PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE ENGENHARIA
FACULDADE DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

HNPlus – A Network on Chip Prototyping Platform with a Generic Traffic Generation Scheme

BRUNO FIN FERREIRA
ISMAEL LUÍS HEINEN

END OF TERM WORK

Prof. Dr. Ney Laert Vilar Calazans
Advisor

Porto Alegre
2011

CONTENTS

List of Abbreviations

BRAM	Block RAM
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous Locally Synchronous
GAPH	Grupo de Apoio ao Projeto de Hardware
GLP	GALS Low Power
HDL	Hardware Description Language
HW	Hardware
IC	Integrated Circuit
IP	Intellectual Property
NoC	Network on Chip
RAM	Random Access Memory
SoC	System on a Chip
SW	Software
TG	Traffic Generator
TI	Traffic Injector
TR	Traffic Receptor
VC	Virtual Channel
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

List of Figures

1. INTRODUCTION

The evolution of the technologies used to fabricate integrated circuits (ICs) brought the possibility of implementing hundreds of millions of transistors in a single chip. Thus, modern ICs can contain all elements of a complex system in a single chip. Such devices receive the name of Systems on a Chip (SoCs) [MOR10]. A SoC is typically composed of information processing modules (the so-called intellectual property cores or IP cores), an intrachip interconnect architecture and interfaces for peripherals of the IC.

Traditional intrachip interconnect architectures, such as dedicated wires and buses, can be considered insufficient for current SoCs. Dedicated wires have poor reusability and flexibility, while shared buses transmit only one data per clock cycle and offer limited scalability. The emerging Network-On-Chip (NoC) interconnect architecture provides an energy-efficient and scalable communication solution for multiple cores, serving as a powerful replacement for bus architectures. Multiple point-to-point data links interconnected by switching nodes typically compose an NoC. Any source module may relay messages to any destination module over several links, by making routing decisions at the switching nodes or at the message´s source.

Due to increase in inter-module communication requirements and demand for reduced time-to-market, validation of intrachip communication infrastructures plays a major role in chip design [LOT11]. Typically, software or HDL simulators accomplish the validation process. The increasing complexity and size of NoCs mandate that simulation software deals with too many signals and data, making the validation process take a significant fraction of the overall chip design time, maybe affecting accuracy. In addition, real application speeds are not possible with software simulators. Fast, cycle-accurate FPGA emulators are an alternative to software simulators as they reduce validation time without compromising accuracy. FPGA emulators allow the user to exploit the parallel nature of hardware, thereby enabling the detection of design and architectural problems early and reducing the number of chip respins [LOT11].

1.1 Motivation

The increasing complexity of NoCs can difficult the process of validating them by software simulator because of time consumption that it takes. Instead, validating complex NoCs by emulation platforms can be a solution, once emulation in hardware can present a three or more order of magnitude gain in terms of time execution when compared to software simulators. Besides this, platforms based in FPGAs are powerful, simple to use and are available to execute this work.

1.2 Objectives

The main objective of this work is to contribute to the validation process of NoCs, by ameliorating an existing NoC evaluation hardware environment, named HardNoC-GLP. The specific goal is the development of a new traffic injector module for an existing environment, making the necessary adjustments and enhancements in the system to ensure its correct operation.

1.2.1 Strategic Objectives

During its development, this work achieved several strategic objectives. Examples are the exploration and understanding of how NoC emulation platforms work, the use of HDL-based hardware design techniques and the dominium of FPGA prototyping techniques. The achievement of these objectives required the use of much of the knowledge acquired during the Computer Engineering course, applying concepts from various disciplines. This included concepts of software development, communication networks, digital circuits, computer organization and architecture, microelectronics and circuit design techniques that were revisited and solidified. Applying such knowledge to this work gave the authors additional experience, close to that needed to develop commercial products.

1.2.2 Specific Objectives

The specific objectives involved enhancing the existing HardNoC-GLP platform, developed at the GAPH research group, to support the standard traffic description files produced by the ATLAS NoC design and verification environment. The new platform denomination is HNPlus. After concluded the development wortk, it was possible to validate and test instances of a specific NoC (the Hermes-GLP) with several traffic patterns distinct from those previously possible. While the original HardNoC-GLP platform worked with only two traffic types, the improvement this work proposed and implemented enables the exercise of NoCs with virtually any traffic type, incuding real traffic data, what makes the exploration of new NoC designs more flexible. Additionally, this work provides en anhancement in the software used to provide interaction between the user and the HNPlus platform. The new software improves the original interface, rather crude and difficult to use, by an interface that adds semantic meaning to the NoC packet structure.

1.3 Organization of the Document

The rest of this work comprises five additional chapters. Chapter 2 describes the basic concepts used along the text. Chapter 3 describes related work. Chapter 4 presents the proposed architecture and its development details. Chapter 5 presents the validation and prototyping of the new platform beyond experiments and some analysis of the results obtained with it. Chapter 6 contains some conclusions and directions for future work.

2. BASIC CONCEPTS

This Chapter explores basic concepts of the main subjects involved in this work. The presentation comprisers five Sections. General concepts include NoCs, addressed in Section 2.1, and globally asynchronous locally synchronous basics and a brief account of the Hermes-GLP NoC both in Section 2.2. The used traffic generation environment, ATLAS, is the focus of Section 2.3. Follows a brief definition and discussion of concepts related to emulation of digital systems, in Section 2.4. The Chapter ends with a brief presentation of the original HardNoC platform, in Section 2.5.

2.1 NoCs

Given the limitations of dedicated wires and buses in intrachip communication architectures of current ICs, NoCs are emerging as a solution to existing interconnection architecture constraints, due to the following NoC good characteristics: *(i)* energy efficiency and reliability; *(ii)* scalability of bandwidth when compared to traditional bus architectures; *(iii)* reusability; *(iv)* distributed routing decisions [MOR04].

NoCs use concepts of telecommunication networks and parallel processing networks, where information is organized and transferred in layers and protocols. Switching nodes and links compose a network. The exchange of *messages* among IP cores accomplishes the exchange of information between source node and target nodes of a system. Often, the structure of some message types is not adequate for communication purposes on NoCs. This leads to the concept of packet. A *packet* is a standard way of representing information in a form adequate for communication. One packet may correspond to a fraction, one or even several messages. In the context of NoCs, packets are frequently a fraction of a message. A header and a payload often compose packets. The *switching nodes* realize the transfer of information between the IP cores by routing the packets accordingly to get them closer to their destination IP cores. To do so, switching nodes comprise a switching fabric, a routing logic, an arbitration logic, and associated communication ports connecting through links to neighbor switching nodes, depending on the network topology. A *network topology* is defined by the way switching nodes connect to one another. A very popular network topology is the 2D Mesh topology, where switching nodes occupy two dimensions forming a square or a rectangle where nodes connect to neighbors in vertical and horizontal directions. The *communication mechanism*, the *switching mode* and the *routing algorithm* are functions of the network topology and are necessary items to the execution of services provided by the NoC.

The *communication mechanism* specifies how messages travel through the network. Two methods for transferring packets are *circuit switching* and *packet switching*. In *circuit switching*, a path from source to target is established by the allocation of a sequence of switching nodes. This is a *connection*. Packets start to travel through the allocated path and, during this connection, any other communication desiring to use any of the allocated path segments is denied. In *packet switching*, packets follow along the network without any need for deploying connection establishment procedures.

A packet switching mechanism requires the use of a *switching mode*, which defines how packets move along switches. *Store-and-forward*, *virtual-cut-through* and *wormhole* are the most important modes. In *store-*

and-forward mode, a switch just forwards a packet when it has been completely received. The switch decides what to do every time a packet comes in. In *virtual-cut-through* mode, the switch needs a guarantee from the next switch that the packet can be received completely. It implies in a buffer to store the complete packet. The *wormhole* switching mode is a variant of the virtual-cut-through mode that avoids the need of large buffer spaces. In this mode, a packet is divided into smaller units, called flits. A *flit* is the smallest unit of flow control. Only the header flit has the routing information. The rest of the flits of the packet must follow the same path reserved for the header.

The *routing algorithm* defines the path a packet takes to go from the source to the target. It is possible to classify a routing algorithm in *source* or *distributed*, depending on where the decisions are taken. In *source* routing the whole path is decided at the source switch node while in *distributed* routing the path is decided by a set of local decision taken at every switch. The *XY algorithm* can be a source or distributed routing algorithm that routes flits according to the position the switch occupies on the network topology, usually 2D Mesh. First, the XY algorithm checks if the incoming flit has arrived at its target. If so, the flit is routed to the switch's local port, typically connected to an IP core. If not, the XY algorithm routes the flit to a switch closer to the target always sending it to a horizontally located neighbor. The flit is sent up or down only if there is no horizontally located neighbor or if the existing ones are farther from the target.

Besides the communication system, a NoC provides communication services to the SoC. As essential services to a SoC, it is possible to cite data integrity, throughput and latency guarantees. In the context of NoCs, latency is the time a packet takes to go from the source to the target. If divided in flits, as in wormhole mode, latency is the time delay from the insertion of the first flit in the NoC until the time the last flit composing the packet leaves the NoC.

2.2 The GALS Style and the Hermes-GLP NoC

Due to the facility of implementation and verification, the use of a single clock signal in digital system design has been widely used along the years. This approach, however, brings some undesired issues, like clock distribution problems, clock skew and clock power consumption. It was easy to deal with these problems until the 80's, but they became a major limitation with the technology scaling of VLSI circuits into deep submicron geometries.

As an alternative to the synchronous design style, the asynchronous style treats time as a continuous variable, creating the possibility to solve clock skew and clock power problems. Completely asynchronous systems, though, are limited by a generalized shortage of adequate design automation (computer aided design or CAD) tools.

A strategy applicable to fill the gap between synchronous and asynchronous design is the adoption of globally asynchronous locally synchronous (GALS) techniques [CHA84]. The adoption of GALS keeps the use of synchronous methods and CAD tools for the design of synchronous modules and transfers all synchronization problems to the interface between modules. In this context, some NoCs with GALS communication support have become popular [VAN08].

The proposition of the Hermes-GALS Low Power NoC (Hermes-GLP) NoC [PON08] intended to support the GALS paradigm, and at the same time reduce power consumption using mechanisms that dynamically adapt the operating conditions of routers in response to communication requirements [PON08].

Hermes-GLP builds upon the synchronous Hermes NoC [MOR04] and includes bi-synchronous FIFOs for communication between each pair of routers and between a router and its local IP Core.

Each Hermes-GLP router is locally synchronous, with all its ports switching at the same clock frequency. The communication between the router's ports and the rest of the NoC, however, passes through a bi-synchronous FIFO, because the rest of the NoC could be using different clock signals. This occurs because the router has available a set of input clock signals and through an internal logic is capable to choose among these different clocks which to use as router clock. Clock selection is made at runtime, according to the priority of all traffics instantaneously passing through the router. If the priority associated with the traffic set is low, the router automatically selects a lower frequency to operate, thus reducing the power consumption by the NoC. If instead, the traffic priority is high, the router selects a higher clock frequency to send the packet according to the urgency it requires. If there is more than one traffic priority, the router sets its clock according to the highest priority one. If there is no traffic in the router, it is possible to gate all clocks and turn the idle router off. It returns to activity immediately if there are incoming packets asking for transmission.

By reducing clock frequency in each router that is idle or with low priority traffic, Hermes-GLP reduces the power consumption of the clock signal at these routers and globally in the NoC.

2.3 ATLAS

ATLAS is an open source tool designed by the GAPH research group. It automates various processes related to the design flow for NoCs proposed by the same group. NoC Generation, synthetic Traffic Generation, NoC Simulation, Traffic Evaluation and Power Evaluation compose the basic current design flow.

In the NoC generation step, it is possible to configure the parameters of the NoC that to create, such as channel bandwidth, buffers depth, number of virtual channels and flow control strategies. After defining the necessary parameters, the user can ask the software to generate all the hardware modules that compose the NoC and appropriate testbench files for simulation. Modules employ VHDL code, and serve as input to classic FPGA synthesis tools. In addition to basic testbenches, the user may generate optional SystemC files that correspond to the traffic injection modules associated to each IP core.

The traffic generation step produces traffic files that simulate the behavior of the processing modules connected to the NoC, injecting synthetic traffic into the NoC through the previously mentioned SystemC modules during simulation. Traffic configuration evolves setting the frequency of packet injection, the target for each IP core, the number of packets to be transmitted, the size of each packet and a probability distribution model, which guides the injection of traffic in the NoC. It is possible to configure a separated traffic for each IP core, with very different parameters. After the definition of a traffic scenery, traffic files are generated for each IP and contain the information needed to reproduce the defined scenery and the traffic data that will be transmitted. The user may define as many traffic sceneries as needed, and choose one of these for simulation in the next step.

The simulation files generated by ATLAS to each IP are just a set of lines describing packets to inject in the NoC, with the format displayed in Figure 2.1. Each flit contains the number of bits set by the user in the NoC generation step. The first field contains the ideal timestamp value for injecting the packet into the NoC. The second field contains the target IP that will receive the current packet. The third field contains the payload size set by the user subtracted by six flits. The reason for this is explained later. The fourth field contains the source IP that will send the packet. The fifth field, composed by four flits, is the packet generation instant, and corresponds to the time the packet should be first inserted into the NoC (it is a repetition of the first field). ATLAS calculates it according to the probability distribution chosen by the user. The sixth field contains the packet sequence number that is the order in which the packet will be inserted into the NoC considering all the packets generated for all routers in the current scenery. This sixth field comprises two flits. The rest of the packet flits are the actual data flits and do not have a fixed number. The data information is synthetic and ATLAS generates it randomly. In addition, data is present only if it is expected to be transmitted, otherwise they do not exist. Therefore, the first six fields form the minimum amount of information sent that composes an ATLAS packet. The reason the third field is subtracted by six flits is the fact that the time the packet should be inserted into the NoC can be different of the time it is actually inserted. These values differ because the local router can be busy with other traffic in the moment the new packet should enter it. Therefore, the time information in the fifth field is not the one used by the target IP to calculate latency values. To be possible to calculate the latency at the target IP, the exact injection moment information must travel within the packet. To send the right insertion time, the simulation/emulation process produces this information and injects it in the packet dynamically. As the payload size is used to count the number of flits to be read from the memory and inserted into the NoC, it should not count reading these values from memory. Instead, these two will be inserted in the packet during simulation.

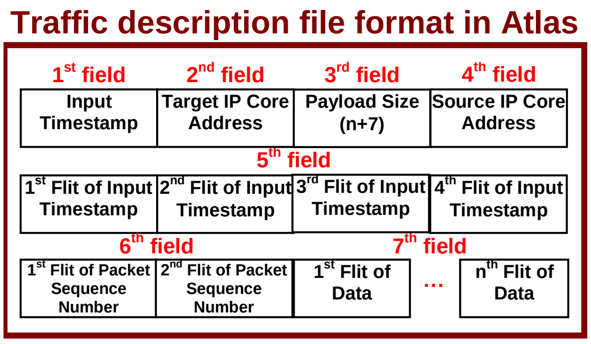

Traffic description file format in Atlas
1st field
2nd field
3rd field
4th field
Input Timestamp
Target IP Core Address
Payload Size (n+7)
Source IP Core Address
5th field
1st Flit of Input Timestamp
2nd Flit of Input Timestamp
3rd Flit of Input Timestamp
4th Flit of Input Timestamp
6th field
7th field
1st Flit of Packet Sequence Number
2nd Flit of Packet Sequence Number
1st Flit of Data
...
nth Flit of Data

Figure 2.1 – General structure for describing traffic in ATLAS traffic files.

The next step implies NoC simulation. It runs an external simulator (today it supports Mentor Graphics Modelsim natively) associated with ATLAS, performs the simulation and generates the simulation output files.

The last step provides the analysis of the simulation results. The traffic evaluation mode facilitates the visualization of the results by presenting different type of graphs, tables and textual accounts.

Power Evaluation is only applicable to the Hermes NoC so far, and is omitted here.

2.4 Analysis by Emulation

Digital systems analysis procedures can be divided into two groups: abstract and concrete analysis. In abstract analysis, the techniques are based on manual or computational manipulation of abstract models of the system to be analyzed. The physical implementation is not required. Concrete analysis, on the other hand, is based on manual or computational manipulation of concrete implementations of the system to be analyzed. The partial or complete physical implementation is necessary.

The complexity to analize NoCs by abstract simulation is increasing with the complexity of these.

When the routers become more complex, more signals need to be analyzed by the simulation software. In addition, the increasing speeds traffic drives in a NoC make the task of analyzing a significant period of it really time consuming. Simulation software needs to deal with too much signals and data and can waste much time to do it.

Simulation by emulation, on the other hand, is one subtype of concrete analysis and can exercise NoCs at actual hardware speeds, once it uses the physical implementations of the NoC under test. Emulation is a process similar to prototyping, but with a physic connection between the host and the prototype through a subsystem denominated *emulator*. Emulators are composed of specific hardware and software to substitute one or more components in the system's prototype [CAL98].

Fast, cycle-accurate FPGA emulators have been used as an alternative to software simulators, as they reduce validation time without compromising accuracy. FPGA emulators allows the user to exploit the parallel nature of hardware, thereby enabling easier detection of design and architectural problems early and reducing the number of chip respins [LOT11].

