The implementation of XHiNoC based MPSoC

Study: Electronic engineer
Author: Jordi Mir Boada
Advisor: Haoyuan Ying
Any: 2011-2012
Acknowledgements

I would like to dedicate a few lines to everyone who has made possible this thesis, to everyone who has supported and help me during my stay in TU Darmstadt. First of all, I would like to express my deep gratitude to Professor K. Hofmann for giving me the chance of working with his team. Without him, this thesis would not have been possible.

Secondly, I would like to sincerely thank to my advisor Haoyuan Ying for his very valuable and always helpful advices as well as for his feedback and support.

In these lines, I cannot forget my “friends abroad”, you have made the best of my stay, we have laughed, shared and been happy altogether. I would have never expected to find anyone like you.

Last but not least, I would like to dedicate some words to my family and in special for Anna. I know that it hasn’t been easy for both but I am sure it has made us stronger. Thanks you, family, for the infinite support you have given to me and all the love I have gotten from you.
Abstract

Nowadays, the electronic is advancing an extremely high rhythm. The new applications demand great computational load and high speeds. Therefore the electronic future is linked with multi-tasking concept and the multi-processor assembled on a single chip. It is known as MPSoC (Multi Processors System-on-chip).

One of the main factors that affects directly to the chip performance is the interconnect system among the different cores. So far, there have developed several interconnect alternatives for systems that have less than tenth cores. However the problem gets out of control when the number of processing elements increases more than 10. Then start to appear problems like bottleneck performance, the effect of the high electromagnetism interference can disturb the interconnect functionality, the wiring complexity problems, in general no one is flexible.

Root of all this, one solution that has won more strongly is known as NoC (Network on chip). The NoC is a communication infrastructure that provides better scalability and flexibility as well as a great bandwidth. In addition it has a large level of reconfiguration.

This thesis is based on, firstly, a documentation chapter where examined two networks-on-chip completely different, HeMPS NoC and XHiNoC. Then there is a design and implementation chapter that involves the adaptation and commissioning of XHiNoC within design platform known as HeMPS. This platform allows us evaluate the NoC performance. The third point we find a measures and comparatives between the two NoC and finally, at the end, there are some conclusion drawn during the project.
Resum del projecte

L’electrònica d’avui en dia està avançant a un ritme extremadament alt. Les noves aplicacions exigeixen unes carregues computacionals i unes velocitats elevadíssimes. És per això que el futur de l’electrònica està lligat en el concepte de multi-tasca i a la integració de multi-processadors en un sol xip, el que es conegut com a MPSoCs. (Multi Processors System-on-chip).

Un dels principals factors que afecten directament en el rendiment del chip és el sistema de interconnexió entre els diferents nuclis. Fins el dia d’avui s’han desenvolupat diferents sistemes per a xips que no arriben a la desena de nuclis. Ara bé, el problema es descontrola quan el número d’elements integrats supera als 10. Llavors comencen aparèixer problemes de coll de botella, nivells als de electromagnetisme i els sistemes de interconnexió és compliquen.

Arrel de tot això, una de les solucions que s’ha imposat amb més força és la coneguda com a NoC (Network-on-chip). La NoC és una infraestructura de comunicació que proveeix una millor escalabilitat i flexibilitat, a part de un gran ample de banda de comunicació. A més posseeix un gran grau de reconfiguració.

Aquesta projecte està basat en, primerament, una part documentaria on s’estudien dues networks-on-chip totalment diferents, HeMPS NoC i XHiNoC. Seguidament una part de disseny i implementació que consisteix en l’adaptació i posta en marxa de XHiNoC dins una plataforma de disseny coneguda com a HeMPS, la qual ens permetrà avaluar el seu rendiment. En tercer punt hi trobarem una part de mesures i comparatives entre les dues NoC i finalment unes conclusions extretes durant la realització del projecte.
# Table of contents

1. Introduction ....................................................................................................................... 7  
   1.1 Motivation .................................................................................................................. 7  
   1.2 Objectives ................................................................................................................... 9  
   1.3 Thesis outline .............................................................................................................. 9  
2. Analysis ............................................................................................................................ 11  
   2.1 Work tools ................................................................................................................ 11  
   2.2 Important aspects ..................................................................................................... 22  
   2.3 Alternatives ............................................................................................................... 23  
   2.3 Justification of the adopted solution ....................................................................... 28  
3. Design .............................................................................................................................. 29  
   3.1 Network Interface general diagram ......................................................................... 29  
   3.1 Network Interface Sending Unit ............................................................................... 32  
   3.2 Network Interface Receiving Unit ............................................................................ 38  
4. Measures ......................................................................................................................... 45  
   4.1 Latency comparison .................................................................................................. 45  
   4.2 Execution time comparison ...................................................................................... 51  
5. Conclusions ...................................................................................................................... 54  
6. Bibliography..................................................................................................................... 56
Chapter 1

Introduction

Contents

1.1 Motivation ................................................................. 7
1.2 Objective ................................................................. 9
1.3 Thesis outline ............................................................ 9

1.1 Motivation

Nowadays, the electronic is subjected to a strong tendency towards the System-on-Chip (SoC). SoC is an integrated circuit (IC) that integrates all components into a single chip and it is one of the potential solutions for system level design. In a near future, SoCs will integrate dozens of processing elements, achieving a great computational power in a single chip. This computational power is commonly used by complexes multimedia algorithms. Thus, systems based on a single processor are not suitable for this technological trend and the development of SoCs with more complexity will be multi processed, known as MPSoCs.

One of the main points on SoC or MPSoCs is the interconnect system, which interconnects several components such as microprocessor cores, MCU, DSP, memories and other intellectual properties components (IP).

Since interconnect technology affects more profoundly on chip performance and power consumption, improving on-chip communication technology has become increasingly important to researchers [1]. A high-throughput communication infrastructure is required to meet the bandwidth requirement of each data communication flows generated due to interacting processors in the MPSoC systems.

