Design and Implementation in VHDL of RPU components of IPNoSys architecture

José Dijon de Oliveira Neto and Sílvio Roberto Fernandes de Araújo
Department of Exact Science and Natural (DCEN)
Rural Federal University of the Semi-Arid (UFERSA)
BR 110, Km 47 – CEP 59.625-900 – Mossoró – RN – Brazil
E-mails: jdoneto@hotmail.com, silvio@ufersa.edu.br

***Abstract*—This paper aims to present an implementation of simplified version of the main IPNoSys architecture component. Such implementation uses VHDL and a methodology to design, implementation and testing. The first functionality results of components integrated and simulated confirm the efficiency of the methodology.**

I. INTRODUCTION

The computer architectures have evolved from models with a single central processing unit (CPU) for which one with organization optimized or parallelism exploration. models with components or better organization replication which can be applied parallel scanning techniques. With the advent of the solutions, combined with the increased integration capabilities in silicon wafers, it was possible to build multiprocessor systems within a single chip, also called MPSoC (*Multiprocessor System-On-Chip*). The main interconnection mechanism for this type of system are the NoC (*Network-on-Chip*) [1].

Based on the characteristics of the NoC, arises IPNoSys (*Integrated Processing NoC System*) [2], which combines communication and processing on the same infrastructure. It is a general purpose architecture that presents a significant reduction in terms of runtime through parallelism using *multithreading*.

IPNoSys was originally developed in *SystemC* [3] with cycle accuracy for all architecture components. However, some results may not be obtained with the current implementation, such as theoretical maximum operation frequency; chip area; and the total power dissipation. Therefore, the aim of this work is related to the construction of a model for synthesis on FPGA (*Field-programmable Gate Array*).

This article is organized as follows: section II shows the general characteristics of IPNoSys, the programming model and tools available; Section III addresses on the design methodology, showing the components designed and implemented this work and their testing; Section IV displays the simulation results and the software and the device used to achieve these results; Finally, following the conclusions, future work and references used.

II. IPNOSYS PLATFORM

IPNoSys (Figure 1) is a general purpose processor with unconventional architecture that take advantage of infrastructure and features from network-on-chip to favoring parallel communication and processing. Such architecture presents a model of computation directed to packages, which connects the processing and communication, once the programs are described on a specific packet format. The architecture is formed by a network of RPU (Routing and Processing Unit) with a MAU (Memory Access Unit) in each corner. The MAUs load and store data, as well as inject packets and execute syncronization instructions. The RPUs are responsible to route packets and execute logic and arithmetic instructions. It also has an input and output system based on DMA (*Direct Memory Access*). It has the following characteristics: topology NoC Square 2D grid; It uses at least two virtual channels; routing XY modified; switching combining *VCT* and *wormhole*; control credit-based flow; distributed arbitration; and storage at the entrance. For more information, see [2].

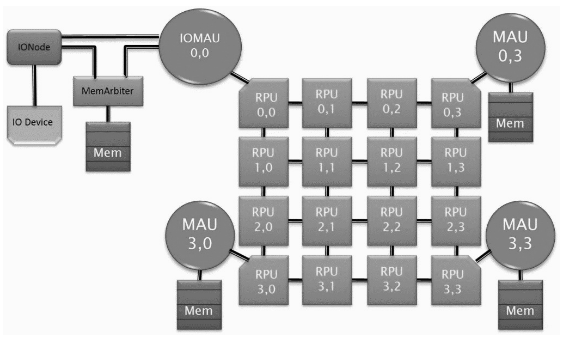


Fig. 1. IPNoSys architecture [2]

One of the great advantages of IPNoSys system is its parallel computing power, presented in the form of packages of pipeline injected by the same MAU and the simultaneous injection of packets across different MAUs (real parallelism). It is possible to exploit parallelism in instruction level (ILP) or *multithreading* (TLP). The ILP parallelism is to verify the data dependency graph and include instruction sequences with addiction in the same package and independent instructions in parallel packages. While multithreading parallelism in IPNoSys, is to divide tasks or snippets of code in independent packages that can be injected alongside the four existing MAUs, mainly using the technique of *loop unrolling* [4]. In [5] is shown that the exploitation of parallelism using multithreading causes a reduction in the relatively more significantly runtime that ILP exploitation. In the case of implementation of DCT - 2D (*Two Dimensional Discrete Cosine Transform*), the performance in IPNoSys is 3.7 times lower compared to a conventional MPSoC [2].

