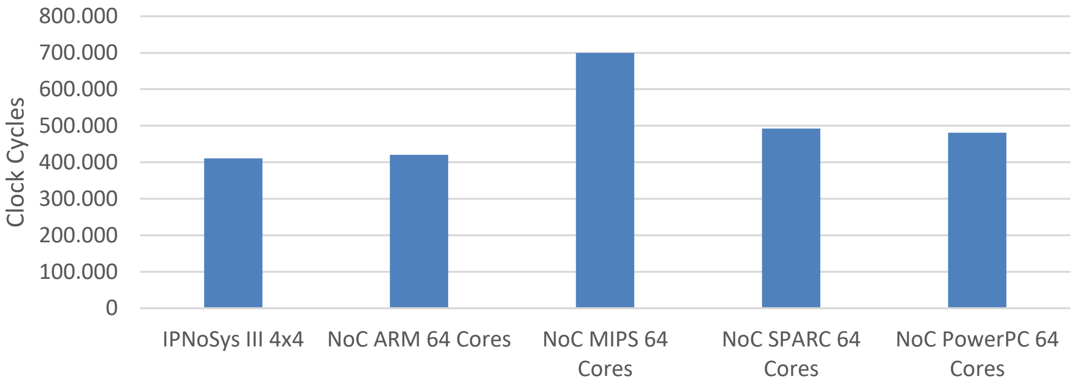
The second scenario evaluated was the behavior of IPNoSys running real applications and comparing them with other NoC. We execute two applications that can run in parallel to obtain the maximum performance of the NoCs. The first application was the multiplication of matrices. We write the applications in PDL’s language to run in IPNoSys III and implement the same in C language to execute in the ARM, MIPS, SPARC, and PowerPC simulators. All architectures run at a frequency of 50MHz. According to Figure 13, it is possible to notice that IPNoSys III has a clock cycle performance better than all other simulated architectures, being approximately 10,47% faster than NoC ARM and almost 45,81% faster than NoC MIPS.

Figure 13 - Execution of Matrix Multiplication



The next application is the DCT-2D algorithm on a 256x256 pixel image. The setup used is the same as the previous test. Figure 14 shows the results in clock cycles for all simulations. In this case, IPNoSys III shows results closer to other architectures, being approximately 2,28% faster than NoC ARM and 42,26% faster than NoC MIPS.

Figure 14 - Execution of DCT-2D



5 CONCLUSION AND FUTURE WORK

This work shows it is possible to join the SDN paradigm and the execution model of IPNoSys, a non-conventional architecture, to obtain a performance increase in parallel applications' execution. The results show a similar performance between the version with the static approach when there is little traffic in the network and better performance when the traffic increases. Dynamic mapping and routing algorithms can make the network perform better than the cost associated with communication between the network and the controller.

As future works, we intend to run the controller software in a real processor, connected to the NoC, and evaluate the cost of another's mapping and routing algorithms. We intend to implement a migrate task algorithm to migrate communicating tasks closer to each other. We are also implementing new benchmarks to compare IPNoSys III in more realistic scenarios.

REFERENCES

[1] C. A. Zeferino and A. A. Susin, “SoCIN: a parametric and scalable network-on-chip,” in *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, 2003, pp. 169–174, doi: 10.1109/SBCCI.2003.1232824.

[2] L. Benini and G. De Micheli, “Networks on chips: a new SoC paradigm,” *Computer (Long. Beach. Calif).*, vol. 35, no. 1, pp. 70–78, 2002, doi: 10.1109/2.976921.

[3] Intel, “Teraflops Research Chip,” 2019. https://www.intel.com/pressroom/kits/Teraflops/index.htm (accessed May 01, 2019).

[4] Tilera, “TILE-Gx72 Processor,” 2019. http://www.mellanox.com/page/products_dyn?product_family=238&mtag=tile_gx72 (accessed May 01, 2019).

[5] L. Soares, P. Dziurzanski, and A. K. Singh, *Dynamic Resource Allocation in Embedded, High-Performance and Cloud Computing*. River Publishers, 2016.