So far, there have been several alternatives in order to solve the interconnect system such as bus-based interconnected system, a full crossbar interconnect, point-to-point interconnect (dedicated wires), but the problems become when the number of IP increase more than ten, then start to appear problems like bottleneck performance [2], the effect of
the high electromagnetism interference can disturb the interconnect functionality, the wiring complexity problems, in general no one is flexible.

This issue can be potentially handled by a communication infrastructure based on the Network-on-Chip (NoC), which has better scalability to provide sufficient communication bandwidth [3]. Network-on-Chip infrastructure also enables advanced intellectual properties communication concepts for MPSoC. In embedded MPSoC systems, NoC can provide a flexible communication infrastructure, in which several components such as microprocessor cores, MCU, DSP, memories and other intellectual property components can be interconnected by using reusable NoC routers via general modular interfaces. The MPSoC systems can also be reconfigured for certain embedded computing application and can be customized to improve the communication performance in the application.

Hence, the NoC-based systems combine performance with design modularity. The main component of the NoC system is an on-chip router. Research in the field of off-chip interconnection network is not a new activity. The off-chip interconnection network has been a mature technology. However, there are some issues that should be addressed regarding the adoption of the “off-chip network” concepts into the “on-chip network” implementations. We are sure that the new innovation related to switching method, adaptive routing algorithm, network flow control and buffering scheme suitable for NoCs are still required. Until now, there is no standard for the NoC architecture similar to that of the internet world.

This thesis is motivated to adaptive a new NoC called XHiNoC (eXtendable Hierarchical Network-on Chip) with new switching method, new adaptive routing strategies, a new deadlock-free theory and methodology for tree-based multicast routing, first in the simulation platform called HeMPS (HERMES Multiprocessor System) and then on the FPGA. HeMPS Station is a dedicated environment for MPSoC, basically a concept for a system structure to evaluate the performance of distributed applications in a given architecture running on an FPGA.
1.2 Objectives

The main purpose of this thesis is to develop and evaluate XHiNoC, the new network-on-chip based MPEG application for multiprocessor systems. First using HeMPS simulation and then loading it into a FPGA.

Concretely we deal with two NoC, HeMPS NoC and XHiNoC. Both with different features such as switching method, routing algorithm, network flow control and the internal NoC router pipeline microarchitecture. However, note that basically we work with XHiNoC.

The specific objectives are explained in the following points:

• Know the two NoC, HeMPS NoC and XHiNoC, understand how they work and be able to differentiate their main features
• Become familiar with HeMPS platform, understand its structure and each part of its communication as well as set up all the software for testing.
• Replace HeMPS NoC within HeMPS system by XHiNoC, and adapt all that it entails.
• Simulate the function of the XHiNoC using HeMPS platform, loading different application and applying multi-task in the processors. Afterward check the execution time and the latency with different applications and compare both systems as well. Finally run the whole system inside a FPGA virtex5.

1.2 Thesis outline

The following points present the structure and a brief description of the rest of this report divided into 4 chapters groups, i.e. analysis, design, measures and conclusions:

1 Chapter 2: This chapter describes, first, the two networks on chip that we use. Then explain the objectives and the important aspects that we have to consider. It continuous talking about the possible alternatives and finally there is the justification of the adopted solution.

2 Chapter 3: This chapter describes all the implementation process of Network Interface block divided in two sub-blocks: Network Interface Sending Unit and Network Interface Receiving Unit. The work methodology consists firstly in explain
the objectives and the specification. Then explain every step of the implementation and at the end, show some simulations to corroborate the good working.

3 Chapter 4: This chapter compares the result between the two NoC. Concretely the latency and the execution time.

4 Chapter 5: This is the last chapter and discusses some conclusions drawn during the project.
Chapter 2

Analysis

Contents

2.1 Work tools ........................................................................................................ 11
  2.1.1 HeMPS ........................................................................................................ 12
    2.1.1.1 HeMPS application ........................................................................ 12
    2.1.1.2 HeMPS NoC ..................................................................................... 14
    2.1.1.3 HeMPS Network Interface .............................................................. 17
  2.1.2 XHiNoC ..................................................................................................... 18
  2.2 Important aspects ........................................................................................... 22
  2.3 Alternatives .................................................................................................... 23
  2.4 Justification of the adopted solution ............................................................... 28

Always before you can tackle any development project, you must be documented, know the work environment. This is one goal of this section; know which and what tools we will work and meet system specifications in order to propose alternatives and choose the best one that suits better. Sometimes it is not trivial to know which one is better and then is often used the trial and error method. In the field of computer science, the method is called generate and test. Learning doesn't happen from failure itself but rather from analyzing the failure, making a change, and then trying again.

2.1 Work tools

Multi-Processor Systems-on-Chip are increasingly popular in embedded systems. Due to their complexity and huge design space to explore for such systems, CAD tools and frameworks to customize MPSoCs are mandatory.
2.1.1 HeMPS

2.1.1.1 HeMPS application

MPSoCs are mostly used in embedded systems, usually without a user interface. However, designers should use dedicated environments to evaluate the performance of the distributed applications running in the target MPSoC. Such environments include:

- Dedicated CAD tools running in a given host.
- Fast communication link between host and MPSoC to enable real time debugging and evaluation.
- Monitoring schemes inserted into the MPSoC architecture enabling performance data (throughput, latency, energy) collection.

The goal of HeMPS Station is to give to the designers an environment to help the development and evaluation of embedded applications targeting a NoC-based MPSoC.

Therefore, HeMPS Station is composed by an MPSoC, an external memory and a Host/MPSoC Interface System, as shown in Fig. 2.1 [4].

![Fig. 2.1: HeMPS Station Environment](image-url)
The MPSoC is a master-slave architecture and it contains three elements:

- Interconnection element, which is a NoC [5] with mesh topology.
- The processing elements that comprises a RISC processor and private memory. This element also needs a network interface and a microkernel.
- The monitoring elements. Their functions are to collect information of the network like throughput, latency or energy.

The external memory is a repository attached to the master processor, storing all tasks of the application to be executed in the system.