IPNoSys afford programming support through the symbolic language called PDL (*Package Description Language*), an assembler and a simulation environment. The programming model is to describe the applications/programs across one or more packets, which are injected and executed in the system in the order of dependence of data between computations that each package is [6]. Programs written in PDL are submitted to the assembler that is responsible for creating the equivalent object code. This object code is used as input by the SystemC simulator in implementing programs in MAUs and RPUs. During the simulation, multiple results are generated and stored in a text file, including: application runtime; memory required; information about each package; average system utilization; computation time, transmission and idle time each RPU; and total power dissipated by buffers and architecture during the execution of the application.

III. Design and Implementation in VHDL

For the design and implementation of an architecture it is necessary use an efficient methodology. According to [7], the methodology of an architecture project basically involves the following steps: to relate characteristics of the target processor, describe the micro-programs using flowcharts, data path design, description of the state machine, the hardware description in HDL and test processor through simulation.

Following this methodology, it was started the RPU implementation, where each component was divided into two parts: the control unit and data path. The data path of the project is a component of the combinational logic using a top down approach in order to specify all subcomponents. The description of the control unit behavior is to build a finite state machine (FSM). Through their states, this FSM describes what actions will be taken and the time that they should be executed, so that synchronize the individual operations performed by components.

The architecture of the RPU consists of: buffers the inputs, one crossbar, arbiters in outputs, one ALU (*Arithmetic Logic Unit*) and SU (*Synchronization Unit*) as shown in Figure 2. In [8] you can get a more detailed description and design of all internal components of the RPU, except the arbiter.

Basically, the arbiter has two functions: to transmit words of a packet and request the execution of a instruction, in addition to the traditional function to solve disputes by the same output. For this, the arbiter removes the current instruction package (arithmetic type, logical or shift) and sends a request to the ALU run. Once this request is accepted, the ALU receives the instruction code and operands, performs the computation and returns the result to the arbiter. The result is saved in the buffer results (Figure 3), in address determined the instruction, and marked as valid. Before transmitting, the arbiter checks the availability of transmission channels in the destination buffers in the next RPU and, if possible, an exclusive channel is maintained until it is passed the ultimate package before stored in the input buffer. During transmission, a counter is incremented every word sent and used to check if you have any valid data on the results buffer to be transmitted. If the word is then transmitted by the arbiter and marked as invalid the results buffer. Otherwise, the arbiter gives the word that is the Input Buffer. Because of the complexity and space issues, the arbiter's drawing will be omitted.

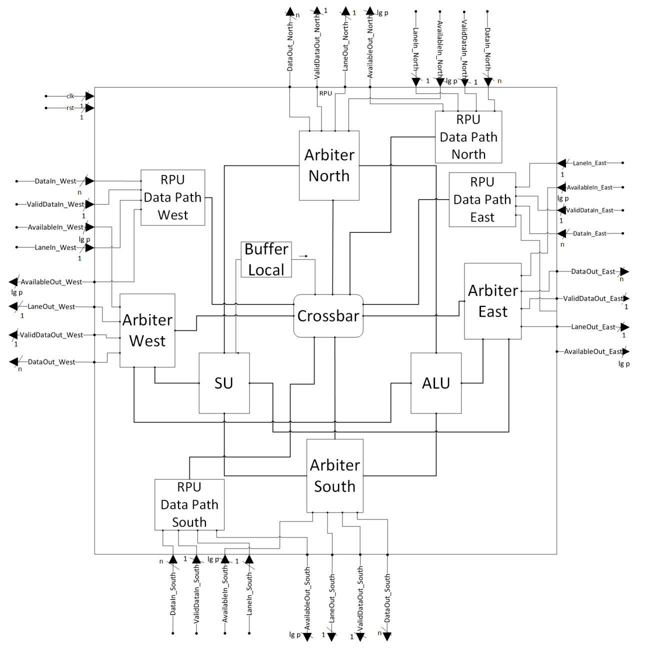


Fig. 2. Abstract version RPU

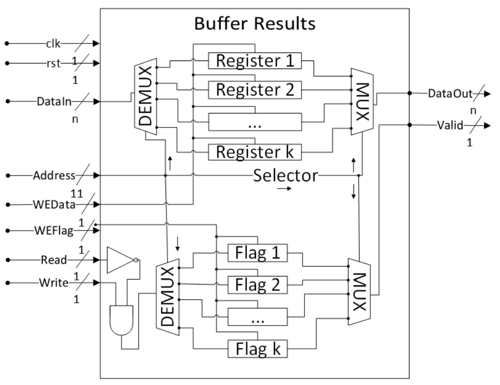


Fig. 3. Buffer Results

Figure 4 illustrates the components that belong to the purpose of this article, i.e.: one Buffer (input), an ALU, a Buffer Results and Arbiter. After the development of each unit or component, it is necessary to incorporate such structures in an entity, in this case, the RPU. In this present version some RPU functions are not availables, such as the routing of the package, the creation of control package and the treatment of simultaneous requests the ALU and SU. However, it presents the functions: store the words received the package, execute a statement, save and read the result of computing and transmit the packet.