2.5 The Original HardNoC Platform

The HardNoC system is a hardware platform to exercise with NoCs in general, first developed for the Hermes-2VC NoC [MEL05]. Both, the HardNoC and the Hermes-2VC NoC were developed and implemented by the GAPH research group at PUCRS. This Chapter describes the platform's hardware in detail and its use.

2.5.1 The HardNoC Block Diagram

The first version of the HardNoC hardware comprised six IP cores connected to a 6-router Hermes-2VC NoC, as illustrated in Figure 2.2. The circuit was originally designed to run in a Xilinx University Program Virtex-II Pro Development System (XUP-V2PRO) with a Xilinx Virtex-II Pro XC2VP30 FPGA. The platform can easily be adapted to other boards and instances of HardNoC with more routers and IP Cores. The HardNoC instance depicted in Figure 2.2 includes five traffic generators (TG) IPs, each implemented with two 16 Kbits Block RAMs, resulting in a memory capacity of 1024 32-bit words at each TG. Also, there is one RS-232 serial IP that provides bi-directional communication with a host processor. The external interface of the HardNoC system is very simple, and comprises four signals:

- reset, responsible to initialize the HardNoC system;
- clock, the basic synchronization signal;
- rxd, serial data, coming from the host computer to the HardNoC system;
- txd, serial data going from the HardNoC system to the host computer.

2.5.2 The HardNoC Packet Formats

Packet formats define a set of services offered by the host and communication network to all IP Cores in the system. The packet formats differ depending on the layer transmission is done, if it is packets transmitted by the NoC or through serial IP. The link layer encompasses a protocol employed to transmit information among the NoC routers, while the network layer includes different protocols to transmit data end-to-end between IP Cores or between IPs and the host.

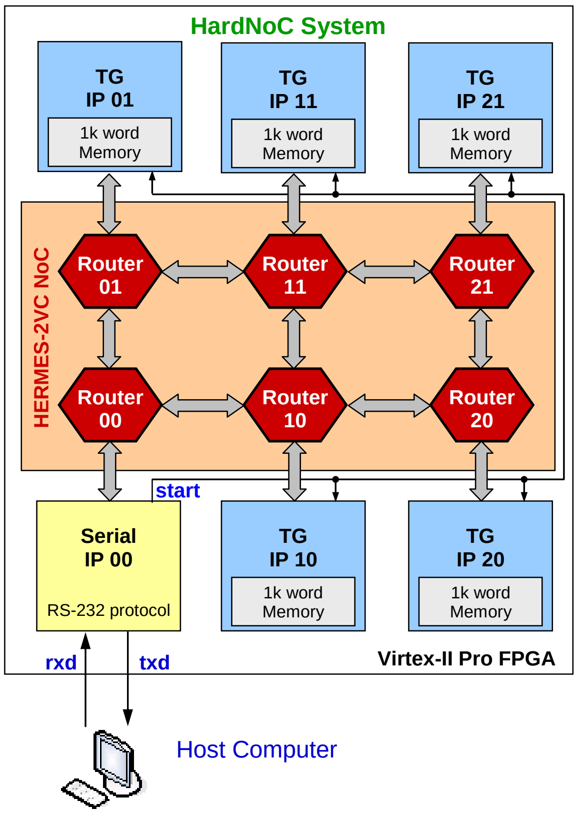


Figure 2.2 – The original HardNoC system block diagram.

2.5.3 Link Layer Packet Format

At the link layer, there is only one packet format, employed to transmit data within the NoC, between routers, as Figure 2.3 depicts. At this level, all the information needed for transmission is the target IP, the payload size and the payload data. Thus, the packet format at the link layer contains two control information in the first two flits, Target IP Core Address (destination address) and Payload Size.

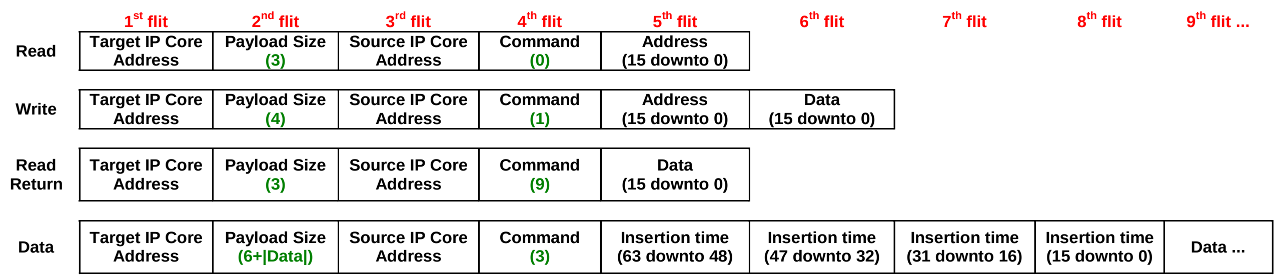
Link Layer (NoC) Packet Format

1st flit	2nd flit	3rd flit	...	(n+2)th flit
Target IP Core Address	Payload Size (N)	Payload Data		Payload Data

Figure 2.3 - The HardNoC link protocol layer and its packet format.

2.5.4 NoC Network Layer Packet Formats

At the network layer, it is necessary to keep information about the source IP and the action to take with the transmitted packet. Thus, packets between TGs need more control fields. See Figure 2.4.



	1st flit	2nd flit	3rd flit	4th flit	5th flit	6th flit	7th flit	8th flit	9th flit ...
Read	Target IP Core Address	Payload Size (3)	Source IP Core Address	Command (0)	Address (15 downto 0)				
Write	Target IP Core Address	Payload Size (4)	Source IP Core Address	Command (1)	Address (15 downto 0)	Data (15 downto 0)			
Read Return	Target IP Core Address	Payload Size (3)	Source IP Core Address	Command (9)	Data (15 downto 0)				
Data	Target IP Core Address	Payload Size (6+|Data|)	Source IP Core Address	Command (3)	Insertion time (63 downto 48)	Insertion time (47 downto 32)	Insertion time (31 downto 16)	Insertion time (15 downto 0)	Data ...

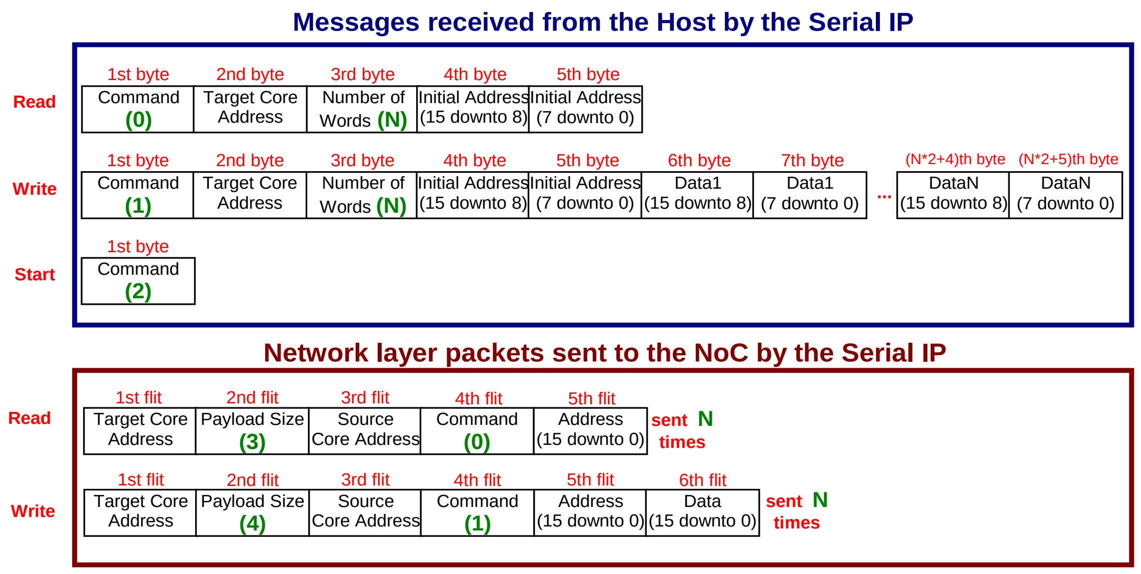
Figure 2.4 - HardNoC TG/Serial IP NoC network layer packet formats. In the Data packet format, |Data| stands for the size, in flits, of the variable size Data part of the packet, from the ninth flit on. Only the Serial IP generates the first two formats, and only TGs generate the last two formats.

Note that all TG packets are indeed NoC packets. A fixed binary code identifies each format uniquely in the Command flit. All packet formats have a fixed size, except for the Data format. The packet denominations/functions are:

1. Read (command 0) - typically used by the serial IP Core to request data from some TG;
2. Write (command 1) - typically used by the serial IP Core to store data at some TG;
3. Read Return (command 9) - typically used by a TG to return data from a read request to the Serial IP. The Source IP Address (first payload flit) in the return packet is the Target IP Address (first flit of the packet) in the read packet.
4. Data (command 3) - typically used to transmit a variable size data block between two TGs.

2.5.5 Network Layer System Packet Formats

The Serial IP Core is responsible for providing communication between the HardNoC host computer and the TGs of the system connected through the NoC. When the information comes from the host computer, the Serial IP creates a valid NoC network layer/link layer packet. When the Serial IP receives each 16-bit flit from the NoC it process this flit and sends it serially to the host computer. The packet formats can be seen in Figure 2.5 and Figure 2.6.



Messages received from the Host by the Serial IP

	1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte		(N*2+4)th byte	(N*2+5)th byte
Read	Command (0)	Target Core Address	Number of Words (N)	Initial Address (15 downto 8)	Initial Address (7 downto 0)					
Write	Command (1)	Target Core Address	Number of Words (N)	Initial Address (15 downto 8)	Initial Address (7 downto 0)	Data1 (15 downto 8)	Data1 (7 downto 0)	...	DataN (15 downto 8)	DataN (7 downto 0)
Start	Command (2)									

Network layer packets sent to the NoC by the Serial IP

	1st flit	2nd flit	3rd flit	4th flit	5th flit	6th flit	
Read	Target Core Address	Payload Size (3)	Source Core Address	Command (0)	Address (15 downto 0)	sent N times	
Write	Target Core Address	Payload Size (4)	Source Core Address	Command (1)	Address (15 downto 0)	Data (15 downto 0)	sent N times

Figure 2.5 - Messages received from the host computer by the Serial IP and corresponding packets sent to the NoC by the Serial IP.

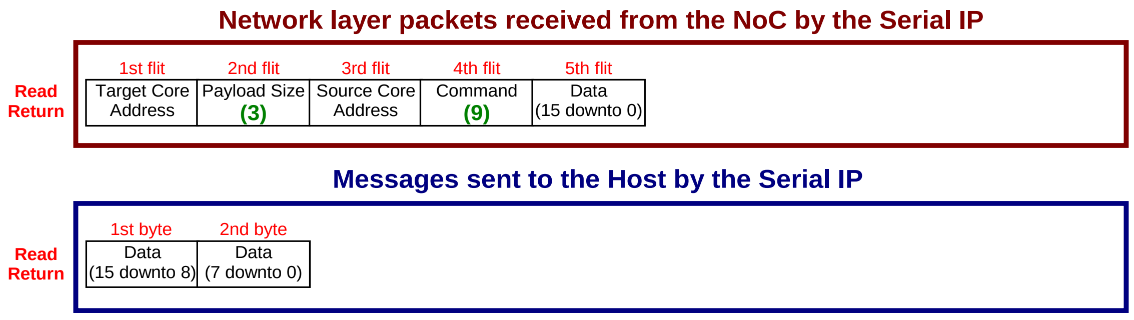


Figure 2.6 - Packets received from the NoC by the Serial IP and corresponding data sent to the Host computer.

The Serial IP accepts four commands. The host computer generates three of these: *(i)* read, *(ii)* write and *(iii)* start. The upper part of Figure 2.5 presents the byte structure of the commands sent by the host computer, while the lower part displays the flits structure of packets assembled and sent to the NoC by the Serial IP as an effect of the commands received from the host. The start command does not generate any traffic on the NoC itself. Instead, it activates the global start signal from the Serial IP to all TGs, synchronizing the start of the HardNoC traffic generation process. A global signal is necessary because all TGs must start their traffic generation process at the same time. The fourth command accepted by the Serial IP comes from the NoC and is called *(iv) Read Return*. This command contains the response to a read command generated by the host. The upper part of Figure 2.6 presents the format of the packet coming from the NoC, while the lower part displays the corresponding message sent to the host computer by the serial IP as a consequence of processing the read return command.

2.5.6 The TG IP Core

The TG IP is responsible for generating packets, sending these to the NoC, receiving packets from other cores connected to the NoC and computing packet latency values. The TG IP contains a memory built with two Xilinx Block RAM modules, each configured as a 2048x8-bit block, and control logic to arbitrate the access to the memory banks. The access to both memory banks proceeds in parallel always in a same address reading and writing 16-bit words.

Although the TG IP accepts three command types, the read, write and data, all discussed in earlier sections, only the data command occur during NoC emulation, once it is used to transmit data packets between TGs. When a TG receives a data packet, it calculates its latency subtracting the time the packet should enter the NoC (the packet generation time, which comes with the data packet) from the time when the last flit of the packet has left. The target TG IP captures this last information. The latency values of all packets are accumulated in four 64-bit words at the end of the TG’s memory.

The TG’s memory is divided into two parts. The last 16 Block RAM positions are reserved for the latency values storage while the rest is left to the information the TG needs to generate the traffic as defined by the user.

Traffic parameters are stored in memory and depend on the chosen traffic distribution: Uniform or Pareto. The traffic distribution type is stored in the first position of the TG memory (address 0), where a 0 value indicates Uniform distribution and a 1 value indicates Pareto distribution. The second position (address 1) in the TG memory informs the number of times to generate that traffic. The other parameters depend on the adopted traffic distribution and provide information about flow priority, flow target, packet size, number of packets, among others.

2.5.7 Using HardNoC

Using the HardNoC is simple. First, a sync signal must be sent to the system to synchronize the speeds of the Serial IP and the host computer. After that, the TGs must be programmed with the available options of traffic before they can generate traffic. Each TG must be programmed individually. This process corresponds to sending messages from the host computer to the TGs containing the information necessary for the TG to generate traffic once the system is started. This information is stored at the beginning of the TG's memory.

After all the TGs are programmed, the start command activates the traffic generation process in all the TGs that must be synchronized to calculate meaning fully related latency values. At this moment, each TG generates its own random data information and sends it to the target IP through the NoC accordingly to the traffic distribution pattern previously programmed.

Each time a packet should enter the NoC, the global time counter value is included in the packet. Every time the last flit of a packet leaves the NoC, the target IP samples the global time, subtracts the insertion time presented in the payload of the packet and accumulates this value properly.

HardNoC collects only the latency of each packet and accumulates them. The messages total latency or the comparison of the packets latency must be done by the user in the host computer. To get the data out of the TG's memory and into the host computer, a read command must be individually given to each TG.

2.5.8 HardNoC-GLP

The original HardNoc is a fully synchronous system. To emulate instances of the HardNoC-GLP NoC it was necessary to adapt the design of HardNoC, which produced the HardNoC-GLP platform. The needed changes involved the support to two input clock frequencies, the insertion of bi-synchrounous FIFOS at the TGs and the adaptation of the Serial IP module and TGs structure.

A first prototype available for this work employs a single clock source that passes through a Digital Clock Management (DCM) component of the target FPGA. This component has several clock outputs, from which two can be used. To allow for significantly different clocks, the initial prototype used a 100MHz clock that feeds the FPGA DCM. The DCM has been parameterized to generate a **divide by 3** output clock, producing a 33.33MHz clock. Also, the **2x** DCM output was used as a second clock. Thus, the two clock frequencies available for each router are 33.33MHz and 200MHz. The TGs and the Serial IP used one of these clock sources, too.

The HardNoC-GLP was chosen to do this work because it is the last version of HardNoC that was prototyped and completely validated in hardware by GAPH research group. However, in this work will not be demonstrated the GLP concepts. The first prototype of the HardNoC-GLP platform used a Xilinx ML505 Virtex5 Evaluation platform. This is the platform used for developing the present work. Mention in the rest of the work to HardNoC refers to the HardNoC-GLP or to both HardNoC and HardNoC-GLP.

3. RELATED WORK

The time consumed in the NoC validation process during a chip design has become an important constraint, because the demand for reduced time-to-market plays a major role in chip design. Several techniques and studies exist in this area, most intended to reduce the consumed validation time. This Chapter discusses two typical approaches to deal with this problem.

3.1 HW/SW Emulation Platforms

Some recent research aims to make the NoCs validation process by emulation faster and more flexible, by mixing the speed of hardware and the flexibility of software. Accordingly, some platforms composed of hardware and software emerged, which enhanced speed and accuracy of software simulations. In [GEN05] and [LOT11] the respective authors present alternative platforms to emulate NoCs based on the concept of a hybrid SW/HW emulator system.

The platform in [LOT11] is centered on a hybrid framework composed of FPGA hardware resources as the hardware part and the Xilinx Micro Blaze soft core processor as the software part. The FPGA contains all needed hardware structures, such as the entire NoC and its connections. The software framework, on his turn, contains the traffic generators, source queues, traffic receptors and the emulation clock generator.