The Host/MPSoC Interface System manages three elements; the host, the master and the external memory. This system is responsible for a fast communication link between host and MPSoC, for loading code in the external memory, for the messages of debug and control with master and for the host’s software that presents the performance data and translate the applications in code for the platform.

Applications running in HeMPS are modeled using task graphs, and each application must have at least one initial task. The task repository is an external MPSoC memory, keeping all task codes necessary to the applications execution.

The system contains a master processor (Plasma-IP MP) [6], responsible for managing system resources. This is the only processor having access to the task repository. When HeMPS starts execution, the master processor allocates initial tasks to the slave processors (Plasma-IP SL). When each slave processor finishes their tasks, inform to the master about theirs results.

Also, HeMPS station has a HeMPS Generation environment that makes easier to set up all the application. Fig. 2.2 shows the interface.
2.1.1.2 HeMPS NoC

From now on we will focus on HeMPS NoC [7] that is one of the main hardware components from HERMES system.

The HeMPS’s NoC interconnects all microprocessors among them and how I mentioned earlier, the main component is the router. Every microprocessor has one router and every router has 5 ports (north, south, east, west and local). Each router consists of 3 blocks; the buffers, the arbiter and the crossbar. Fig. 2.3 shows the router internal structure [8]. Each port has its particular buffer. The Arbiter unit is utilized to select a packet from a certain incoming port to access its requested outgoing port. The arbiter plays a role as a referee to control contentions between some packets requiring the same outgoing port in the router. Last is the crossbar that interconnects between input and output ports of the router.
The HeMPS’s NoC has the following features:

- **Wormhole packet-switched mesh topology.**

  In the wormhole switching method, messages are divided into a number of flow control digit or commonly called as flit. Every flit may bring a data word. The main advantage of the wormhole switching is that the buffer size can be set as small as possible to reduce the buffering area cost. The main drawback of the wormhole switching method is the problem of head-of-line blocking.

  When flits from one source (S1 for example) occupy or reserve some buffers in the input ports of the network routers. Other messages cannot acquire the reserved buffers until the flits from S1 have released the buffer reservation. The tail flit or the end flit of the message will terminate the buffer reservation.

- **XY routing algorithm.**

  Typically in 2D mesh network, a dimension-order X-First (XY) routing algorithm can be used. The turn model of the static XY routing algorithm is presented in Fig. 2.4 and has been introduced in [9]. As shown in the figure, four turns are prohibited to avoid a deadlock configuration, i.e. North–East, South–East, North–West and South–West turns. Because of the applied prohibited turns, routing algorithm is static whereby packets are always routed firstly to X-direction, then to Y-direction.
Credit-based flow control.

The Fig. 2.5 shows the data structure from HeMPS NoC [8]. It is divided in 3 different packets of 16 bits size. The target or header which informs about the destination, then there is the size that says the payload size and last the payload. The plasma sends the data in flits of 32 bits each one (Fig.2.5a) but the HeMPS NoC works in flits of 16 (Fig.2.5b). So to solve this conflict, each PLASMA-IP is connected to NoC using a Network interface block which adapts the data.

![Fig. 2.4: XY Routing algorithm [3]](image)

![Fig. 2.5: HeMPS NoC data form](image)

The target or header flit of each packet makes a routing direction on each node and reserves a set of routing paths, while the payload flits will follow the path set up by the header flit. During the transmission, others messages cannot acquire the reserved buffers.
So the whole packet is transmitted without interruption due to there is only one slot for each port. The tail flit of each packet at the end will then terminate the reservation. Unfortunately, the main problem of this traditional wormhole switching method is a head-of-line blocking problem. If the header flit is blocked then it will also block the remaining paths that are still used by the wormhole packet.

- **8-flits buffer depth.**

  HeMPS NoC uses 8 flits buffer depth. Depending on the switching method, it allows to set this buffer with more or less slots.

### 2.1.1.3 HeMPS Network Interface

The great objective of *Network interface* is to adapt the signals between PLASMA-IP or core and the NoC router and works with their protocols. In sending to the NoC it must divide the 32 bits of each PLASMA flit in 2 of 16 bits for NoC protocol and in the opposite size, it must packet 2 flits of 16 bits to make 1 of 32 bits. To do it, Network interface uses a memory RAM of 32x16 bits. It is vital to use a RAM to store the information because the core doesn’t read immediately. Fig. 2.6 shows the *Network Interface* block with its signals to connect, On one hand to the PLASMA-IP and for the other to the NoC.

![Network Interface block](image)

*Fig. 2.6: Network Interface block*
2.1.2 XHiNoC

The XHiNoC is developed not only to provide a communication infrastructure for embedded multiprocessor systems-on-chip (MPSoC) devices, but also for further implementation of chip-level multiprocessor (CMP) systems, which are designed for general-purpose multicore computer systems. The main purpose of the XHiNoC is to give a flexible shared communication media of the on-chip interconnection network. In order to provide such flexible communication media sharing, a concept of locally organized packet identity (ID) division multiple access (IDMA) method suitable for NoCs is introduced. Local ID slots are distributed over every communication link, which can be attached to every flit of a packet or data stream as its local ID-tag.

The generic microarchitecture of XHiNoC router is presented in Fig. 2.7 [3]. In general, the router consists of two main component groups; component groups at input and output ports.

![Fig. 2.7: Microarchitecture of XHiNoC router](image)

At every input port, there are a First-In First-Out (FIFO) buffer and a Routing Engine with Data Buffering (REB) components. The REB components consists of three modules; Router Buffer, a Routing Engine (RE) and a Grant Controller (GC).

- The FIFO Buffer is used to buffer data coming from a neighbor to the input of the NoC router. The depth of the FIFO buffer can be set to only 2 registers.

- The RE module is used to make a routing decision such that a message can be routed from an input port to an output port.
• The Route Buffer is used to buffer a message flit soon after the routing decision has been made for the flit.

• The GC module is a combinatorial logic used to control the data read operation of the FIFO buffer.

At each output port, there are two main modules; an Arbiter (A) unit and a Crossbar Multiplexor with ID Management (IDM) Unit (MIM). In each MIM modules, there is an ID Slot Table. The functionality of the output port component group is described briefly in the following items.