The last step of this methodology is the functional simulation or temporal where you must use different levels of testing to ensure that the entity is operating as expected. For this, at first, unit tests are applied to a single unit system, which does not possess external dependencies, to test it alone (considering all possible scenarios). Figure 5 illustrates a generated script to automate unit tests the ALU. Signs *AN, BN, RN and ON* mean, respectively, the Data A, Data B, Operation and North ALU port Results for a sum. The *Assert* and *Report* are used

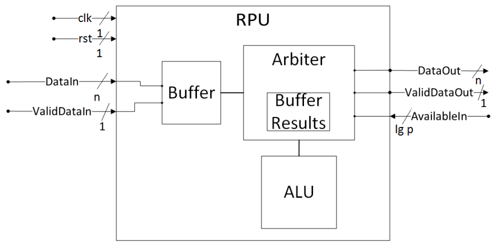
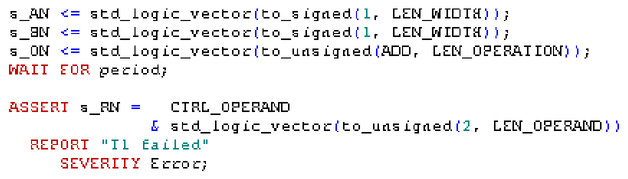


Fig. 4. The simplified version of RPU project

to notify illegal conditions (very useful when the code pass through maintenance), that is, when the test fails.



```
s_AN <= std_logic_vector(to_signed(1, LEN_WIDTH));
s_BN <= std_logic_vector(to_signed(1, LEN_WIDTH));
s_ON <= std_logic_vector(to_unsigned(ADD, LEN_OPERATION));
WAIT FOR period;

ASSERT s_RN =   CTRL_OPERAND
              & std_logic_vector(to_unsigned(2, LEN_OPERAND))
  REPORT "T1 failed"
     SEVERITY Error;
```

Fig. 5. Script excerpt in ModelSim for ALU

Finally, when two dependent parts of the system pass through successfully unit tests, then an integration test must be prepared and implemented to verify proper synchronization between the components.

IV. Results

To implement the architecture of Figure 4 we used the Quartus II Web Edition 14.0 [9]. In this article, the FPGA device chosen as a target for synthesis was the family EP4CE115F29I8L *Cyclone IV E* manufacturer *Altera*.

The description in VHDL simulation was performed using the ModelSim-Altera Starter Edition software (can be used independently of the Quartus), by running scripts to automate the testing process.

The Table I contains the results obtained by compiling, for chip area, maximum operating frequency and power dissipation of the components described.

TABLE I. Results obtained with the description of the hardware

Component	Area on Chip (LE)	Maximum Operating Frequency (MHz)	Power Dissipation (mW)
Buffer	430	248.63	158.66
Buffer Results	1,392	965.25	157.99
ALU	2,615	–	214.58
Arbiter	1,996	97.88	207.33
RPU	4,182	91.02	154.41

In the simulation images the results are presented in hexadecimal. Initially, the word will only be inserted into the Input Buffer if it is valid. The first three words of Figure 6 correspond to header and contains information on the type of package, the number of instructions and the pointer (number of words processed along the route of the package, indicating the next instruction to be executed). The next words are instruction type and operand, to be performed in the ALU and the address where it should be saved the result of the computation. Soon after, the operands are inserted. At this time, the Arbiter calls ALU that add the operands (2 and 1) and enter the result in position 4, from the first instruction (starting from 0).