The main advantage of such an approach is the mixing of the FPGA hardware to simulate at speed with the software flexibility to generate and analyze traffic. The weakness is the tight coupling of software and hardware. The use of a shared bus to connect the software and hardware frameworks reduces the speed of emulation. Bus congestion affects results, once each framework must communicate with the other during the emulation process.

The hybrid framework presented in [GEN05] uses an FPGA board with an embedded Power PC. The FPGA contains the NoC, the traffic generators and receptors, a control unit and a filter unit that manages the decoding of the address bus. The software part of the framework deals with the programming of traffic generators and receptors and traffic analysis.

The main advantage here is the use of the speed of the hardware and the flexibility of the software, once is it possible to include traces of traffic generated by real applications, in the NoC that is being emulated. The limitation may be the use of FPGA space, because there are several complex modules, such as the traffic generators. This limits the size of the NoC to emulate.

Wen et al. [WEN09] describe a scheme to configure traffic generators before or during NoC emulation. They call it OCTG (On-line Configurable Traffic Generator), which is directly configured through a JTAG interface, in order not to affect the NoC, if it is actively producing traffic. This NoC emulation system is quite similar to HardNoC.

Tan et al. [TAN11] demonstrate a generic design flow for NoC exploration on FPGAs. The authors used ATLAS to generate the Hermes NoC and introduced IPs for traffic generation and traffic reception. These TGs and TRs are different from de HardNoC version because they are separated IPs: a TG can only inject traffic and a TR can only receive traffic to compute latencies. Thus, a limitation is that before prototyping it is necessary to know the traffic scenery that will be emulated to create the TGs/TRs distribution.

3.2 Emulating Different NoCs without Resynthesis

Another approach to reduce NoC's validation time is to concentrate in fast context switching. The idea of making an almost generic platform that could emulate a large variety of NoCs by simply programming the system to accept another NoC type without synthesizing it every time was the aim of some research.

The time to synthesize a NoC design can be in the order of hours, depending on the desired network topology and size. Thus, when several network topologies need to be tested, the time consumed by synthesis can be unsatisfactorily large. In [KRA08] and [WAN10], the authors demonstrate how to emulate different NoCs without the need to resynthesize each time the whole system. Some options were implemented in FPGA that allow reconfiguring the NoCs under software control.

The possibility to configure each router connection also brings the possibility to explore alternative interconnections between routers (i.e. to producxe alternative interconnection topologies at runtime). However, this requires more FPGA area and the reported speed up is just between 80 to 300 times in comparison with software simulation.

4. The HNPlus Platform

The limitations of the HardNoC platform include the availability of a fixed number of traffic distribution patterns. Also, the platform is difficult to operate and results analysis has to be done manually. This motivated the proposition of the HNPlus (from HardNoC Plus) platform that brings two improvements to HardNoC: *(i)* a new scheme to generate and inject traffic into the NoC and *(ii)* a new software to control the platform from a host.

In HNPlus, it is possible to generate virtually any traffic pattern of both, synthetic distributions and real traffic. To achieve these goals, the HardNoC's original TG was modified to be able to read generic traffic descriptions from its local memory and to inject it into the NoC, without worrying about traffic distribution computation during emulation. The flexibility in traffic types come from integration with the ATLAS environment, which is capable to generate several types of traffic distribution models. ATLAS generates traffic files and these may be simply written in the new TG memory before emulation. Making the TG independent of the used traffic distribution type makes traffic generation much simpler and makes the HNPlus platform more flexible and powerful, because it is possible to generate virtually any kind of traffic.

The next sections describe the modifications made in the original HardNoC platform to create HNPlus, always comparing and explaining the obtained advantages.

4.1 TG

Figure 4.1 and Figure 4.2 respectively display the original modules and the modules modified of the TGs. The new TG comprises traffic injector and traffic collector cores that are able to transmit and receive packets to and from the NoC. Both operate the same way the original HardNoC's TGs does. However, some modifications in the traffic receptor, traffic generator and also in the traffic parameters written into Block RAM compose the upgrades that make HNPlus more generic. The external interfaces, clock cycles counter and simulation results remain the same as in HardNoC. The traffic generator module is no longer a generator, but just a traffic injector during emulation, since it just has to read packets from its memory and transmit it to the NoC, with just a few processing. The traffic receptor was adapted to receive some additional information and there is no need to write more traffic parameters into the local memory, only traffic packets.

The new traffic injector reads from its memory the entire packet information and injects into the NoC. This makes the traffic injector much simpler than the HardNoC traffic generator, but memory space becomes a limiting factor, because each packet consumes at least 16 bytes and the entire memory space is only 2048 bytes, so each TG can inject a maximum of 128 packets. Bigger local memories can alleviate this problem. Consideration of this problem was overlooked only due to lack of time.

The packet information that coordinates traffic injection appears in Figure 4.3 and its fields are as follows:

- **Packet Target:** Address of another TG that will be the packet target.
- **Payload Size:** Packet size in flits.

- **Insertion Time:** Number of clock cycles that the packet should wait before being injected in the NoC.
- **Packet Sequence Number:** Sequence number generated by ATLAS. Actually, this is not used here.
- **Packet Data:** It is the only optional field, because it is not necessary for emulation and may occupy much space in memory. However, it is necessary when injecting real data traffic.

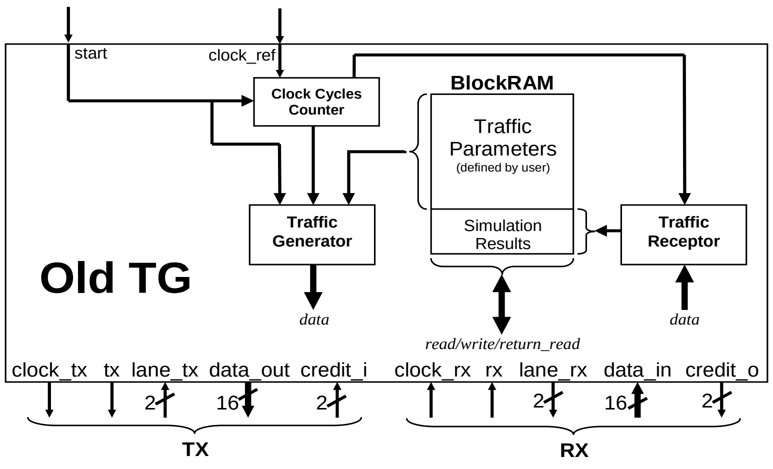


Figure 4.1 - The HardNoC TG structure.

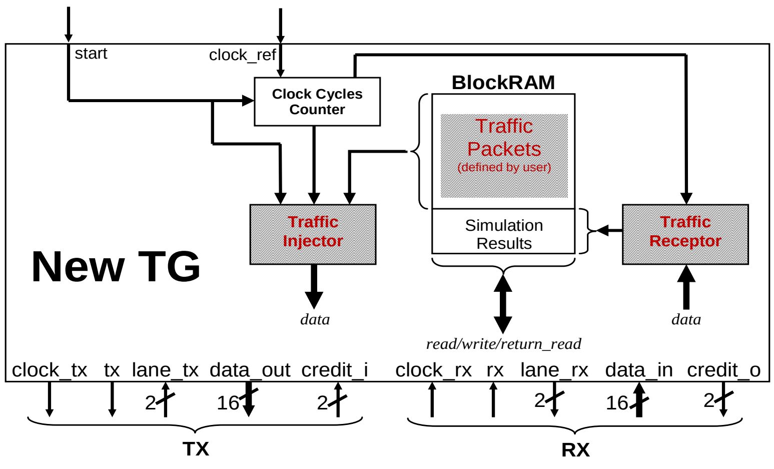


Figure 4.2 – The HNPlus TG structure.

	1st flit	2nd flit	3rd flit	4th flit	5th flit	6th flit	7th flit	8th flit
Packet	Target IP Core Address	Payload Size (Packet Size - 6)	Source IP Core Address	Creation time (63 downto 48)	Creation time (47 downto 32)	Creation time (31 downto 16)	Creation time (15 downto 0)	Sequence Number (31 downto 16)
	9st flit	10nd flit	11rd flit	12th flit	13th flit	14th flit	15th flit	16th flit …
	Sequence Number (15 downto 0)	Insertion time (63 downto 48)	Insertion time (47 downto 32)	Insertion time (31 downto 16)	Insertion time (15 downto 0)	Data (1)	Data (2)	Data (3)…

Figure 4.3 – The HNPlus Packet format.

Figure 4.4 exemplifies the new memory organization of the TG, showing how a traffic with 40 packets should be written, assuming no real data traffic. The first address is reserved to set some options to the traffic injector, such as traffic availability flag, traffic with real data flag and traffic priority. The traffic availability flag is set to indicate if there is at least one packet written in its memory. The traffic with real data flag indicates if the written packets have real data or if the traffic injector has to generate random data flits. The traffic priority contributes to the choice of the clock source that will be used during the transmission of these packets. In the example below, just the traffic availability flag is set to indicate that there is a valid traffic. It is also necessary to set a flag in address after sending the last packet to indicate the end of the traffic generation process by this TG. A packet target with value 0xFFFF is written in memory after all packets, to stop the transmission state machine. Thus, considering that the memory space is 1024 words of 16 bits, the first word is reserved to set the options, the last packet should have an invalid target and each packet consumes at least 8 words, so each TG can only inject 127 valid packets into the NoC for each emulation run.

Address	Data	Description	
0	1	Options	
1	01	Packet Target	
2	10	Payload Size	
3	0	Insertion Time (1º flit)	
4	0	Insertion Time (2º flit)	
5	0	Insertion Time (3º flit)	1st Packet
6	2	Insertion Time (4º flit)	
7	0	Packet Sequence Number (1º flit)	
8	1	Packet Sequence Number (2º flit)	
9	01	Packet Target	
10	10	Payload Size	
11	0	Insertion Time (1º flit)	
12	0	Insertion Time (2º flit)	2nd Packet
13	0	Insertion Time (3º flit)	
14	15	Insertion Time (4º flit)	
15	0	Packet Sequence Number (1º flit)	
16	2	Packet Sequence Number (2º flit)	
…	…	…	
321	0xFFFF	Packet Target	

Figure 4.4 – Memory organization example, describing a traffic with 40 packets, without real data.

Figure 4.5 demonstrates how the same traffic of Figure 4.4 should be written into the memory but this time considering a real data traffic. The options must indicate that there is a real data to be sent with each packet, and this data must be written according to the payload size, after the packet sequence number.

Address	Data	Description	
0	3	Options	
1	01	Packet Target	
2	10	Payload Size	
3	0	Insertion Time (1º flit)	
4	0	Insertion Time (2º flit)	
5	0	Insertion Time (3º flit)	
6	2	Insertion Time (4º flit)	1st Packet
7	0	Packet Sequence Number (1º flit)	
8	1	Packet Sequence Number (2º flit)	
9	8	Data	
10	9	Data	
11	10	Data	
12	01	Packet Target	
13	10	Payload Size	
14	0	Insertion Time (1º flit)	
15	0	Insertion Time (2º flit)	
16	0	Insertion Time (3º flit)	
17	15	Insertion Time (4º flit)	2nd Packet
18	0	Packet Sequence Number (1º flit)	
19	2	Packet Sequence Number (2º flit)	
20	8	Data	
21	9	Data	
22	10	Data	
…	…	…	
441	0xFFFF	Packet Target	

Figure 4.5 - Memory organization example, describing traffic with 40 packets containing real data traffic.

In this case, it is very important to set the correct payload size, to avoid a wrong operation of the transmission state machine, because it is based on the payload size that this state machine knows where each packet begins and ends. The packet size is the number of flits to inject, transmitting all information and data of each packet. The payload size is defined by the packet size minus six, which is calculated by ATLAS during traffic generation, so if the payload size is ten, sixteen flits will be sent.

The Transmission State Machine is responsible for everything that the TG sends to the NoC. Figure 4.6 shows that it is composed of three operations: Return Packet, Pre Reading and Sending Packets. Return Packet activates after the TG receives a read command, when it is necessary to respond with requested data. Pre Reading is activated when the TG has been programmed with some traffic. Then, it is necessary to be ready to transmit a packet before it receives the start signal. Sending packets activates when the TG receives the start signal. It is during this operation that the traffic is injected into the NoC.

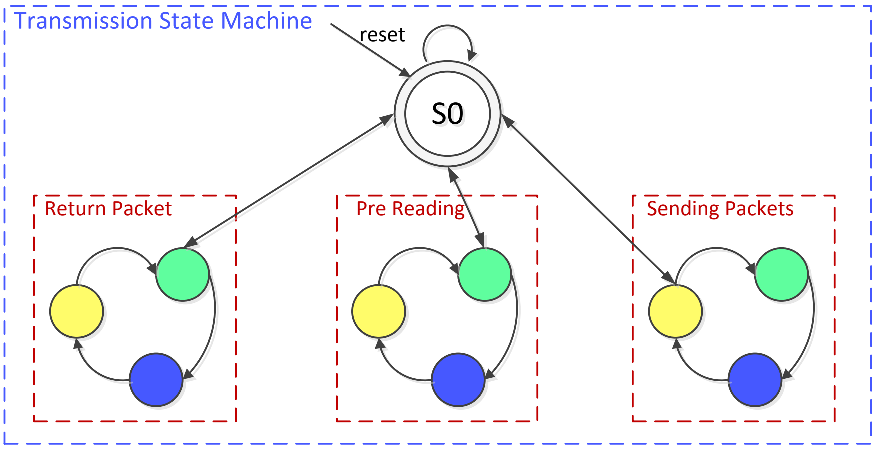


Figure 4.6 – Operating modes of Transmission State Machine.

The Return Packet operation is responsible for answering to a read command. When the TG receives this command, the Transmission State Machine starts to transmit the requested data. See Figure 4.7. This set of states were not modified, it is the same as in the HardNoC's TG.

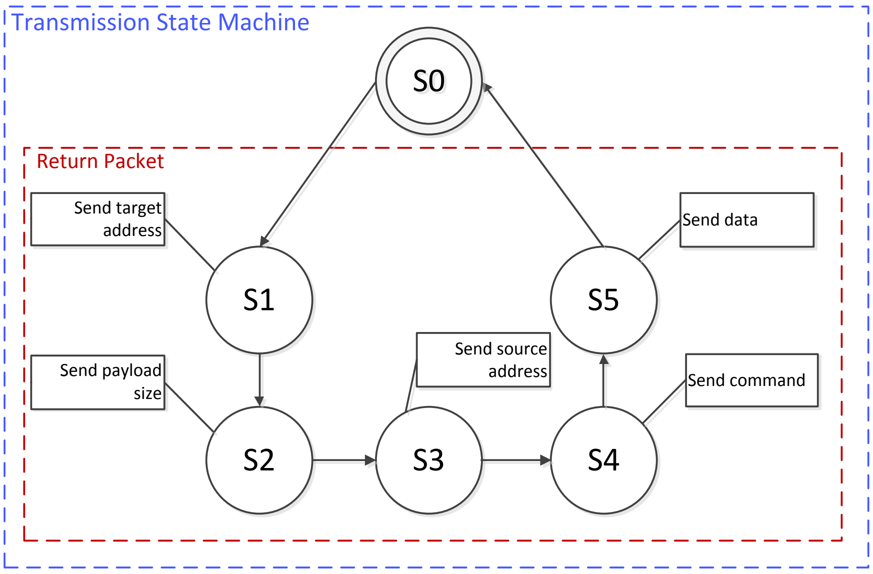


Figure 4.7 – Return Packet States.

The Pre Reading operation is responsible for preparing the TG to be ready to inject its first data packet into the NoC. While the traffic file is being written into TG memory, this operation is activated after the first packet is written. Thus, pre reading of the first packet always occurs. After this, the TG is ready to start traffic injection when it receives the start signal. Figure 4.8 shows all pre reading operation states.