• The Arbiter or Arbitration unit at an output is used to select a message flit from an input port that will be switched out to the output port.

• The MIM module is used to multiplex message flits from input ports and concurrently used to update the ID-tag of each message and manages the ID Slot Table in such a way that flits belonging to the same message will have the same ID-tag.

Fig. 2.8 shows the detailed component and architecture of XHiNoC [3].

![Fig. 2.8: Detailed component and architecture of XHiNoC](image-url)
In the following points we mention some characteristics of the XHiNoC that makes it a unique NoC router. Of course there are more interesting features of the XHiNoC but to explain all them would take a long time and it is not the objective of this thesis.

- **Specific Wormhole Switching with Flit-Level Packet Interleaving.**

  One of the special interesting features of the XHiNoC design concept is the implementation of a unique wormhole switching technique where flits of different messages can be interleaved and share the same communication media based on the locally organized message identity. Flits belonging to the same message will always have the same local ID-tag when acquiring a communication media (network link). By applying the flit-by-flit circulating arbitration technique, the wormhole messages can be interleaved at flit-level because every flit brings a local ID-tag to differentiate it from other flits of different flits. The local ID tag is updated by an ID management unit implemented in output port. By using this kind of wormhole switching, the head-of-line blocking problem commonly happen in the traditional wormhole switching can be solved partially without implementing virtual channels.

- **Hold-Release Tagging Policy for Deadlock-Free Multicast Routing.**

  By using the local-ID-based method to switch wormhole packet over the network and by using further the flit-by-flit arbitration technique, a multicast deadlock configuration problem due to a multicast data request dependency can be solved effectively by using a so-called hold/release-tagging-based multicast policy implemented on every NoC router. The multicast conflicts, which potentially lead to deadlock configuration (multicast dependency), are allowed and well organized by using the multicast conflict control and management resulting in a new deadlock-free multicast routing methodology. The multicast flow control is based on the fact that a multicast data will not be released from FIFO buffer at input port if the set of all multicast routing requests has not been granted to access the multiple output ports. If a subset of the requests is granted, then the granted requests will be reset to avoid improper multicast flit replications.
Packet format

The detail packet format and the control bits used in the XHiNoC architecture are presented in Fig. 2.9. The message is split into several flits and has 39-bit width, 32 bits for data word plus 7 extra bits; 3-bit field to define the type of flits and 4-bit field to determine the local identity label or ID-tag of the message. The ID-tag field is set 4 bits, resulting in 16 available ID-tag which conforms to the number of available ID slots on every communication link.

![XHiNoC data form](image)

Fig. 2.9: XHiNoC data form

The types of flits can be identified into four types that are described in the following.

- **Header** flit is a flit that is attached in the probe of a message. As the leading flit, the address of the destination node is written on the bit-field of the flit. The header flit initiates the ID slot reservation including routing table slot reservation and ID tag updating function. The header flit is used basically to establish routing path or to configure connection.

- **Databody** flit is identified as the payload data of a message/stream. Hence, the substantial word of the message is injected into the NoC as databody flits.

- **Tail** flit is used to mark the end of a message/stream. The tail flit is basically used to close the routing path or to terminate the connection.
• **Response** flit is used when a connection-oriented guaranteed-service communication protocol is implemented in the NoC router, where the response flit is sent by a destination node to inform the source node about the status of the connection, i.e. successful or fail. When a best-effort data communication protocol is implemented, the response flit is used to initiate data retransmission when data drop mechanism is allowed, because one free ID tag in the acquired link cannot be allocated for the message, since all available ID slots have been reserved by other messages (run out of ID slots).

Besides these tools, we also use the software “ModelSim” to simulate the whole system and it allows us to check whatever signal that we want.

### 2.2 Important aspects

As mentioned earlier, this thesis is motivated to evaluate the operation and performance of XHiNoC NoC when it is embedded inside of MPSoC and is subject to different workloads. Such as multimedia application which require high communications bandwidth or also in case of multitask.

Once we know, on one side, how HeMPS NoC is structured and interconnected with the core and, for the other side, the same but with XHiNoC, then our task is to replace HeMPS NoC for XHiNoC and adapt the signals and the operation protocol.

Basically there are two important aspects to solve:

1. **Adapt the data form**

Both systems use a completely different data form. On one hand, HeMPS Network-on-Chip uses 16 bits per flit and there are three different flit types; Target, Size and Payload as show Fig. 2.5. But no one have extra bits for information. Also the all flits which belong to the same packet travel together and they arrive to the destination in order.

On the other hand, XHiNoC uses 39 bits per flit, 32 for data and 7 for header. 4 of this 7 are for the concept of locally organized packet identity (ID) division multiple access method and the rest determines the flit type; Header, Databody, Tail and Response as present the Fig. 2.9. In that case not all the flits travel together and it is possible that they arrive disordered at the destination.
2. The interleaved

The XHiNoC wormhole switching technique makes that flits of different messages can be interleaved. So it is possible to receive some flits from different source interleaved at the Network Interface buffer. Hence, the Network Interface block must be able to select all the flits belong to the same messages and order them before delivery them to the core.

After analyzing the main aspects, we are able to know that us task is focused on change, redesign the Network Interface block in order to expire with the aims. So the next step is practically VHDL design.

2.3 Alternatives

Before starting to redesign the Network Interface block is recommended to study the possible alternatives and choose the most suitable.

Concerning the data form, there are two alternatives but each one affects differently on the second aspect.

- The first alternative is to put 32 bits of PLASMA-IP information in each XHiNoC flit and then add 7 bits of ID and flit type. It is represented in Fig. 2.10. Using this alternative then the Network Interface block would be a little bit more complex to solve the interleaved aspect.

Fig. 2.10: Alternative to adapt data form using 32 data bits
• The second alternative to adapt the data form is to keep the actual dimension and put the 16 bits in each XHiNoC flit. That means in every flit there are 7 bits of header, 16 bits of data and 16 extra for information that would be useful for the reception *Network Interface* block and facilitate its work. The Fig. 2.11 depicts this second alternative.