[6] N. Dutt, A. Jantsch, and S. Sarma, “Self-aware Cyber-Physical Systems-on-Chip,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2015, pp. 46–50, doi: 10.1109/ICCAD.2015.7372548.

[7] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jun. 2015, doi: 10.1109/JPROC.2014.2371999.

[8] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013, doi: 10.1109/MCOM.2013.6461195.

[9] Y. Luo, P. Cascon, E. Murray, and J. Ortega, “Accelerating OpenFlow switching with network processors,” in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems - ANCS ’09*, 2009, p. 70, doi: 10.1145/1882486.1882504.

[10] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, “Wireless NoC as Interconnection Backbone for Multicore Chips: Promises and Challenges,” *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 2, no. 2, pp. 228–239, Jun. 2012, doi: 10.1109/JETCAS.2012.2193835.

[11] L. Cong, W. Wen, and W. Zhiying, “A configurable, programmable and software-defined network on chip,” in *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, Sep. 2014, pp. 813–816, doi: 10.1109/WARTIA.2014.6976396.

[12] X. Zhou and Z. Zhu, “A dynamic task mapping algorithm for SDNoC,” *Microelectronics J.*, vol. 63, pp. 58–65, May 2017, doi: 10.1016/j.mejo.2017.03.003.

[13] A. Scionti, S. Mazumdar, and A. Portero, “Towards a Scalable Software Defined Network-on-Chip for Next Generation Cloud,” *Sensors*, vol. 18, no. 7, p. 24, Jul. 2018, doi: 10.3390/s18072330.

[14] P. Dzhunev, “Improving Network Monitoring with Software Defined Networking,” *IEEE Commun. Mag.*, vol. 46, pp. 114–119, 2013, [Online]. Available: https://inis.iaea.org/search/search.aspx?orig_q=RN:46130071.

[15] S. R. Fernandes, B. C. Oliveira, and I. S. Silva, “Using NoC routers as processing elements,” in *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design Chip on the Dunes - SBCCI ’09*, 2009, vol. 24, p. 1, doi:

10.1145/1601896.1601927.

[16] T. R. B. S. Soares, I. S. Silva, and S. R. Fernandes, “IPNoSys II: A New Architecture for IPNoSys Programming Model,” in *Proceedings of the 28th Symposium on Integrated Circuits and Systems Design - SBCCI ’15*, 2015, pp. 1–7, doi: 10.1145/2800986.2801012.

[17] J. Wang, M. Zhu, C. Peng, L. Zhou, Y. Qian, and W. Dou, “Software-Defined Photonic Network-on-Chip,” in *The Third International Conference on e-Technologies and Networks for Development (ICeND2014)*, Apr. 2014, pp. 127–130, doi: 10.1109/ICeND.2014.6991365.

[18] S. Ellinidou, G. Sharma, T. Rigas, T. Vanspouwen, O. Markowitch, and J. Dricot, “SSPSoC: A Secure SDN-Based Protocol over MPSoC,” *Secur. Commun. Networks*, vol. 2019, p. 11, Mar. 2019, doi: 10.1155/2019/4869167.

[19] S. R. Fernandes, B. C. Oliveira, M. Costa, and I. S. Silva, “Processing while routing: a network-on-chip-based parallel system,” *IET Comput. Digit. Tech.*, vol. 3, no. 5, pp. 525–538, 2009, doi: 10.1049/iet-cdt.2008.0071.

[20] S. R. F. de Araújo, “Projeto de Sistemas Integrados de Propósito Geral Baseados em Redes em Chip – Expandindo as Funcionalidades dos Roteadores para Execução de Operações: A plataforma IPNoSys,” Tese de Doutorado, Universidade Federal do Rio Grande do Norte, 2012.

[21] I. A. de S. Moura, “Proposta , Validação e Avaliação Qualitativa de uma Arquitetura MPSoC Baseada Nos Elementos de Processamento de IPNoSys II,” 2020.

[22] L. Duenha, H. Almeida, M. Guedes, M. Boy, and R. Azevedo, “MPSoCBench: A Toolset for MPSoC System Level Evaluation,” 2014, [Online]. Available: http://archc.lsc.ic.unicamp.br/benchs/mpsocbench/index.html.