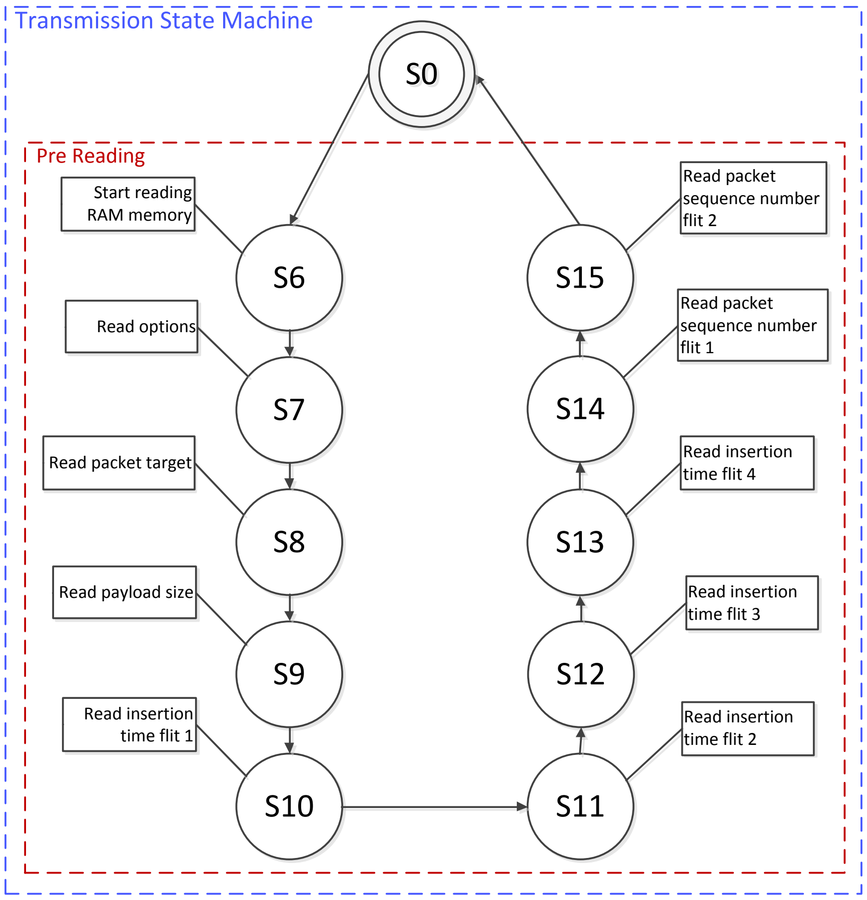
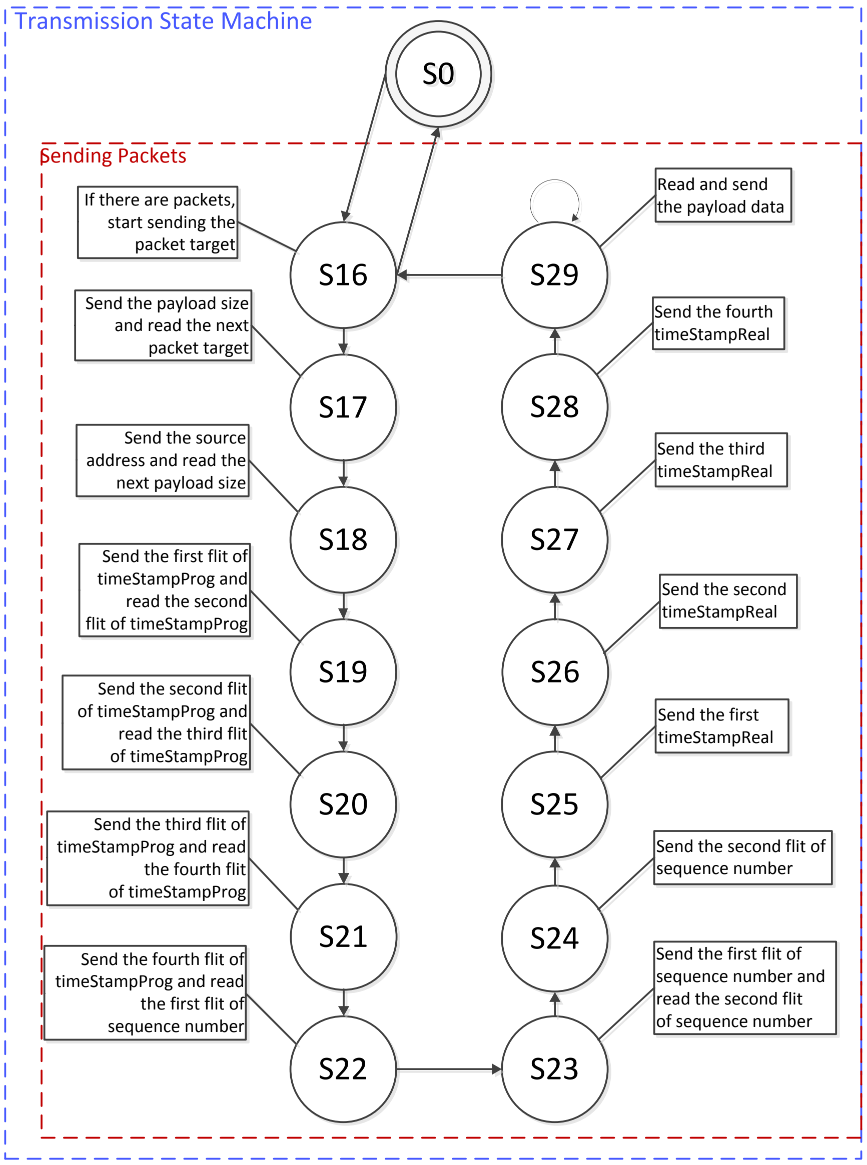

Transmission State Machine
S0
Pre Reading
Start reading RAM memory
S6
Read options
S7
Read packet target
S8
Read payload size
S9
Read insertion time flit 1
S10
S11
Read insertion time flit 2
S12
Read insertion time flit 3
S13
Read insertion time flit 4
S14
Read packet sequence number flit 1
S15
Read packet sequence number flit 2

Figure 4.8 – Pre Reading States.

The Sending Packets operation is responsible for injecting traffic packets into the NoC. When the TG receives the *start* signal, the transmission state machine starts injecting the packet read during the pre reading operation, so that, if necessary, it is possible to start injecting a packet in the first simulation clock cycle. This state machine works as a pipeline. While a packet is injected, the next packet is being read, such that when a packet ends to be injected, the next will be ready to start injection. Thus, packets can be injected without wasting clock cycles if the traffic pattern dictates so.



Transmission State Machine
S0
Sending Packets
If there are packets, start sending the packet target
S16
Send the payload size and read the next packet target
S17
Send the source address and read the next payload size
S18
Send the first flit of timeStampProg and read the second flit of timeStampProg
S19
Send the second flit of timeStampProg and read the third flit of timeStampProg
S20
Send the third flit of timeStampProg and read the fourth flit of timeStampProg
S21
Send the fourth flit of timeStampProg and read the first flit of sequence number
S22
S23
Send the first flit of sequence number and read the second flit of sequence number
S24
Send the second flit of sequence number
S25
Send the first timeStampReal
S26
Send the second timeStampReal
S27
Send the third timeStampReal
S28
Send the fourth timeStampReal
S29
Read and send the payload data

Figure 4.9 – Sending Packet States.

The Reception State Machine is responsible for receiving commands from the serial IP and computing statistics results of the packets it receives. A few modifications were made in the traffic receptor module, as Section 2.5 discusses. There were four command types (*read*, *write*, *read return* and *data*). The new traffic receptor module has only three command types: *read*, *write* and *read return*. The *data* command is no more a command but just a packet type that is sent between TGs during NoC emulation. This modification was necessary to make traffic injector and receptor as similar as possible to the behavior observed in the ATLAS environment. Thus, the packet information sent by a traffic injector is the same that is sent during an ATLAS simulation. The HNPlus traffic receptor continues to calculate latencies that were calculated by the HardNoC traffic receptor in the same way.

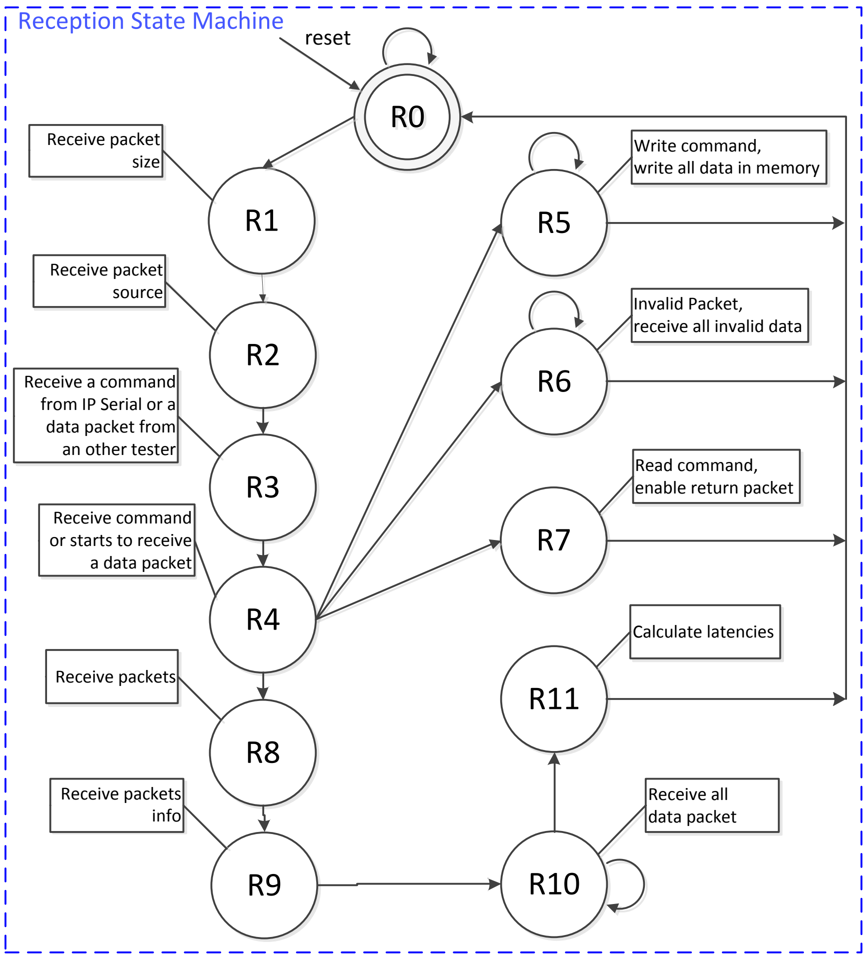
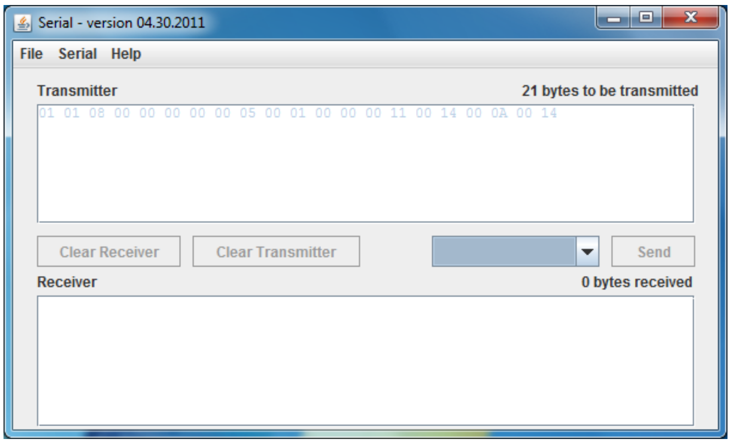

Reception State Machine
reset
R0
Receive packet size
R1
Receive packet source
R2
Receive a command from IP Serial or a data packet from an other tester
R3
Receive command or starts to receive a data packet
R4
Receive packets
R8
Receive packets info
R9
Write command, write all data in memory
R5
Invalid Packet, receive all invalid data
R6
Read command, enable return packet
R7
Calculate latencies
R11
Receive all data packet
R10

Figure 4.10 – Reception State Machine.

4.2 Software

The original HardNoC software interface appears in Figure 4.11. It contains some important limitations: the configuration and programming of traffic generators, for example, is done manually and has not a friendly and intuitive interface. The following modifications were made to solve this problem, keeping the interface compatible with modifications conducted in the hardware. Beginning with the graphical interface, it was completely remodeled, showing more clearly how the information necessary to operate the platform is structured, and not just be a plain serial interface for sending/receiving bytes. In addition, the new version of the program can open, read and modify the traffic files generated by ATLAS before sending them to the HNPlus Platform. Figure 4.12 illustrates the new interface.



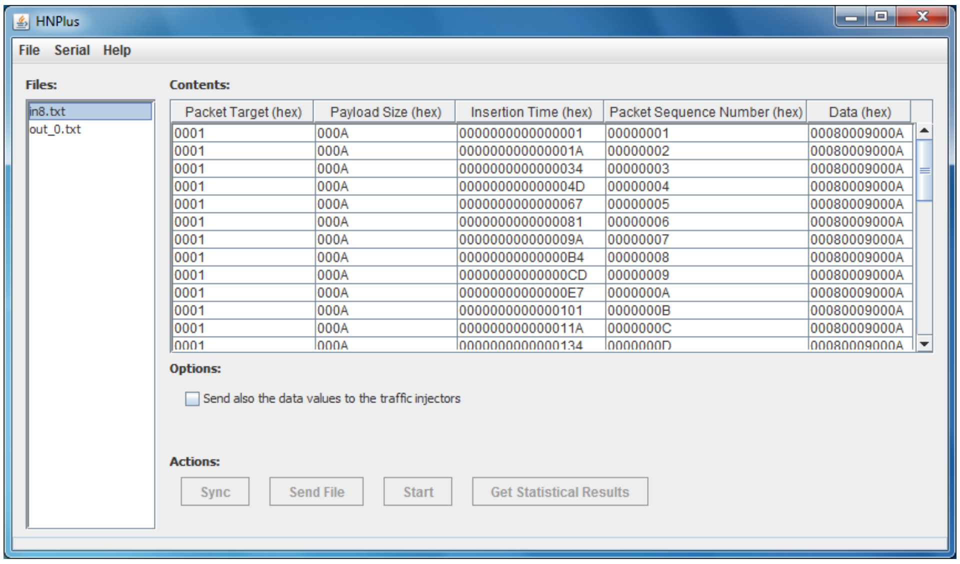
Transmitter — 21 bytes to be transmitted

```
01 01 08 00 00 00 00 00 05 00 01 00 00 00 11 00 14 00 0A 00 14
```

Receiver — 0 bytes received

Figure 4.11 – HardNoC, old host software.

The HNPlus software was developed in Java for portability. In addition, the original HardNoC software was implemented in Java and parts of its code could be reused. Thus, there were no major reasons to justify the choice of another programming language to develop the new software. There are several tools to help the software development available, especially in high-level languages and the Eclipse Java IDE was chosen because it provides tools that facilitate software development, especially in the creation of graphical interfaces, and the authors dominate this IDE.



Files: in8.txt, out_0.txt

Contents:

Packet Target (hex)	Payload Size (hex)	Insertion Time (hex)	Packet Sequence Number (hex)	Data (hex)
0001	000A	0000000000000001	00000001	00080009000A
0001	000A	000000000000001A	00000002	00080009000A
0001	000A	0000000000000034	00000003	00080009000A
0001	000A	000000000000004D	00000004	00080009000A
0001	000A	0000000000000067	00000005	00080009000A
0001	000A	0000000000000081	00000006	00080009000A
0001	000A	000000000000009A	00000007	00080009000A
0001	000A	00000000000000B4	00000008	00080009000A
0001	000A	00000000000000CD	00000009	00080009000A
0001	000A	00000000000000E7	0000000A	00080009000A
0001	000A	0000000000000101	0000000B	00080009000A
0001	000A	000000000000011A	0000000C	00080009000A
0001	000A	0000000000000134	0000000D	00080009000A

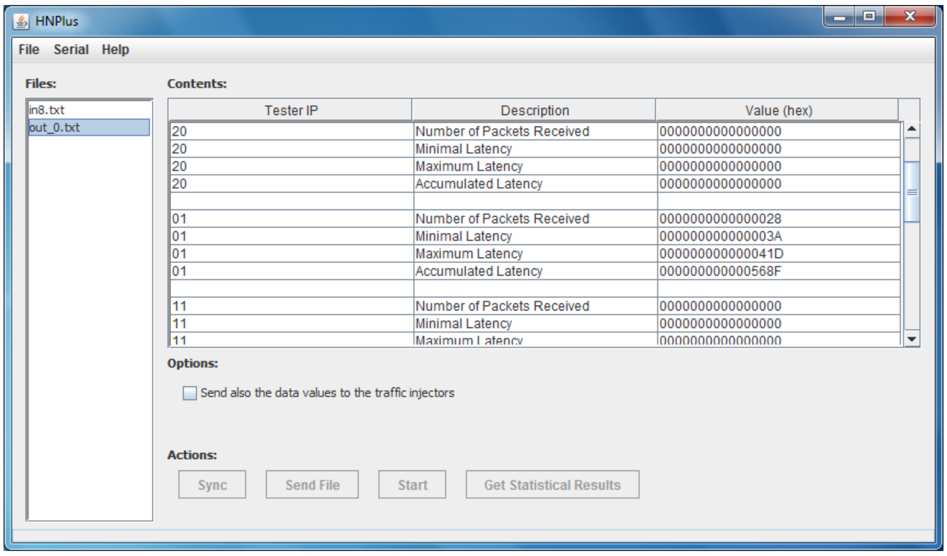
Options:

☐ Send also the data values to the traffic injectors

Actions:

Figure 4.12 – The HNPlus host software, showing contents of the traffic file generated by ATLAS.

The new software interface is more user-friendly, intuitive and easier to use. In simple steps, it is possible to program the HNPlus platform and start NoC emulation. The first step is to open all traffic files generated by ATLAS and, when some file is selected, all of its contents are shown, to easily check if it is the correct file. The second step is to connect to the serial port that should be linked to the HNPlus platform and then send all desired files to program the HNPlus TGs. The third step is to send the start signal, clicking on the start button. The fourth step is to request to all TGs the statistical results, so it is possible to verify if latencies are as expected and to confirm that all packets were transmitted between TGs in the emulation session.



Tester IP	Description	Value (hex)
20	Number of Packets Received	0000000000000000
20	Minimal Latency	0000000000000000
20	Maximum Latency	0000000000000000
20	Accumulated Latency	0000000000000000
01	Number of Packets Received	0000000000000028
01	Minimal Latency	000000000000003A
01	Maximum Latency	000000000000041D
01	Accumulated Latency	000000000000568F
11	Number of Packets Received	0000000000000000
11	Minimal Latency	0000000000000000
11	Maximum Latency	0000000000000000

☐ Send also the data values to the traffic injectors

Figure 4.13 – The HNPlus host software, showing the obtained statistical results.