![Fig. 2.11: Alternative to adapt data form using 16 data bits](image)

Regarding the interleaved, there are some points to consider:

1. All flits which belong the same packet could be interleaved with others flits belong different packet that comes from another source.
2. At time to send the packet to the PLASMA-IP is necessary to keep the correct order.
3. Could be the possibility that two different packets come from different source have the same ID-tag at the destination *Network Interface* buffer before delivered them to the PLASMA-IP. The Fig. 2.12 presents one case where all the source want transmit to router C local port. The data from router D is longer than A and B. The ID slot table present in the picture is from Router C local port. In the first vignette we can see how the packet D with identification 1 and packet A with identification 2 have been transmitted but the *Network Interface* block starts with packet D to
send to the PLASMA-IP. In the second vignette we can see how the packet A is transmitted and the identification is released. Finally, in the thirds vignette the packet B has been transmitted using the identification 2 again. Hence, there are two different packets, A and B inside of Network Interface buffer with the same identification.

Fig. 2.12: Conflict case where two different sources have the same ID-tag at the destination

Basically we have opted for two alternatives to solve the interleaved problem.

The first consist in implement a flow control protocol. So each source before start the transmission, send a request to the destination and until don’t received the acknowledge, they must wait (Fig. 2.13a). It doesn’t mean that the network link is reserved, only the destination. The Fig. 2.13b shows how effectively the link is not reserved.
The second alternative there isn’t any flow control, all sources could transmit at the same time but then the received unit of *Network Interface* block must be a little bit more complex to solve the interleaved and it should be able to:

1. Identifies first the *Header* type.
2. Then seek flits which have the same ID-tag, come from the same source and the order is correct.
3. Do it for all *Databody* flits.
4. When find a *Tail* flit, it means the packet is finished and starts again the loop.

To do it, need a look up table to store all information reference of every flit such as the ID-tag, the origin source, the order, the flit type and one bit to know if the slot is free or not. Separately it needs a block RAM to store the data. Fig. 2.14 depicts the look up table structure and Fig. 2.15 the algorithm to solve the interleaved.
Fig. 2.14: Look up table structure

Fig. 2.15: Algorithm to solve the interleaved
2.4 Justification of the adopted solution

Although initially we have considered appropriate to implement the second alternative in order to resolve the interleaved problem. We observed it doesn’t work due to the memory RAM locks the transmission. The Fig. 2.16 represents the lock case.

![Fig. 2.16: Lock case using interleaved algorithm](image)

Initially there is source S1 that is transmitting but from point 1, the source S2 starts as well. Hence the flits arrive interspersed but only the flits from S1 are sending to the PLASMA-IP. The green box indicates the flit has been read and the slot is free. Finally there is a lock situation on point 2 where all slots are full of S2 flits. It shows in case that memory RAM has 8 positions but the case can be extended for 16, 32 and so on. Notice also how in this point the flits from S2 are disorder.

So we have changed and decided to carry out the flow control protocol jointly to pack the data by 16 to 16 bits and keep the rest 16 bits in case that we need to send some extra information. Also this alternative doesn’t increase too much the resources of the FPGA.
Chapter 3

Design

Contents

3.1 Network Interface general diagram ........................................................... 29
3.2 Network Interface Sending Unit ................................................................. 32
  3.2.1 Objective and specification ................................................................. 32
  3.2.2 Implementation .................................................................................. 32
  3.2.3 Simulation ........................................................................................... 35
3.3 Network Interface Receiving Unit .............................................................. 38
  3.3.1 Objectives and specification ............................................................... 38
  3.3.2 Implementation .................................................................................. 38
  3.3.3 Simulation ........................................................................................... 42

Having reached this point and taking the approach of the solution decided, the next step is implemented it. In this Chapter we pretend to explain step by step and block by block how we develop the solution, seeing first the objectives and the specification, then the implementation and finally the simulation of each block. Thus we ensure the perfect function.

3.1 Network Interface general diagram

But before to start, we would like to present a general approach of the design to understand better the next design steps and have a general vision as well.

The following table 3.1 explains the input and output Network Interface signals.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Input</td>
<td>Clock signal.</td>
</tr>
<tr>
<td>Reset</td>
<td>Input</td>
<td>Reset signal actives high level.</td>
</tr>
<tr>
<td>Data_write[31:0]</td>
<td>Input</td>
<td>Data bus that comes from PLASMA-IP and contains information to be transmitted.</td>
</tr>
<tr>
<td>Send_av</td>
<td>Output</td>
<td>This signal actives high level means the Network Interface (NI) is ready to transmit.</td>
</tr>
<tr>
<td>Signal Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>Send_data</td>
<td>Input</td>
<td>It comes from PLASMA-IP and enables the Data_write. Active-high.</td>
</tr>
<tr>
<td>tx</td>
<td>Output</td>
<td>This signal enables Data_out bus to NoC. Also it is active-high.</td>
</tr>
<tr>
<td>Data_out[38:0]</td>
<td>Output</td>
<td>Data bus from NI to XHiNoC.</td>
</tr>
<tr>
<td>Credit_i</td>
<td>Input</td>
<td>Flow control signal from NoC to NI. Activated at low level.</td>
</tr>
<tr>
<td>Clock_tx</td>
<td>Output</td>
<td>Clock for the transmission. It doesn’t work with XHiNoC, only with HeMPS NoC.</td>
</tr>
<tr>
<td>Read_data</td>
<td>Input</td>
<td>Enable signal to read the Data_read bus. Activated at high level.</td>
</tr>
<tr>
<td>Read_av</td>
<td>Output</td>
<td>Signal from NI to PLASMA-IP that informs there are new data to read. Active-high.</td>
</tr>
<tr>
<td>Intro</td>
<td>Output</td>
<td>Signal from NI to PLASMA-IP that just is activated for the first flit of every packet.</td>
</tr>
<tr>
<td>rx</td>
<td>Input</td>
<td>This signal enables Data_IN bus from NoC to NI. It is active-high.</td>
</tr>
<tr>
<td>Data_in[38:0]</td>
<td>Input</td>
<td>Data bus from XHiNoC to NI.</td>
</tr>
<tr>
<td>Credit_o</td>
<td>Output</td>
<td>Flow control signal from NI to NoC. Activated at low level.</td>
</tr>
<tr>
<td>Clock_tx</td>
<td>Input</td>
<td>Clock for the reception. It doesn’t work with XHiNoC, only with HeMPS NoC.</td>
</tr>
<tr>
<td>Config[3:0]</td>
<td>Input</td>
<td>It sets up the number of this NI.</td>
</tr>
<tr>
<td>Packet_ack</td>
<td>Input</td>
<td>This signal belongs to Error data system from PLASMA-IP but just works until HeMPS version 3.8. It means the packet has been received correctly.</td>
</tr>
<tr>
<td>Packet_nack</td>
<td>Input</td>
<td>This signal belongs to Error data system from PLASMA-IP but just works until HeMPS version 3.8. It means the packet hasn’t been received correctly.</td>
</tr>
<tr>
<td>Packet_end</td>
<td>Input</td>
<td>This signal belongs to Error data system from PLASMA-IP but just works until HeMPS version 3.8. It means the end of the packet.</td>
</tr>
</tbody>
</table>