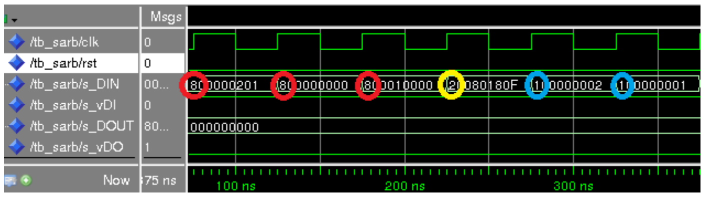


Fig. 6. Insert the three header words (red), an instruction (yellow) and two operands (blue) in the Buffer

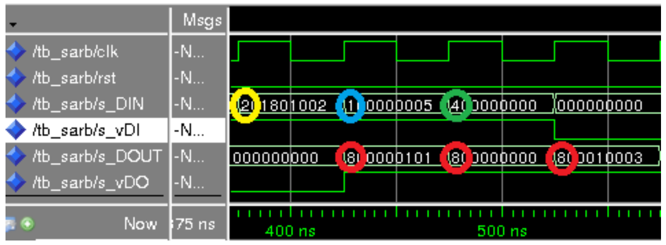


Fig. 7. Insert another instruction (yellow), operating (blue) and packet end word (green) in the Buffer and sending the header three words (red)

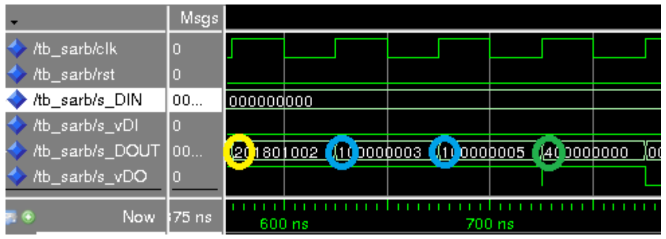


Fig. 8. Sending the remaining instruction (yellow), insertion loss (blue), sending the remaining operands (blue) and the packet end word (green)

V. Conclusion

By adopting this design methodology, it was possible to obtain faster and more accurate implementations. Also applies to the benchmark methodology, as test cases include virtually all possibilities, are executed quickly and accurately and present the test results immediately.

Moreover, the implementation of RPU in VHDL components allowed it to be obtained results of the area, frequency and power of each entity (unlike in SystemC implementation, where only the buffer has a calculated power).

According to simulation, it can be concluded that for an instruction with two operands, the processing time takes 7 clock cycles, and the processing in ALU 1 cycle lasted for a sum. Regarding the transmission time, the packet begins to be transmitted in the next cycle of the computation.

As future work will be done to integrate the current implementation with the SU, treatment instructions and the other types of packages, the addition of simultaneous requests and routing, and the development of other components of IPNoSys. Finally, the assembler will be adapted to generate memory initialization file (.mif) required to implement a program directly to the FPGA.

References

[1] Benini, L. and Micheli, G. D. (2002) *Networks on Chips: A New SoC Paradigm*. IEEE Computer Society Press. 35: p.70-78.

[2] Araújo, S. R. F. (2012) *Projeto de Sistemas Integrados de Propósito Geral Baseados em Redes em Chip – Expandindo as Funcionalidades dos Roteadores para Execução de Operações: A plataforma IPNoSys*. 191 f. Tese (Doutorado em Sistema e Computação) – Departamento de Informática e Matemática Aplicada. Universidade Federal do Rio Grande do Norte. Natal.

[3] Accellera Systems Initiative. SystemC. Disponível em: <http://www.accellera.org/downloads/standards/systemc>. Acesso em Out. 2013.

[4] Pereira, M. M. et al. (2008) *Using traditional loop unrolling to fit application on a new hybrid reconfigurable architecture*. In: 23rd Annual ACM Symposium o Applied Computing. Fortaleza.

[5] Fernandes, S. et al. (2009) *Processing while routing: a network-on-chip-based parallel system*. In: IET Computers & Digital Techniques, v.3, n. 5, p. 525-538. Disponível em: <http://link.aip.org/link/?CDT/3/525/1>

[6] Fernandes, S.; Silva, I. S.; Kreutz, M. (2010) *Packet-driven General Purpose Instruction Execution on Communication-based Architecture*. In: JICS – Journal of Integrated Circuits and Systems, v. 5, p. 14. ISSN 1807-1953.

[7] Costa, R. V et al. (2012) *SICXE: Improving Experience with Didactic Processors*. Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC). Natal.

[8] Neto, J. and Fernandes, S. (2013). *Metodologia de projeto para descrição da arquitetura ipnosys em vhdl*. In *EPOCA 2013*, Mossoró/RN.

[9] Altera Measurable Advantage. Quartus Web Edition and ModelSim-Altera Starter Edition. Disponível em: <https://www.altera.com/products/design-software/fpga-design/quartus-ii/quartus-ii-web-edition.html>. Acesso em May. 2015.