4.3 The HNPlus Platform Operation

From the user viewpoint, the HNPlus platform operates almost in the same way the HardNoC platform did. The main differences to the user are in the interface software, through which emulation configuration and results can be better analyzed.

HNPlus offers an intuitive software interface to the user. The numerated arrows in Figure 4.14 indicate the HNPlus basic flow for a simple example, with just one TG sending data to another. First, the user uploads the traffic files generated with ATLAS to each TG (Step 1, in this case only one file is sent to a single TG). Here, the user also defines the use of real traffic for emulation, already informed in the traffic file, or a synthetic traffic that can be generated at runtime. The traffic file is sent by the host to the Serial IP (Step 2), which transfers it to the correct TG 12 using the NoC infrastructure (Step 3). The TG is now programmed and emulation starts with a command in the software interface. During emulation, TG 12 injects traffic into the NoC according the configuration written in its memory (Step 4). After the emulation process finishes, a new command in the software interface asks for the results stored in the TG that received traffic (Step 5). Again, the Serial IP transfers the command to the correct TG that returns the latency information stored in the memory of TG 02 (Step 6). The Serial IP transfers it to the software interface (Step 7) for user analysis.

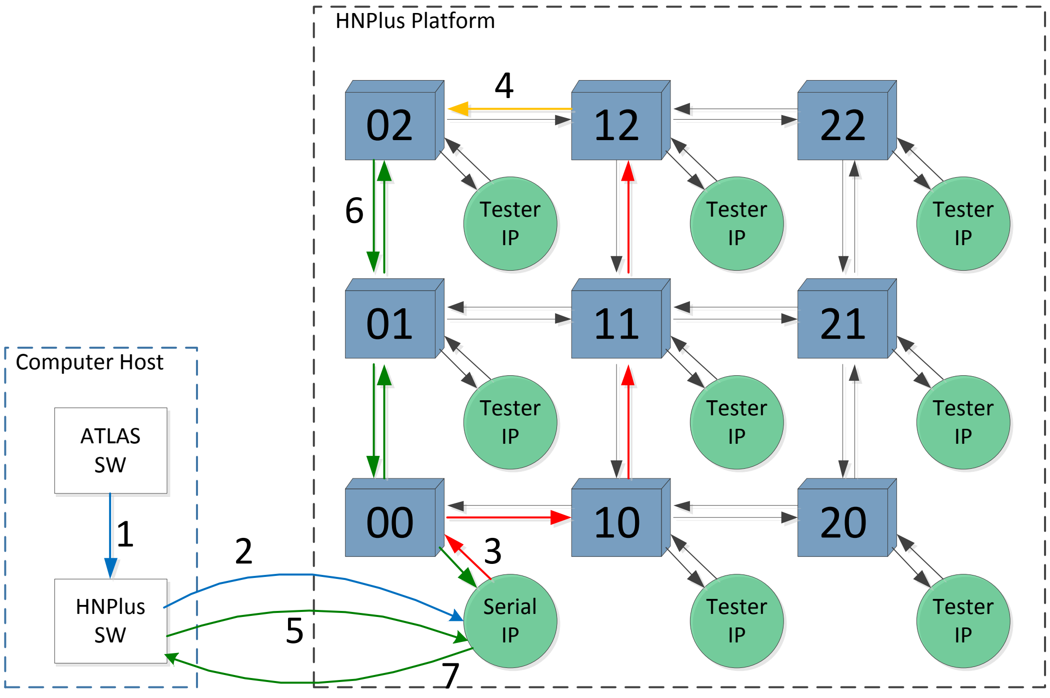


Figure 4.14 – HNPlus platform operation overview.

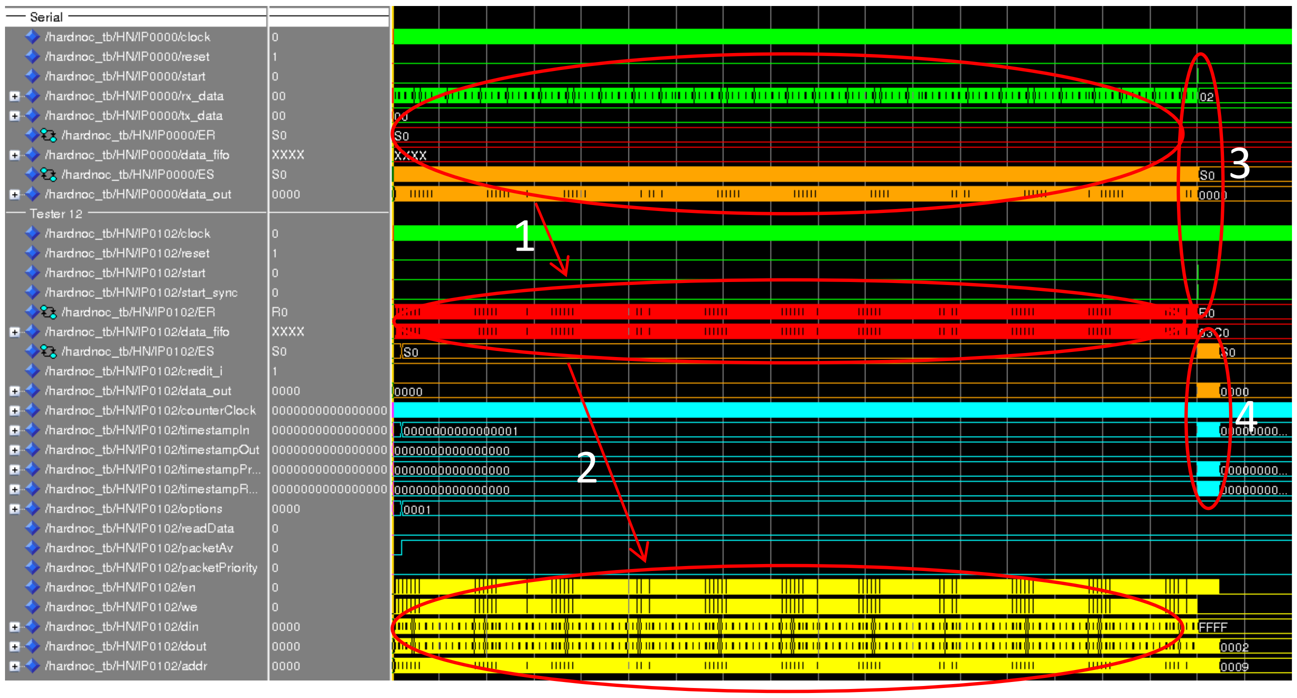
5. VALIDATION, PROTOTYPING AND EXPERIMENTATION

This Section describes a series of simulations conducted to validate the HNPlus platform, some prototyping details and some experiments that show the correct operation of the new platform. Section 5.1 presents the validation of the new platform, showing in detail its operation. Section 5.2 presents a prototyping analysis comparing HardNoC and HNPlus hardware synthesis. Finally, Section 5.3 presents some experiments with the new platform.

5.1 Overview of HNPlus Operation

Figure 5.1 presents an overall simulation waveform, going from TG programming until the end of the emulation process enabled by the start signal. The relevant details are numbered and commented in the respective items below:

1. The circle marked with number 1 shows the Serial IP sending all the write commands to program the TG with its specific traffic. Each write command leads to a new reception in the TG 12 receive machine state.
2. The previous action (1) enables the write in the TG 12 memory. Circle number 2 shows the traffic file being written in memory.
3. The circle marked with number 3 shows the Serial IP sending the start signal. This signal synchronizes all TGs internal clocks and starts the emulation process.
4. Here occurs the actual emulation process, with the injection of all packets from the TG 12 to TG 02 through the NoC.



```
Serial
/hardnoc_tb/HN/IP0000/clock             0
/hardnoc_tb/HN/IP0000/reset             1
/hardnoc_tb/HN/IP0000/start             0
/hardnoc_tb/HN/IP0000/rx_data           00
/hardnoc_tb/HN/IP0000/tx_data           00
/hardnoc_tb/HN/IP0000/ER                S0
/hardnoc_tb/HN/IP0000/data_fifo         XXXX
/hardnoc_tb/HN/IP0000/ES                S0
/hardnoc_tb/HN/IP0000/data_out          0000
Tester 12
/hardnoc_tb/HN/IP0102/clock             0
/hardnoc_tb/HN/IP0102/reset             1
/hardnoc_tb/HN/IP0102/start             0
/hardnoc_tb/HN/IP0102/start_sync        0
/hardnoc_tb/HN/IP0102/ER                R0
/hardnoc_tb/HN/IP0102/data_fifo         XXXX
/hardnoc_tb/HN/IP0102/ES                S0
/hardnoc_tb/HN/IP0102/credit_i          1
/hardnoc_tb/HN/IP0102/data_out          0000
/hardnoc_tb/HN/IP0102/counterClock      0000000000000000
/hardnoc_tb/HN/IP0102/timestampIn       0000000000000000
/hardnoc_tb/HN/IP0102/timestampOut      0000000000000000
/hardnoc_tb/HN/IP0102/timestampPr...    0000000000000000
/hardnoc_tb/HN/IP0102/timestampR...     0000000000000000
/hardnoc_tb/HN/IP0102/options           0000
/hardnoc_tb/HN/IP0102/readData          0
/hardnoc_tb/HN/IP0102/packetAv          0
/hardnoc_tb/HN/IP0102/packetPriority    0
/hardnoc_tb/HN/IP0102/en                0
/hardnoc_tb/HN/IP0102/we                0
/hardnoc_tb/HN/IP0102/din               0000
/hardnoc_tb/HN/IP0102/dout              0000
/hardnoc_tb/HN/IP0102/addr              0000
```

Figure 5.1 – An overview of a complete simulation of the HNPlus platform: Serial IP programs the TG, followed by start signal generation and the emulation process.

5.2 Validation

The HNPlus platform improvements were validated in simulation using the Modelsim 10.0c simulator of Mentor Graphics. A testbench was used to simulate a traffic file generated by ATLAS for the same scenario of the Figure 4.14. The rest of this Section describes a simulation overview and details of the operation of the new features in HNPlus.

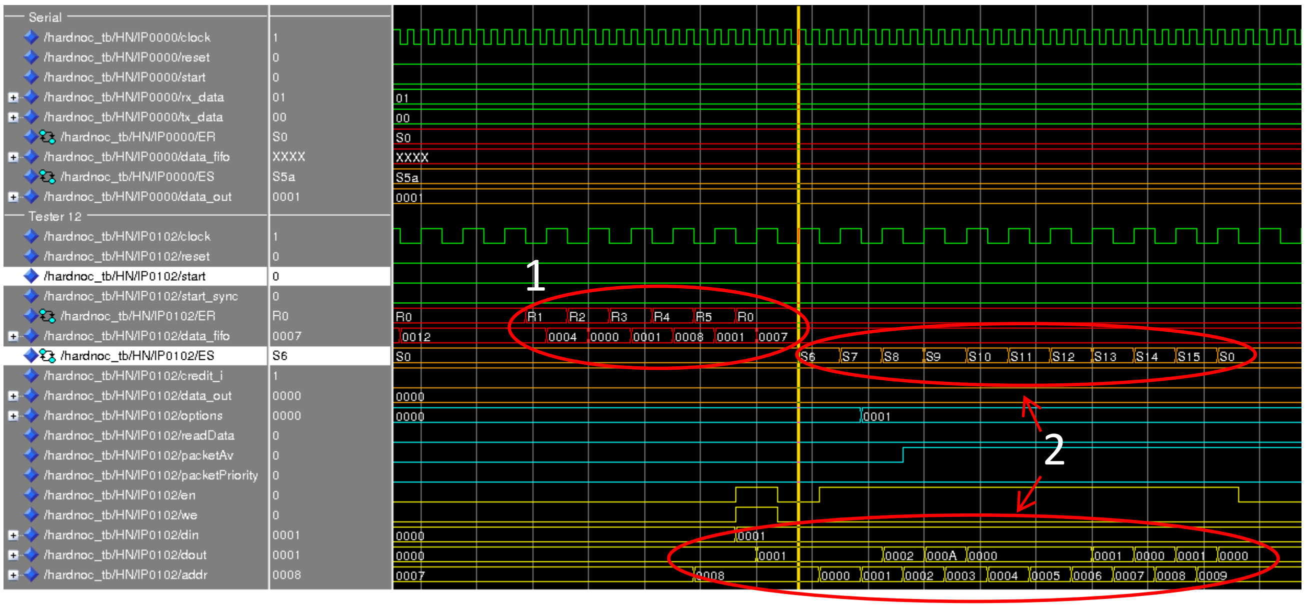
5.2.1 TG Programming and Pre Reading

Figure 5.2 shows an important step in programming the traffic of a TG. The pre reading event occurs during the writing of the first packet of the traffic file in the TG's memory. The relevant details are numbered in the simulation waveform and are commented in the respective items below:

1. The circle marked with number 1 shows the state sequence that writes traffic information in the TG memory and the associated information. Here, TG 12 is receiving the last word that composes the first packet of traffic file, which corresponds to the last word of the header information for the first packet. This word always occupies memory address 0x0008 of the local Block RAM, as Figure 4.4 and Figure 4.5 show. This can be identified when the TG receives a write command from the Serial IP to this address.
2. The previous action (1) activates the pre reading of the first packet, shown in circles marked with 2. The pre reading of all header information of the first packet is made to allow the TG to be able to start injecting traffic in the first emulation clock cycle, if necessary, once this information is stored in internal registers. This would not be possible if the header information

still had to be read after the start command, because the cycles needed to execute this task would delay the injection time and consequently increase transmission latency beyond the predicted insertion time. Besides the pre reading states S6 to S15, the TG's memory signals are also marked with 2 and it is possible to see the addressing and the data read in each pre reading state.

Pre reading occurs between two Serial IP write commands, in addresses 0x0008 and 0x0009 respectively. This is possible because a serial transmission is much slower than the reading of the header information from the memory. If this constraint is not respected, an addressing error may occur.



Serial
/hardnoc_tb/HN/IP0000/clock
/hardnoc_tb/HN/IP0000/reset
/hardnoc_tb/HN/IP0000/start
/hardnoc_tb/HN/IP0000/rx_data
/hardnoc_tb/HN/IP0000/tx_data
/hardnoc_tb/HN/IP0000/ER
/hardnoc_tb/HN/IP0000/data_fifo
/hardnoc_tb/HN/IP0000/ES
/hardnoc_tb/HN/IP0000/data_out
Tester 12
/hardnoc_tb/HN/IP0102/clock
/hardnoc_tb/HN/IP0102/reset
/hardnoc_tb/HN/IP0102/start
/hardnoc_tb/HN/IP0102/start_sync
/hardnoc_tb/HN/IP0102/ER
/hardnoc_tb/HN/IP0102/data_fifo
/hardnoc_tb/HN/IP0102/ES
/hardnoc_tb/HN/IP0102/credit_i
/hardnoc_tb/HN/IP0102/data_out
/hardnoc_tb/HN/IP0102/options
/hardnoc_tb/HN/IP0102/readData
/hardnoc_tb/HN/IP0102/packetAv
/hardnoc_tb/HN/IP0102/packetPriority
/hardnoc_tb/HN/IP0102/en
/hardnoc_tb/HN/IP0102/we
/hardnoc_tb/HN/IP0102/din
/hardnoc_tb/HN/IP0102/dout
/hardnoc_tb/HN/IP0102/addr
1
2
R0 R1 R2 R3 R4 R5 R0
0012 0004 0000 0001 0008 0001 0007
S0 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S0
0001 0002 000A 0000 0001 0000 0001 0000
0007 0008 0000 0001 0002 0003 0004 0005 0006 0007 0008 0009

Figure 5.2 – TG receiving the last header information of the first packet from Serial IP and activation of pre reading.

5.2.2 Start of HNPlus Emulation

Figure 5.3 shows three relevant details of the new HNPlus platform operation, all numbered and commented in the items below:

1. The circle marked with number 1 shows the Serial IP transmitting the start signal to all TGs. It is important to notice that there are actually three steps to make the start command to occur. First the user sends the start message using the software interface, asking for the start of emulation. The other two steps correspond to the start signal that the TG 12 receives at its interface and the one the TG 12 actually works with, after the signal passes through the bi-synchronous FIFO and synchronizes with the receiving TG clock.
2. The circle marked with number 2 shows the complete transmission of the first packet of the traffic. Here, it is possible to see the emulation startup and the first packet being injected in the first clock cycle, demonstrating the correct operation of pre reading. In addition, all the packet transmission states occur, including states S17 to S29. The actual data are transmitted from TG 12 to TG 02. It is important to notice that the payload value is the same as that from the file generated by ATLAS plus four flits that corresponds to the actual insertion timestamp. The

four-flit timestamp is the time instante where the packet actually entered the NoC. The used traffic scenario has packet size of 16 flits. As the basic information in a packet occupies 13 flits, only three additional data flits were sent during emulation. These three data flits were generated at runtime and they correspond to a simple flit counter. If the scenario would require real traffic, data would be read directly from the TG's memory and injected into the NoC.