Table 3.1: Input and output signal of Network Interface

The next Fig. 3.1 shows *Network Interface* block with its inputs and outputs. Also the different blocks that it is divided are shown. We can distinguish two bigger sub-blocs: Send and Receive Unit. Inside we can see the exploded and the signal connection among the different sub-blocs as well.
Fig. 3.1: General diagram of Network Interface
3.2 Network Interface Sending Unit

3.2.1 Objective and specification

This part is responsible for sending packages from the PLASMA-IP to XHiNoC. So the objective is received the information from the core, change the data form, adapt and send it though the new NoC. Also, it should control the XHiNoC flow control system.

Before to start the design we must be considered the next points:

- The XHiNoC duty cycle is 50%. That means after to send one flit, we must wait one cycle clock before send the next flit.
- We cannot deliver one flit to the NoC if the signal flow control system is activated. On the other hand it must control the data flow for PLASMA-IP side.
- Among the different types of XHiNoC flits, the RESP has priority over others.
- Due to the flow control protocol that we want to implement to solve the interleaved problem, firstly it must send the HEADER type and until it doesn’t received the permission from the destination, it cannot start to send the rest of the data.

3.2.2 Implementation

The working meteorology of Sending Unit consists in following the states of the state machine for every packet. So there are 5 states: Header XHiNoC, Waiting permission, sending destination, sending 16 higher bits and sending 16 lower bits. The state machine starts in Header XHiNoC state and when it receives the first flit from PLASMA-IP, it sends the Header XHiNoC flit to NoC and goes to the next state that must wait for the permission. When it gets the permission from the destination, sends the DBody XHiNoC flits; first the destination, then the payload size and finally all the data. Exactly the same as HeMPS NoC works. Notice that the PLASMA-IP flit is 32 bits. Hence there are two states to send and adapt this flit in XHiNoC system; sending 16 higher bits and sending 16 lower bits.

In the following point we present the ASM diagram of Sending Unit. Notice the 5 states and see how all the specification such as the 50% duty cycle, the RESP has priority over others or flow control system as well have been respected.
Fig. 3.2: ASM diagram of *NI Sending Unit* (part 1)
ACTION 1

Condition 1: New data

No

Duty_50<='0';
Tx_reg<='0';
Credit_i='1'

Yes

ACTION 3

Sending 16 Higher bits

Condition 1: New data

No

Duty_50<='0';

No

Yes

Send Payload:size=1

ACTION 8

ACTION 7

ACTION 6

ACTION 9

Condition 2: Is there any Response?

Yes

Condition 3: Is the new data stored?

No

No

ACTION 11

CONDITION 5

ACTION 10

Sending 16 Lower bits

Duty_50<='0';
Tx_reg<='0';
Credit_i='1'

Yes

Condition 2: Is there any Response?

Yes

No

ACTION 2

ACTION 10

CONDITION 5

ACTION 11

Send Payload:size=1

Fig. 3.3: ASM diagram of NI Sending Unit (part 2)
**Notes of Network Interface unit sending ASM Diagram**

**CONDITION 1**: send_data = '1' and send_av_aux = '1' (Send_data = '1' means the PLASMA-IP has a new data in the bus and send_av_aux = '1' means the Network Interface is waiting a new data)

**CONDITION 2**: write_punter /= read_punter and Destination_busy = '0' (write_punter/=read_punter means there are some source which want transmit and Destination_busy = '0' means that now the reception block is free)

**CONDITION 3**: send_av_aux='0' (that means the next data is stored in payload_high_low buffer)

**CONDITION 4**: send_data = '1' (That means the new Data is just coming in this clock edge)

**CONDITION 5**: payload_size = '0' (If it is the last flit, then go to start again the loop)

**ACTION 1**: Disenale the signal send_av_aux and store the new data in a buffer called payload_high_low.

**ACTION 2**: Destination_busy <= '1'; => Now the reception block is busy. Make the RESPONSE data form with the XHiNoC type, "1111" ID, the destination and other information and enable the transmission. Also increase the punter of waiting source table.

**ACTION 3**: Make the HEADER data form of the XHiNoC with Header flit type, "0000" ID, the router address, the destination and extension information. Also enable the transmission, change to the next state (Waiting_PERMISSION) and enable the new next data from PLASMA-IP. All the information using to make the XHiNoC data form is stored in payload_high_low buffer because the data has been stored before.

**ACTION 4**: Do exactly the same of ACTION 3 but now the information is in the Data_write bus.

**ACTION 5**: Here it makes the second XHiNoC flit that consist to send the destination, so it is a DBODY type with "0000" ID, the router address, the order of this flit and the last 16 bits are for the destination information. Also enable the transmission and the Signal Send_payload This signal helps us to know that the next flit should be the payload size. Finally change the state.

**ACTION 6**: Here it sends the payload size using a DBODY XHiNoC type and then disenale the signal Send_payload. All the information is stored in payload_high_low buffer because the data has been stored before. Keep the same state to send the first 32 data bits. Also it stores the payload size in the payload_size buffer.

**ACTION 7**: Exactly the same of ACTION 6 but now reading from directly the data_write bus. Keep the same state to send the first 32 data bits.

**ACTION 8**: The same of ACTION 3 but using DBODY flit type and sending the 16 higher bits of 32 bits from PLASMA-IP. Jump to the next state and decrement the payload_size buffer.

**ACTION 9**: The same of ACTION 4 but using DBODY flit type and sending the 16 higher bits of 32 bits from PLASMA-IP. Jump to the next state and decrement the payload_size buffer.

**ACTION 10**: Here it sends the Tail flit type and the 16 lower bits of 32 bits from PLASMA-IP and go back to the initial state. So Header XHiNoC.

**ACTION 11**: The same of ACTION 9 but sending the 16 lower bits of 32 bits from PLASMA-IP and go back to SENDING 16 HIGH BITS state.

**LOOP 1**: This is a loop that every clock edge is checking if the destination is free or not. If it is free, then allows sending Unit makes a RESPONSE flit for the next source that want transmit.

**LOOP 2**: This loop is checking every clock edge and control when the destination gives the permission to start the transmission.

---

**3.2.3 Simulation**

The first simulation aims to show how Sending Unit works (Fig. 3.5). The idea is to send one packets to one destination and see how the system follow the ASM diagram shown in Fig. 3.2 and respect the 50% duty cycle or flow control system as well.

The second simulation represents what could be the destination of the first simulation (Fig. 3.6). While the destination is working to send a packet to another core it can receive the request to receive another packet. Concretely represent the point 3 of the Fig. 3.5.
The simulation starts at point 1 where the NI is waiting for the PLASMA-IP (\textit{trans\_state} = \textit{Header\_XHiNoC}, \textit{send\_av} = 1). When the first flit arrives, point 2, automatically NI sends the Header XHiNoC flit (point 3) and jump to the next state (\textit{trans\_state} = \textit{Waiting\_permission}). While NI is waiting for the answer of the destination, keeps \textit{send\_av} = 1 because it is ready to receive the next flit from the PLASMA-IP (\textit{Payload\_size}) which arrives at point 4, then the NI goes down the \textit{send\_av} signal. Is at point 5 when the \textit{Sending Unit} receives the authorization to continuous the transmission and at point 6 it sends, first, the destination and second, the payload size. At the same time it asked to PLASMA-IP for the next 32 bits. Afterwards the state machine is commuted between \textit{Sending 16 higher bits} and \textit{Sending 16 lower bits} to send the rest of the data as long as the next data is available (marked in yellow). Remember that each PLASMA-IP flit is 32 bits. Notice that after receive a PLASMA-IP flit, NI goes down the \textit{send\_av} signal because it can’t asked the next one until the last is not sent.
In this simulation we can see how the Sending Unit is working. It is sending the 32 bits of the core in two times as indicated point 1. But suddenly at point 2 appears a reception request. The source is stored at waiting table. So the next step is to answer this request due to the destination is not receiving anything and it has priority. After answering (point 3) the destination continues working (point 4). Remember that:

RESPONSE type = “111” = 7 and ID =”1111” = A

DBODY type = “010” = 2 and the ID =”0000” = 0 because all ID in the origin have to start with 0.
3.3 Network Interface Receiving Unit

3.3.1 Objective and specification

This part is responsible for receiving packages from the XHiNoC to PLASMA-IP. So the objective is received the information from the NoC, classify and process it depend on its type and also, it should control the XHiNoC flow control system.

As in Sending Unit, there are some points to be considered before to start the design:

- There are 4 different types of XHiNoC flits and each one should be classified for its back custom processing.
- Only one source can be transmitting due to flow control protocol implemented to solve the interleaved, the rest have to wait their turn that should be according to the order of arrival.
- When it received a RESPONSE flit, immediately it has to alert the Sending Unit.
- The payload should be stored in a RAM memory until it is sent to the PLASMA-IP.
- The PLASMA-IP expects a 32 bits data format.
- Just when there are data to be transmitted, it should alert the PLASMA-IP.
- It also should control the XHiNoC flow data control.

3.3.2 Implementation

To explain this point we prefer to divide it in sub-blocks as shows the Fig. 3.1. At least we can explain it in more detail and everything will be clear and more understandable.

We suggest start with the MUX block. The function of this block is classified the incoming data depend on the flit type (HEADER, RESPONSE or DBODY/TAIL) and decides the back processing.

The Fig. 3.7 shows the ASM diagram of MUX, it is totally asynchronous, so it doesn’t depend on the clock edge.
Before to process the incoming data in Network interface receiving unit, there is a MUX block that select the process that corresponds depend on the flit type.

- **CASE 1**: Store the destination in the WAITING_TABLE because the source wants transmit.
- **CASE 2**: Store the data in the RAM memory.
- **CASE 3**: This is the RESPONSE of the destination that allow the unit sending to start the data transition.

The first case is when it received a Header XHiNoC flit. It means someone want transmit. So the idea is store the source in a buffer called Waiting_table and that is handled by Store the Header block. Then when the Send data to PLASMA-IP block is free, the Sending Unit sends the permission to the source that is store in the next position of this table. Fig. 3.8 shows the ASM diagram of Store the Header block.

Fig. 3.7: ASM diagram of MUX block

Fig. 3.8: ASM diagram of Store the Header block
The second case is when it receives a DBODY or a TAIL flit, so this information must be store in the RAM. This is the job of the Store the DBody/Tail block. It also controls the state of the RAM and decides when the flow control system has to activated. So when the RAM is full it actives the tem_espaco signal that is the same as credit_o. Fig. 3.9 shows the ASM diagram of Store the DBody/Tail block.