3. The circle marked with number 3 shows the parallel reading. The parallel reading enables the continuous transmission of packets. The pre reading scheme cannot be used during emulation time because the time between two packets transmission is not enough to load the data as it is possible when transmission occurs from the Serial IP. Therefore, a parallel reading procedure is necessary. At the same time state S17 starts the transmission of the packet, it also access the memory address that corresponds to the address of the beginning of the next packet. This address is 0x0009 in the example of Figure 5.3, because the scenario is set with packets of the minimum size of eight words (plus an additional reserved 0x0000 address, as can be seen in Figure 4.4 and Figure 4.5). In case the scenario works with real traffic, the address accessed would be the payload value plus the six flits that are discounted by ATLAS. The data in address 0x0009 is read in the next clock cycle and the parallel reading should end before the previous packet is transmitted. This enables the immediate transmission of the next packet whose data is stored in internal registers if the time between both transmissions timestamps is smaller than the minimum of 13 clock cycles. In case this constraint is not respected, the packet latency can increase, once the insertion would have to wait for the parallel reading end.

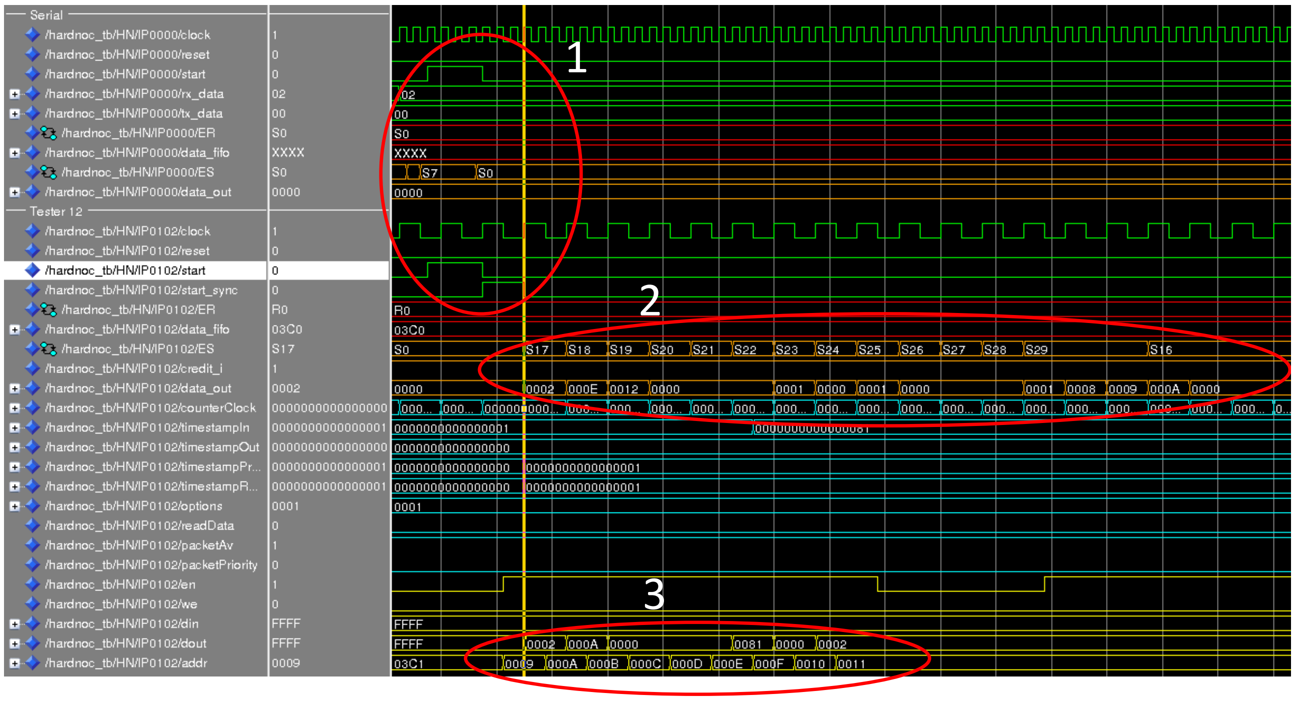
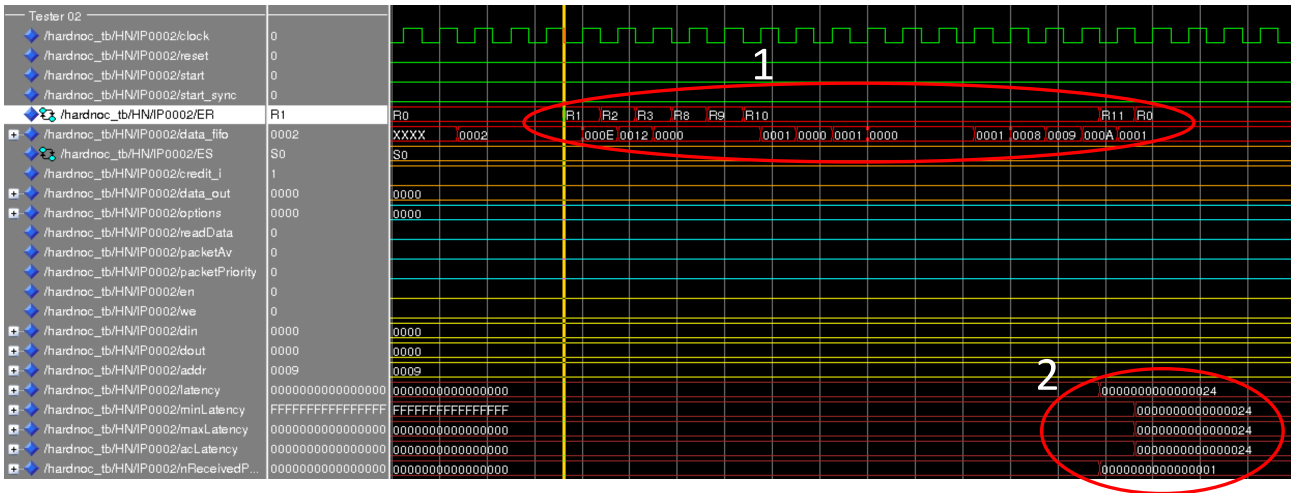

```
Serial
/hardnoc_tb/HN/IP0000/clock            1
/hardnoc_tb/HN/IP0000/reset            0
/hardnoc_tb/HN/IP0000/start            0
/hardnoc_tb/HN/IP0000/rx_data          02
/hardnoc_tb/HN/IP0000/tx_data          00
/hardnoc_tb/HN/IP0000/ER               S0
/hardnoc_tb/HN/IP0000/data_fifo        XXXX
/hardnoc_tb/HN/IP0000/ES               S0
/hardnoc_tb/HN/IP0000/data_out         0000
Tester 12
/hardnoc_tb/HN/IP0102/clock            1
/hardnoc_tb/HN/IP0102/reset            0
/hardnoc_tb/HN/IP0102/start            0
/hardnoc_tb/HN/IP0102/start_sync       0
/hardnoc_tb/HN/IP0102/ER               R0
/hardnoc_tb/HN/IP0102/data_fifo        03C0
/hardnoc_tb/HN/IP0102/ES               S17
/hardnoc_tb/HN/IP0102/credit_i         1
/hardnoc_tb/HN/IP0102/data_out         0002
/hardnoc_tb/HN/IP0102/counterClock     0000000000000000
/hardnoc_tb/HN/IP0102/timestampIn      0000000000000001
/hardnoc_tb/HN/IP0102/timestampOut     0000000000000000
/hardnoc_tb/HN/IP0102/timestampPr...   0000000000000001
/hardnoc_tb/HN/IP0102/timestampR...    0000000000000001
/hardnoc_tb/HN/IP0102/options          0001
/hardnoc_tb/HN/IP0102/readData         0
/hardnoc_tb/HN/IP0102/packetAv         1
/hardnoc_tb/HN/IP0102/packetPriority   0
/hardnoc_tb/HN/IP0102/en               1
/hardnoc_tb/HN/IP0102/we               0
/hardnoc_tb/HN/IP0102/din              FFFF
/hardnoc_tb/HN/IP0102/dout             FFFF
/hardnoc_tb/HN/IP0102/addr             0009
```
1
2
3
S7
S17 S18 S19 S20 S21 S22 S23 S24 S25 S26 S27 S28 S29 S16
0002 000E 0012 0000 0001 0000 0001 0000 0001 0008 0009 000A 0000
0002 000A 0000 0081 0000 0002
03C1 0009 000A 000B 000C 000D 000E 000F 0010 0011

Figure 5.3 – Start signal reception, injection of the first packet in the first clock cycle and parallel reading of the second packet.

5.2.3 Emulation Packet Reception

Figure 5.4 shows TG 02 receiving the first packet injected by TG 12 and the latency values computed at the end of the transmission. Both details are numbered and commented in the items below:

1. Number 1 shows the complete reception of the first packet by TG 02. The TG stays in the idle state R0 until a flit arrives and starts the receiving process before it returns to state R0. It is possible to notice that 16 flits arrived, although the payload transmitted is 0x000E (the defined 16 flits, minus the 6 ATLAS discounts, plus the 4 the TG insert). In addition, it is possible to see the two timestamps. Both are equal, because the packet was injected without contention.
2. Number 2 shows the latency values calculated by TG 02 and stored in memory. The first packet latency also appears as minimum, maximum and accumulated latency, because only one packet was transmitted to TG 02 so far.



Tester 02	
/hardnoc_tb/HN/IP0002/clock	0
/hardnoc_tb/HN/IP0002/reset	0
/hardnoc_tb/HN/IP0002/start	0
/hardnoc_tb/HN/IP0002/start_sync	0
/hardnoc_tb/HN/IP0002/ER	R1
/hardnoc_tb/HN/IP0002/data_fifo	0002
/hardnoc_tb/HN/IP0002/ES	S0
/hardnoc_tb/HN/IP0002/credit_i	1
/hardnoc_tb/HN/IP0002/data_out	0000
/hardnoc_tb/HN/IP0002/options	0000
/hardnoc_tb/HN/IP0002/readData	0
/hardnoc_tb/HN/IP0002/packetAv	0
/hardnoc_tb/HN/IP0002/packetPriority	0
/hardnoc_tb/HN/IP0002/en	0
/hardnoc_tb/HN/IP0002/we	0
/hardnoc_tb/HN/IP0002/din	0000
/hardnoc_tb/HN/IP0002/dout	0000
/hardnoc_tb/HN/IP0002/addr	0009
/hardnoc_tb/HN/IP0002/latency	0000000000000000
/hardnoc_tb/HN/IP0002/minLatency	FFFFFFFFFFFFFFFF
/hardnoc_tb/HN/IP0002/maxLatency	0000000000000000
/hardnoc_tb/HN/IP0002/acLatency	0000000000000000
/hardnoc_tb/HN/IP0002/nReceivedP...	0000000000000000

Figure 5.4 – TG 02 receiving the first packet from TG 12.

5.2.4 Steady State Emulation Operation

Figure 5.5 shows an overview of the emulation start. The details are numbered and commented in the items below:

1. The circle marked with number 1 shows the moment of the emulation start. The start signal sent by the user arrives at the Serial IP, which sends it to all TGs. The TG 12 receives the start signal from the bi-synchronous FIFO and starts its internal counter.
2. Here occurs the injection of the first packet into the NoC. In the scenario used (the same described in Figure 4.14), the first packet is injected in timestamp zero. This injection would not be possible if the first packet would not be pre read during the TG 12 programming period.
3. Number 3 shows the injection of the second packet into the NoC. Parallel reading during the first packet injection stored this information in internal registers before injection.
4. Number 4, just as number 3, shows a complete packet transmission and emphasizes the parallel reading of the packet executed during the previous injection process.

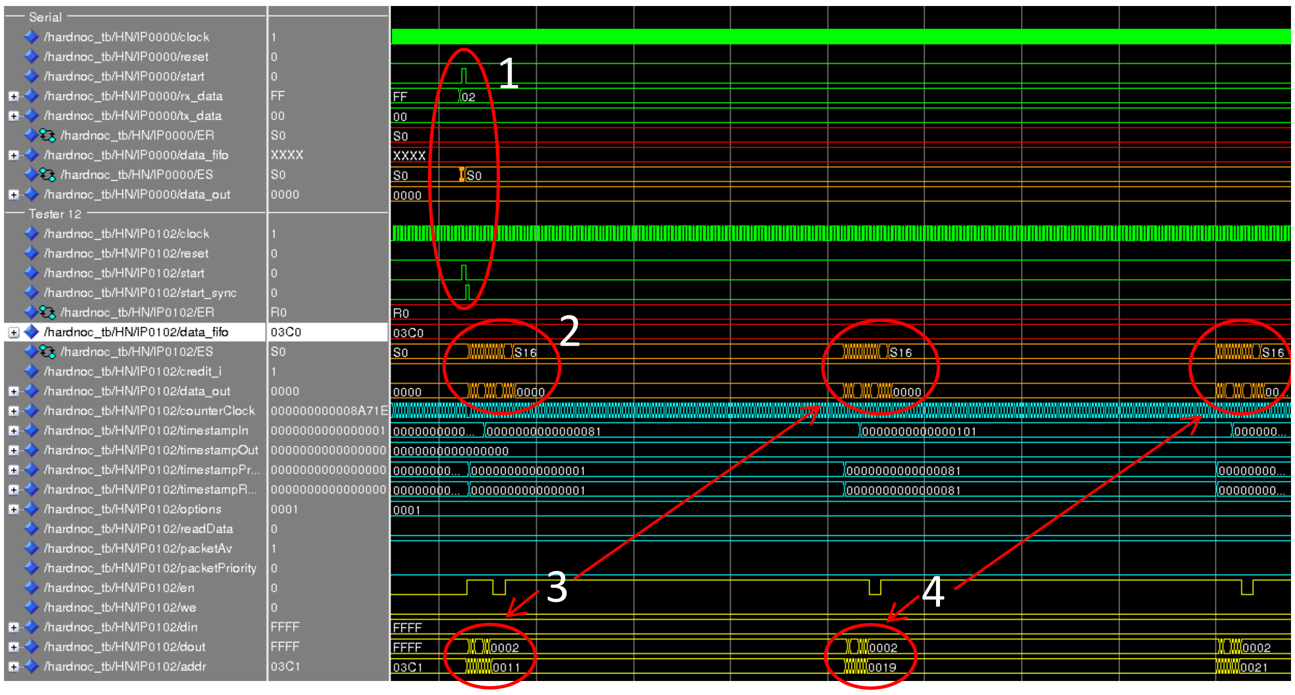
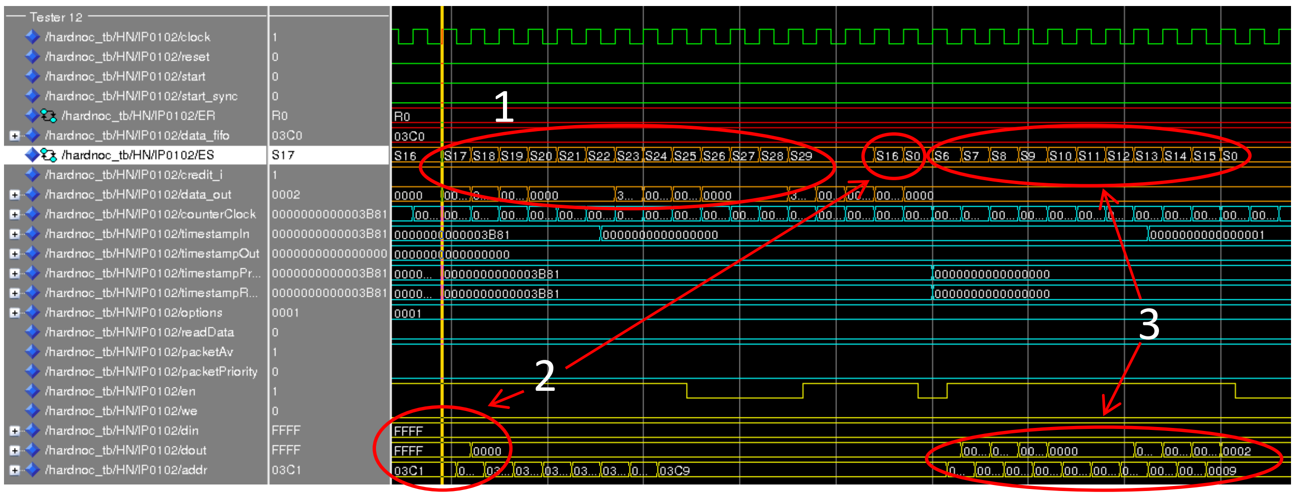

Serial
/hardnoc_tb/HN/IP0000/clock
/hardnoc_tb/HN/IP0000/reset
/hardnoc_tb/HN/IP0000/start
/hardnoc_tb/HN/IP0000/rx_data
/hardnoc_tb/HN/IP0000/tx_data
/hardnoc_tb/HN/IP0000/ER
/hardnoc_tb/HN/IP0000/data_fifo
/hardnoc_tb/HN/IP0000/ES
/hardnoc_tb/HN/IP0000/data_out
Tester 12
/hardnoc_tb/HN/IP0102/clock
/hardnoc_tb/HN/IP0102/reset
/hardnoc_tb/HN/IP0102/start
/hardnoc_tb/HN/IP0102/start_sync
/hardnoc_tb/HN/IP0102/ER
/hardnoc_tb/HN/IP0102/data_fifo
/hardnoc_tb/HN/IP0102/ES
/hardnoc_tb/HN/IP0102/credit_i
/hardnoc_tb/HN/IP0102/data_out
/hardnoc_tb/HN/IP0102/counterClock
/hardnoc_tb/HN/IP0102/timestampIn
/hardnoc_tb/HN/IP0102/timestampOut
/hardnoc_tb/HN/IP0102/timestampPr...
/hardnoc_tb/HN/IP0102/timestampR...
/hardnoc_tb/HN/IP0102/options
/hardnoc_tb/HN/IP0102/readData
/hardnoc_tb/HN/IP0102/packetAv
/hardnoc_tb/HN/IP0102/packetPriority
/hardnoc_tb/HN/IP0102/en
/hardnoc_tb/HN/IP0102/we
/hardnoc_tb/HN/IP0102/din
/hardnoc_tb/HN/IP0102/dout
/hardnoc_tb/HN/IP0102/addr
1
2
3
4

Figure 5.5 – Overview of the simulation start, showing start signal processing, the first three packets being sent and the pre readings.

5.2.5 End of Packet Sending during Emulation

Figure 5.6 shows the sending of the last packet of the traffic and the end of the transmission. The details of how the platform identifies the last packet and the actions it takes after this are numbered and commented in the items below:

1. The circle marked with number 1 shows the complete sending of the last packet from TG 12 to TG 02. The machine stays at state R29 until all data flits are sent.
2. Number 2 marks two circles. The lower circle emphasizes the parallel reading in the beginning of the last packet transmission. This step reads the next target as 0xFFFF, meaning the end of the traffic. The upper circle shows that the TG returns to state S16 to wait for further packets from the NoC, since it identifies that the next target address is invalid, which indicates the transmission is over. This leads to state S0, the initial, idle state.
3. Number 3 also marks two circles. The lower circle shows what happens when a transmission is over. Once the state machine returns to the state S0, an immediate pre reading of the first packet is triggered. This enables a new emulation with the same traffic without the need for reprogram the TG. It would not be possible to start a new emulation with the same traffic because the injection on timestamp zero would not give enough time to the TG load the first packet header information. The upper circle shows the pre reading states sequence.



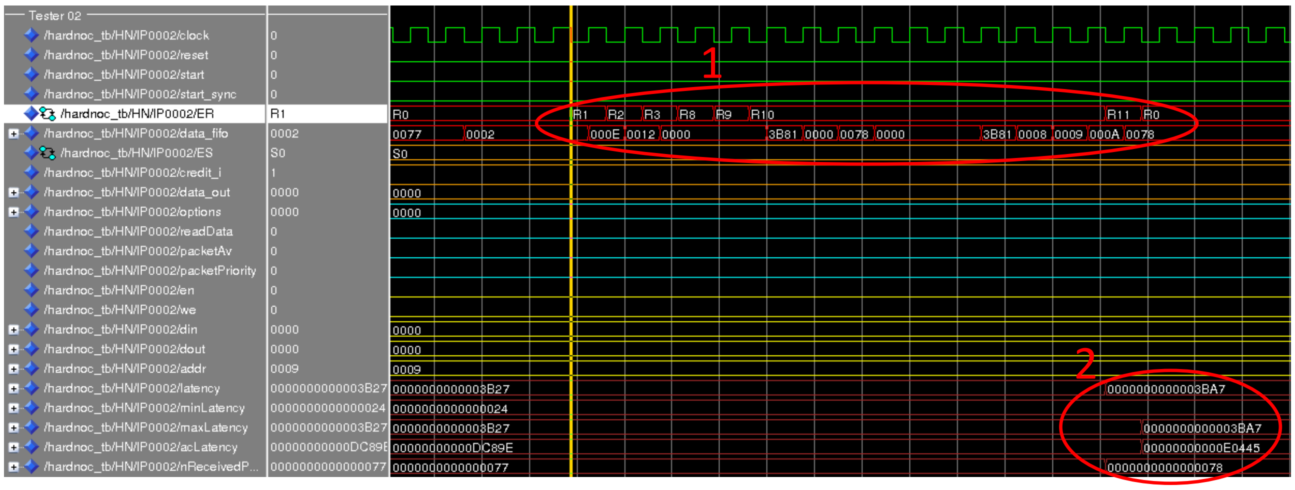
```
Tester 12
/hardnoc_tb/HN/IP0102/clock            1
/hardnoc_tb/HN/IP0102/reset            0
/hardnoc_tb/HN/IP0102/start            0
/hardnoc_tb/HN/IP0102/start_sync       0
/hardnoc_tb/HN/IP0102/ER               R0
/hardnoc_tb/HN/IP0102/data_fifo        03C0
/hardnoc_tb/HN/IP0102/ES               S17
/hardnoc_tb/HN/IP0102/credit_i         1
/hardnoc_tb/HN/IP0102/data_out         0002
/hardnoc_tb/HN/IP0102/counterClock     0000000000003B81
/hardnoc_tb/HN/IP0102/timestampIn      0000000000003B81
/hardnoc_tb/HN/IP0102/timestampOut     0000000000000000
/hardnoc_tb/HN/IP0102/timestampPr...   0000000000003B81
/hardnoc_tb/HN/IP0102/timestampR...    0000000000003B81
/hardnoc_tb/HN/IP0102/options          0001
/hardnoc_tb/HN/IP0102/readData         0
/hardnoc_tb/HN/IP0102/packetAv         1
/hardnoc_tb/HN/IP0102/packetPriority   0
/hardnoc_tb/HN/IP0102/en               1
/hardnoc_tb/HN/IP0102/we               0
/hardnoc_tb/HN/IP0102/din              FFFF
/hardnoc_tb/HN/IP0102/dout             FFFF
/hardnoc_tb/HN/IP0102/addr             03C1
```

Figure 5.6 – Sending the last packet of traffic, the identification by parallel reading of the traffic end and the pre reading activation to prepare for a possible repetition of emulation.

5.2.6 End of Packet Reception during Emulation

Figure 5.7 shows TG 02 receiving the last packet injected by TG 12 and the latency values computed at the end of the transmission. Both details are numbered and commented in the items below:

1. The circle marked with number 1 shows the reception of the last packet by a receiving TG (TG 12). It can be seen that the receiver state machine operation remains the same for every packet received, being it the last or not.
2. Number 2 shows the computation of latency values by TG 02 and its storage in memory. These results contain minimum and maximum latency of a packet transmitted between TG 12 and TG 02. The accumulated latency of all 120 packets (0x78 in hexadecimal notation) is also computed.



```
Tester 02
/hardnoc_tb/HN/IP0002/clock            0
/hardnoc_tb/HN/IP0002/reset            0
/hardnoc_tb/HN/IP0002/start            0
/hardnoc_tb/HN/IP0002/start_sync       0
/hardnoc_tb/HN/IP0002/ER               R1
/hardnoc_tb/HN/IP0002/data_fifo        0002
/hardnoc_tb/HN/IP0002/ES               S0
/hardnoc_tb/HN/IP0002/credit_i         1
/hardnoc_tb/HN/IP0002/data_out         0000
/hardnoc_tb/HN/IP0002/options          0000
/hardnoc_tb/HN/IP0002/readData         0
/hardnoc_tb/HN/IP0002/packetAv         0
/hardnoc_tb/HN/IP0002/packetPriority   0
/hardnoc_tb/HN/IP0002/en               0
/hardnoc_tb/HN/IP0002/we               0
/hardnoc_tb/HN/IP0002/din              0000
/hardnoc_tb/HN/IP0002/dout             0000
/hardnoc_tb/HN/IP0002/addr             0009
/hardnoc_tb/HN/IP0002/latency          0000000000003B27
/hardnoc_tb/HN/IP0002/minLatency       0000000000000024
/hardnoc_tb/HN/IP0002/maxLatency       0000000000003B27
/hardnoc_tb/HN/IP0002/acLatency        00000000000DC89E
/hardnoc_tb/HN/IP0002/nReceivedP...    0000000000000077
```

Figure 5.7 – TG 02 receiving the last packet of a traffic.

5.3 Prototyping

The HNPlus was prototyped in the Xilinx ML505 Evaluation Platform, which contains a Virtex 5 XC5VLX50T-FF1136 FPGA. This FPGA contains 120 18Kbits Block RAMs, or equivalently, 60 36Kbits Block RAMs. Figure 5.8 shows the floorplan used to optimize the physical synthesis of HardNoC-GLP and HNPlus, each block contains one IP and its respective router. The PlanAhead was utilized to make the florrplan and the implementation. The physical synthesis reports of the HardNoC and HNPlus appear in Figure 5.9 and in Figure 5.10 respectively. This gives an idea of the hardware occupation by both platforms. The HardNoC-GLP and HNPlus were synthesized with the same 3x3 Hermes-GLP NoC and the only differences between them are the TGs structure and the Serial IP structure. It is possible to see that the HNPlus platform uses simpler TG modules, reducing the logic complexity, mostly because HNPlus state machines were simplified compared to the HardNoC-GLP. Even though, the new platform is more flexible.

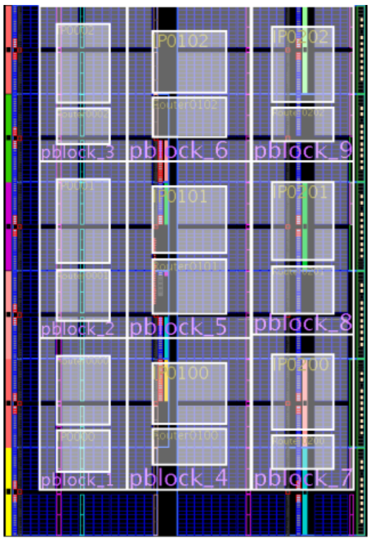
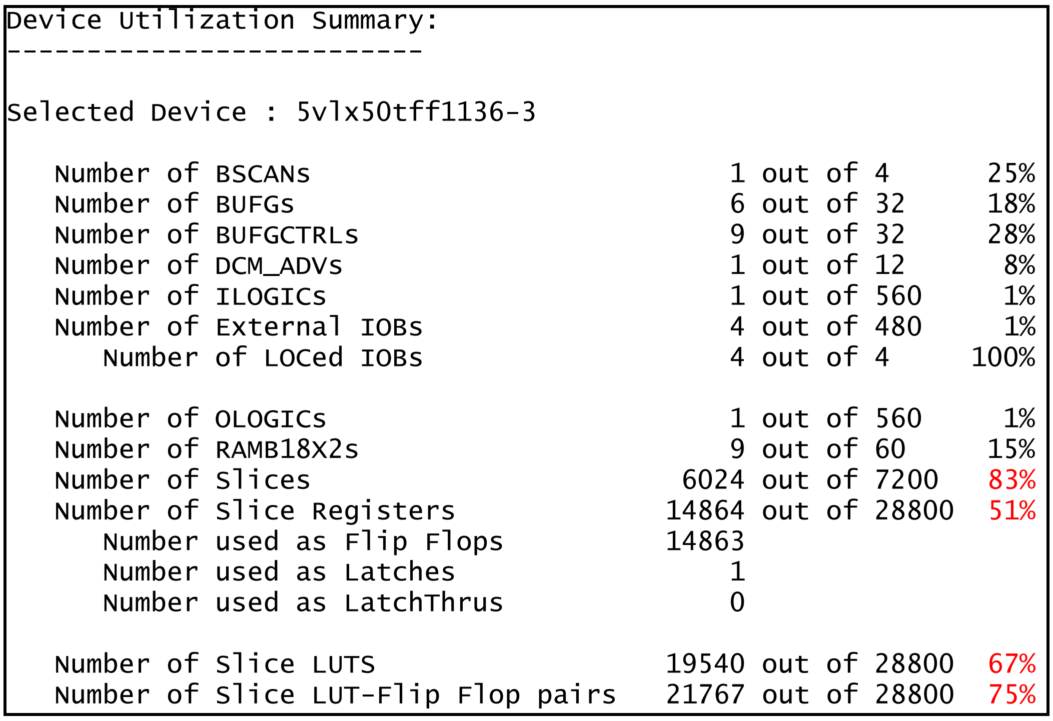


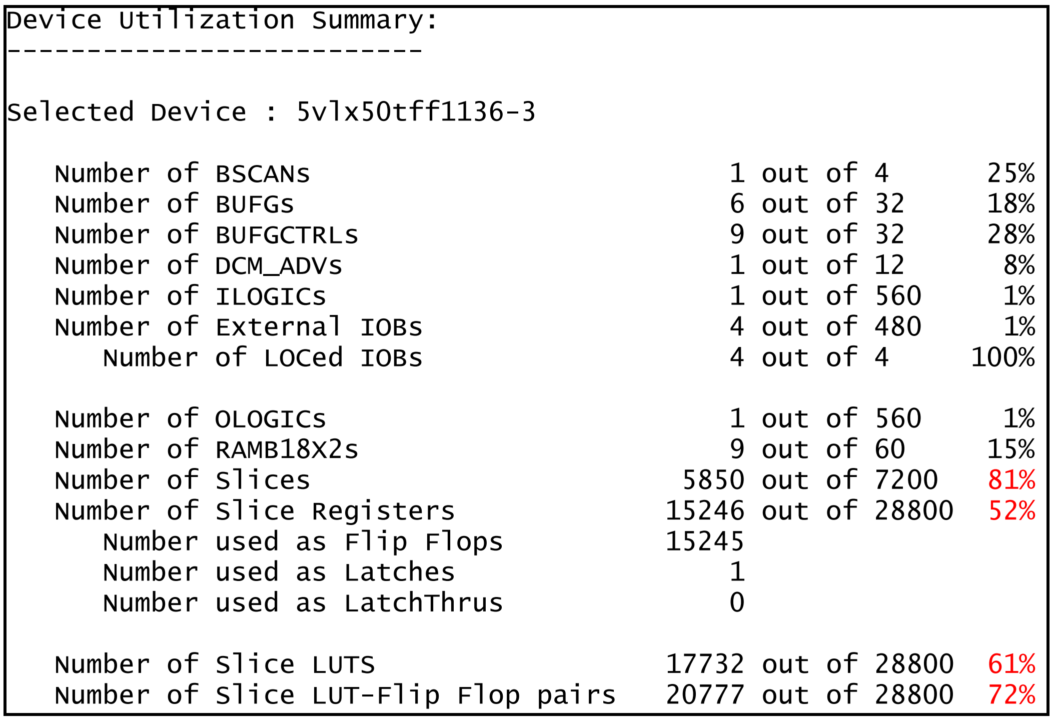
Figure 5.8 – Floorplanning used to optimize the physical synthesis of HardNoC-GLP and HNPlus.

Note that there is a decrease on the number of used slices from HardNoC-GLP to HNPlus (6024 to 5850), but a small increase on the number of slice registers (from 14864 to 15246). The reduction occurs because the logic of the TGs state machines became simpler. What causes the increase is the fact that the HNPlus platform contains extra registers to keep pre reading and parallel reading information between packet transmissions. The overall number of LUTS is reduced (from 19540 to 17732) and the overall number of slice LUT flip-flop pairs also reduces (from 21767 to 20777) because of the simpler logic at the TGs.



```
Device Utilization Summary:
--------------------------

Selected Device : 5vlx50tff1136-3

   Number of BSCANs                         1 out of 4       25%
   Number of BUFGs                          6 out of 32      18%
   Number of BUFGCTRLs                      9 out of 32      28%
   Number of DCM_ADVs                       1 out of 12       8%
   Number of ILOGICs                        1 out of 560      1%
   Number of External IOBs                  4 out of 480      1%
      Number of LOCed IOBs                  4 out of 4      100%

   Number of OLOGICs                        1 out of 560      1%
   Number of RAMB18X2s                      9 out of 60      15%
   Number of Slices                      6024 out of 7200    83%
   Number of Slice Registers            14864 out of 28800   51%
      Number used as Flip Flops         14863
      Number used as Latches                1
      Number used as LatchThrus             0

   Number of Slice LUTS                 19540 out of 28800   67%
   Number of Slice LUT-Flip Flop pairs  21767 out of 28800   75%
```

Figure 5.9 – HardNoC-GLP physical synthesis report using a 3x3 Hermes-GLP.



```
Device Utilization Summary:
--------------------------

Selected Device : 5vlx50tff1136-3

   Number of BSCANs                         1 out of 4       25%
   Number of BUFGs                          6 out of 32      18%
   Number of BUFGCTRLs                      9 out of 32      28%
   Number of DCM_ADVs                       1 out of 12       8%
   Number of ILOGICs                        1 out of 560      1%
   Number of External IOBs                  4 out of 480      1%
      Number of LOCed IOBs                  4 out of 4      100%

   Number of OLOGICs                        1 out of 560      1%
   Number of RAMB18X2s                      9 out of 60      15%
   Number of Slices                      5850 out of 7200    81%
   Number of Slice Registers            15246 out of 28800   52%
      Number used as Flip Flops         15245
      Number used as Latches                1
      Number used as LatchThrus             0

   Number of Slice LUTS                 17732 out of 28800   61%
   Number of Slice LUT-Flip Flop pairs  20777 out of 28800   72%
```

Figure 5.10 - HNPlus physical synthesis report using a 3x3 Hermes-GLP.

5.4 Experimentation

This Section describes two experiments. The first emulates two simple traffics and verifies the latency values at each target TG. The second tries to cause a congestion, where two TGs send packets through a common path to the same target TG. The latency values are calculated by the number of clock cycles that each packet needed to be transmitted, but in GALS systems this could be not valid if the IPs were using different clocks. So because this limitation it is necessary to pay attention in the anlysis of the results if all IPs involved were using the same clock signal. In these experiments, all IPs were configured to use the same clock.

5.4.1 One Source TG Sending Packets to Two Different Targets

This experiment verifies if statistical results are coherent. Figure 5.11 shows two different traffic scenarios generated with ATLAS. Both have the same distribution type and characteristics, but with different TG targets. First, TG 02 sends 100 packets to TG 20 (blue path). Afterwards, it sends 100 packets to TG 01 (orange path). Latency results for TG 20 should be bigger than the ones for TG 01.

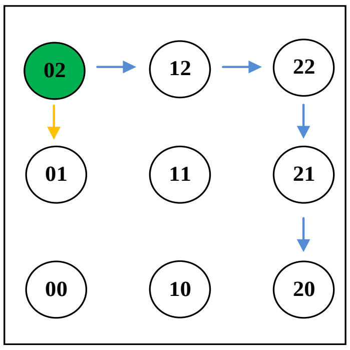
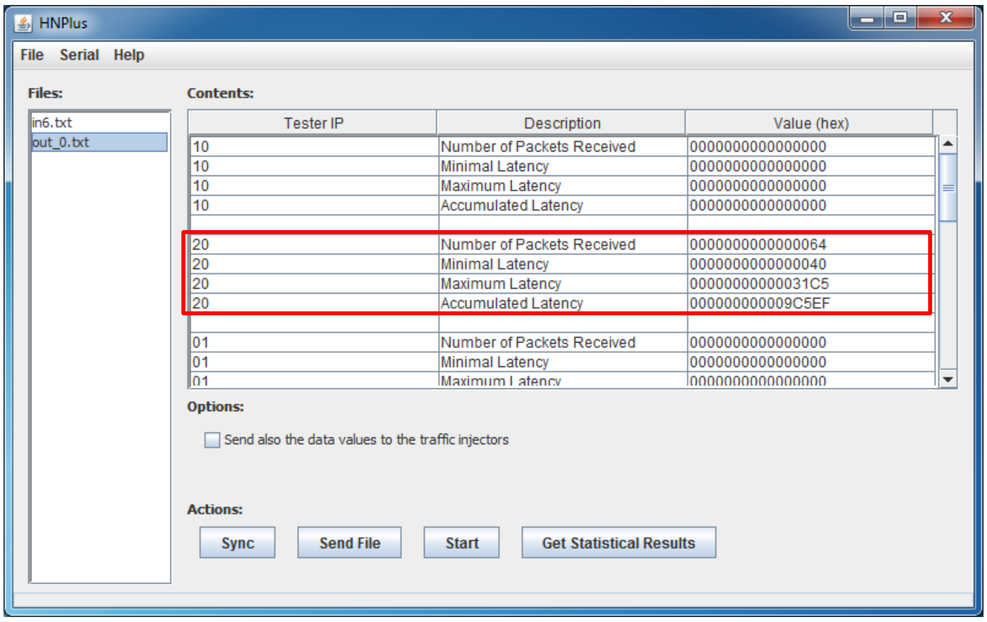


Figure 5.11 – First traffic is the blue path and second traffic is the orange path.

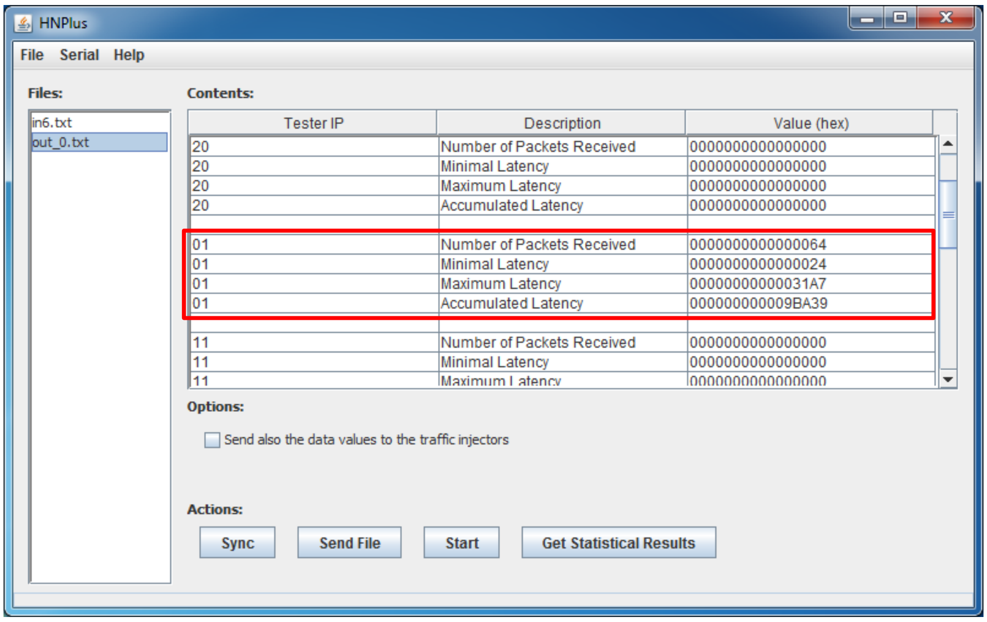
Figure 5.12 and Figure 5.13 show the statistical results obtained with the traffic emulations scenarios. It is noticeable that latency values in the first are higher than the second scenario. As the path for packets to go from TG 02 to TG 20 is longer than the one to go from TG 02 to TG 01, these results were the expected ones. It is also possible to verify that all the 100 packets transmitted to both of the target TGs arrived correctly. Note that the wormhole nature of traffic creates a pipeline, and that latency increase is small because of the pipeline operation.



Tester IP	Description	Value (hex)
10	Number of Packets Received	0000000000000000
10	Minimal Latency	0000000000000000
10	Maximum Latency	0000000000000000
10	Accumulated Latency	0000000000000000
20	Number of Packets Received	0000000000000064
20	Minimal Latency	0000000000000040
20	Maximum Latency	00000000000031C5
20	Accumulated Latency	000000000009C5EF
01	Number of Packets Received	0000000000000000
01	Minimal Latency	0000000000000000
01	Maximum Latency	0000000000000000

☐ Send also the data values to the traffic injectors

Figure 5.12 – Statistical results, after first traffic emulation.



Tester IP	Description	Value (hex)
20	Number of Packets Received	0000000000000000
20	Minimal Latency	0000000000000000
20	Maximum Latency	0000000000000000
20	Accumulated Latency	0000000000000000
01	Number of Packets Received	0000000000000064
01	Minimal Latency	0000000000000024
01	Maximum Latency	00000000000031A7
01	Accumulated Latency	000000000009BA39
11	Number of Packets Received	0000000000000000
11	Minimal Latency	0000000000000000
11	Maximum Latency	0000000000000000

☐ Send also the data values to the traffic injectors

Figure 5.13 – Statistical results, after second traffic emulation.

5.4.2 Two TGs Sending Packets to the Same Target

This experiment causes a hotspot on node 11. To do achieve this, the used scenario contains two TGs (01 and 21) sending packets to the same target TG (12). According the algorithm XY used in Hermes-GLP, both paths should cross node 11. The priorities and packet timestamps of both traffics are identical, causing concurrency in this node. Thus, each time a traffic flow finds the path from node 11 occupied (either coming from 01 or from 21) it must wait until the flow ends and then take this path. This scenario would be more interesting if it were possible to know the individual latencies of the packets. This is discussed a little more in Section 6.1.

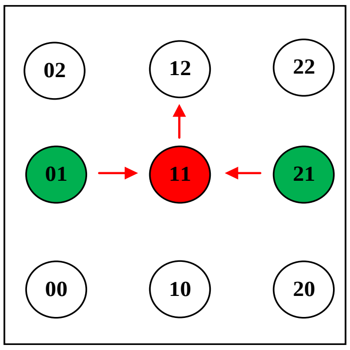
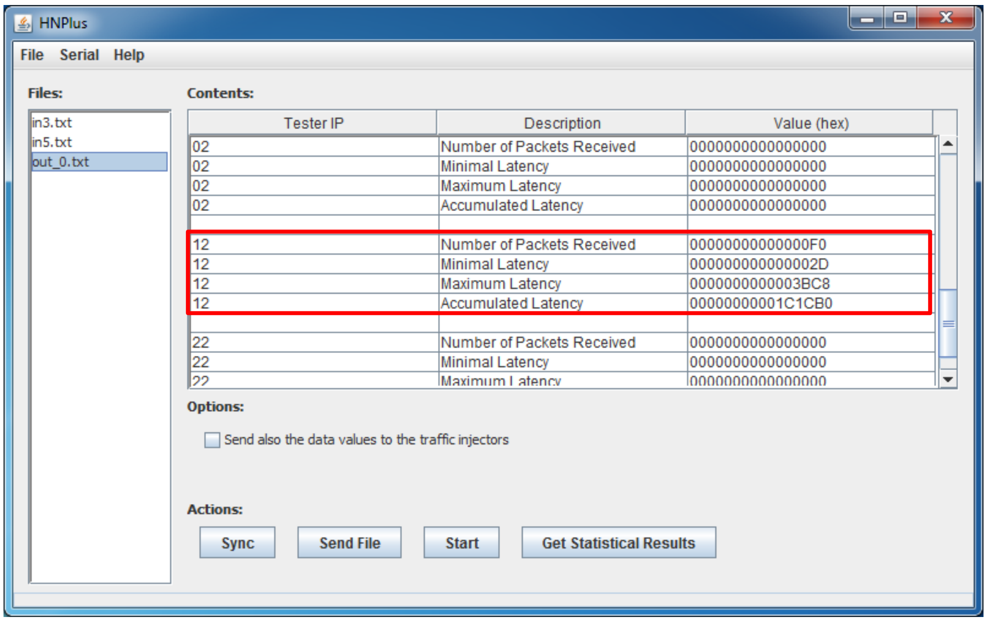


Figure 5.14 – Traffic scenario used to cause a traffic hotspot on node 11.

Analyzing the results, in Figure 5.15, is possible to verify that all packets from the two TGs were sent, because each one sent 120 packets and 240 packets arrived in TG 12. Comparing the maximum latency and the accumulated latency with previous experiments, this scenario presents considerably bigger latency values, demonstrating the occurrence of contention in the traffic hostspot.



Tester IP	Description	Value (hex)
02	Number of Packets Received	0000000000000000
02	Minimal Latency	0000000000000000
02	Maximum Latency	0000000000000000
02	Accumulated Latency	0000000000000000
12	Number of Packets Received	00000000000000F0
12	Minimal Latency	000000000000002D
12	Maximum Latency	0000000000003BC8
12	Accumulated Latency	00000000001C1CB0
22	Number of Packets Received	0000000000000000
22	Minimal Latency	0000000000000000
22	Maximum Latency	0000000000000000

☐ Send also the data values to the traffic injectors

Figure 5.15 - Statistical results obtained after hotspot emulation.

6. CONCLUSION AND FUTURE WORK

This work produced a new emulation platform for NoCs. This platform is called HNPlus and its main contribution is a new traffic injection scheme and the integration with the traffic files generated automatically by the ATLAS tool. The new traffic injection scheme enables testing the NoC within the platform with both real traffic or synthetic data. The fact that the new TG just injects traffic according the information stored in the TG memories makes it more flexible about the kind of traffic to use, when compared with the HardNoC-GLP version. The integration with the ATLAS tool, in addition, enables the automatic generation of the traffic files for the TGs.

A new software interface makes the use and the analysis of results simpler to the user. The software interface also automatically modifies the files generated by ATLAS, making it compatible with the HNPlus platform input traffic files used to program the TGs.

The HNPlus platform was validated by simulation and had most of its main features verified and tested. After that, the prototyping of the platform was conducted using a Xilinx Evaluation Platform and the experiments showed its correct operation.

The objectives of the work were thoroughly achieved. The proposed flexibility of traffic emulation was tested and comproved along this document.

6.1 Future Work

The work described in this document has several topics for further development and research. The HNPlus platform new scheme for generation and injection of generic traffic in NoCs allows the emulation and analysis of virtually any kind of traffic. The stored results in each TG at the end of the emulation process, however, are quite limited. The TG could provide more statistical information, not just latencies. It could verify if the received packets are correct, sending always the same data and checking if it is correct. It would also be possible to compute latencies according to the packet size, so that in just one emulation it would be possible to analyze latency differences of packets with different sizes. In addition, it could be possible to calculate the minimum and maximum waiting times, that is the time difference between the time that the packet should be injected and the real time that the packet was injected into the NoC, which may occur if some path of the NoC is congested. When the current TGs work with synthetic traffic data, it is always insert the same data, so the TGs could use LFSRs to inject random data.

The TG local memory size is another important topic. Currently, each TG stores only 127 packets in its local memory, if the traffic is configured to work with synthetic traffic data. Although even for the small FPGA used here each local memory could be made to contain as much as six times more packets without exceeding the FPGA capacity. Even if this would still not be enough for emulations with large real traffic data sets or longer emulations. In addition, the HNPlus software could veify if the traffic file selected will fit into the memory space.

REFERENCES

[CAL98] Calazans, N. L. V. “Projeto Lógico Automatizado de Sistemas Digitais Sequenciais”. Rio de Janeiro: Imprinta Gráfica e Editora Ltda, 1998, 318p.

[CHA84] Chapiro, D. M. “Globally-Asynchronous Locally-Synchronous Systems”. PhD Thesis, CS Dept, Stanford University, Oct. 1984, 140 p.

[GEN05] Genko, N.; Atienza, D.; De Micheli, G.; Mendias, J. M.; Hermida, R.; Catthoor, F. “A Complete Network-On-Chip Emulation Framework”. In: Design, Automation and Test in Europe Conference and Exhibition (DATE’05), Munich, Mar. 2005, pp. 246-251.

[KRA08] Krasteva, Y. E.; Criado, F.; de la Torre, E.; Riesgo, T. “A Fast Emulation-based NoC Prototyping Framework”. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig'08), Cancun, Dec. 2008, pp. 211-216.

[LOT11] Lotlikar, S.; Pai, V.; Gratz, P. “AcENoCs: A configurable HW/SW Platform for FPGA Accelerated NoC Emulation”. In: 24th International Conference on VLSI Design (VLSID’11), Chennai, Jan. 2011, pp. 147-152.

[MAH05] Mahadevan, S.; Angiolini, F.; Storgaard, M.; Olsen, R. G. “A Network Generator Model for Fast Network-on-Chip Simulation”. In: Design, Automation and Test in Europe Conference and Exhibition, (DATE’05), Munich, Mar. 2005, pp. 780-785.

[MEL05] Mello, A. V. de; Tedesco, L. P.; Calazans, N. L. V.; Moraes, F. G. "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC". In: 18th Symposium on Integrated Circuits and Systems Design (SBCCI'05), Florianópolis, Sep. 2005, pp. 178-183.

[MOR04] Moraes, F. G.; Calazans, N. L. V; Mello A. V.; Möller L.; Ost L. “Hermes: an infrastructure for low area overhead packet-switching networks on chip”. Integration the VLSI journal, 38(1), Oct. 2004, pp. 69-93.

[MOR10] Moreira, M. T. “Design and Implementation of a Standard Cell Library for Building Asynchronous ASICs”. Revista da Graduação, 4(1), Pontifícia Universidade Católica do Rio Grande do Sul, 2011, 125 p. Disponível em http://revistaseletronicas.pucrs.br/ojs/index.php/graduacao/article/viewFile/8657/6118.

[PON08] Pontes, J. J. H.; Moreira, M. T.; Soares, R. I.; Calazans, N. L. V. “Hermes-GLP: A GALS Network on Chip Router with Power Control Techniques”. In: IEEE Computer Society Annual Symposium on VLSI Design (ISVLSI’08), Montpellier, Apr. 2008, pp. 347-352.

[TAN11] Tan, J.; Fresse, V.; Rousseau, F. “Generation of emulation platforms for NoC exploration on FPGA”. In: IEEE International Symposium on Rapid System Prototyping (RSP’11), Karlsruhe, May, 2011, pp. 186-192.

[VAN08] Vangal, S. R.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Singh, A.; Jacob, T.; Jain, S.; Erraguntla, V.; Roberts, C.; Hoskote, Y.; Borkar, N.; Borkar, S. “An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS”. IEEE Journal of Solid-State Circuits, 43(1), Jan. 2008, pp. 29-41.

[WAN10] Wang, D.; Jerger, N. E.; Steffan, J. G. “DART: Fast and Flexible NoC Simulation using FPGAs”. In: 5th ACM/IEEE International Symposium on Networks-on-Chip (NOCS’11), Pittsburgh,May 2011, pp. 145-152.

[WEN09] Wen, H.; Du, C.; Zhang, D.; Geng, L.; Gao, M.; Chen, Y.; Verhoeff, T. “Design of An On-Line

Configurable Traffic Generator for NoC". In: 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication (ASID'09), Hong Kong, Aug. 2009, pp. 556-559.