**ASM Diagram of PROCESS CASE 2: Store the DBody/Tail in the RAM memory**

**CONDITION 1**: $rx\_RAM = '1'$ and $tem\_espaco = '1'$

($rx\_RAM='1'$ means there is a DBODY or TAIL flit and $tem\_espaco='1'$ means there is still space in the memory RAM)

**CONDITION 2**: $first /= last(Address\_var downto 1) + 1$ (first is the punter to write the data in the RAM and last is the punter to read. When this condition is true means the RAM is full and the XHiNoC flow control system is activated)

**ACTION 1**: it increases the RAM write address.
**ACTION 2**: $tem\_espaco <= '0'$: RAM isn’t full.
**ACTION 3**: $tem\_espaco <= '1'$: RAM is full.

Fig. 3.9: ASM diagram of Store the DDody or Tail block

The depth of the RAM and the punter size as well is configured using one variable called $Address\_var$. Also, we decided to make our own RAM for two reasons; the first is because you can change the depth easy using one variable ($Address\_var$) and the second is because we preferred that the reading process was asynchronous, hence we don’t wait the clock cycles to have the data. For this design we decided to use two RAM memories. Its dimension is calculated by:

$$16 \times (2^{Address\_var} - 1)$$

So each memory has 16 bits wide and a variable depth. When we want to store the data, we use the LSB (Least Significant Bit) of last (punter to write) to alternate the RAM enable. This bit is not connected to manage the internal address. We increase the last bus 1 by 1.
In the other side, when we want to read, directly read 32 by 32 bits. Fig. 3.10 shows the memory connections.

![Fig. 3.10: RAM connections](image)

The third case is when it receives a RESPONSE flit, but it is directly connected with LOOP 2 that appear in Fig. 3.2. It allows to start sending the data.

Another block that belongs in Receiving Unit is Sending data to PLASMA-IP. This block has the function of taking the data from the RAM and sends to the core when he asks for it. When it detects the last flit using the Payload signal, automatically that is transferred to LOOP 1 seen in Fig. 3.2. That means it is ready to start the next transmission. Loop 3 is responsible for warning the core when there are data. Fig. 3.11 shows the ASM diagram of Sending data to PLASMA-IP block.

Sending data to PLASMA-IP block has three states; S0, S1, S2. S0 is the initial state and jump to S1 when the transmission starts. The state S1 is basically to store the payload size which one is decreased while the machine is running in S2 state. When payload size is 0 means the packet is finished and return to S0.
CONDITION 1: Destination_busy='1' (means if the Sending Unit has already send the permission or not)
CONDITION 2: last(Address_var downto 1) /= first (it is waiting for the first flit)
CONDITION 3: read_data = '1' (That means the PLASMA-IP want to read)
CONDITION 4: payload = 2 (it is the last flit of the actual packet)

ACTION 1: In affirmative case the Busy_control goes up because early it begins to receive information.
ACTION 2: Activated Intro signal that means the beginning of the packet
ACTION 3: Store the payload, increase the address to read from the RAM and disenable intro signal.
ACTION 4: Decrement 2 by 2 the payload and increase the address to read from the RAM.
ACTION 5: Active the signal Busy_control to inform that the link is free and it is ready for the next transmission.

LOOP 3: When the Address_in and Address_out mismatch, then alert the PLASMA-IP about the incoming data

Fig. 3.11: ASM diagram of Send the data to the PLASMA-IP block

3.3.3 Simulation
The Fig. 3.12 represents the moment that NI begins to receive whole data after answering the request. From point 1 to point 2 the NI receives all the data. We can see how the two RAM enables are changing depend on whether the data is the 16 higher or lower flit bits (also appear some glitches due to it is asynchronous but they don’t affect the system performance). Notice also that the write punter (\textit{last} signal) is increasing at the same time that the RAM packets 32 bits. Finally see how effectively when the RAM accumulates more than 32 bits the \textit{read\_av} and \textit{Intro} are activated to alert the PLASMA-IP that there are new data to read. When PLASMA-IP reads the first flit then the \textit{Intro} is disenabe.
The aim of this simulation is to show the flow System control. As can be seen from Fig. 3.13, the system is working until point 1 that it stops because the RAM is full, so \textit{tem\_espaco} goes down as \textit{credit\_o}. That means the source has to stop sending until the PLASMA-IP reads. This is what happens at point 2 and 3, the core reads the data and allows the sources to send some flits more.

Both \textit{Sending Unit} simulations like \textit{Receiving Unit} simulations corroborate the proper operation of the NI block. So we take for complete the design chapter and the next step consists in take measures of HeMPS platform with the new NoC working and compared them with the origin HeMPS.
Chapter 4

Measures

Contents

4.1 Latency comparison ............................................................................................. 45
4.2 Execution time comparison .............................................................................. 51

In this chapter we present the result linked directly with the design implemented in the previous chapter. The objective is to compare both systems and to discuss the advantages and disadvantages of each scheme.

This chapter is organized as follows. In Section 4.1, the latency of both communication schemes is compared. The comparison of the execution time is done in Section 4.2.

4.1 Latency comparison

In Fig. 4.1 the latencies for HeMPS NoC and for XHiNoC are shown. Concretely it represents when the master sends the task A of Communication application to the slave and it is formed by 882 flits. Fig. 4.1a shows the latency of HeMPS NoC which is 10800ns from the master NI starts to send the packet to the slave NI receives the last flit. However in Fig. 4.1b shows the latency of XHiNoC which is 18320ns. This big different is because two reason according to the design implemented. First, XHiNoC system must request permission to transmit to the destination and it must wait until doesn’t get the confirmation. The second reason is because XHiNoC works 50% duty. It is easy to think that for both reason XHiNoC latency should be double than HeMPS and this is true. We can find the explanation in the depth of the RAM. While HeMPS has 16 position depth, XHiNoC has 128. This affection can be seen directly in the Fig. 4.1a where credit_o signal goes down because the RAM is full while Fig. 4.1b this signal never goes down.
Fig. 4.1: Latency of one packet of 882 flits
The cause because XHiNoC RAM has 128 position depth instead of 16 as HeMPS RAM is a mix among multi-tasking, the flow control protocol and the cores (slaves and master) because they are half duplex. So when one slave has a multi-tasking, for example task A and task B, first the master sends task A and then the task B to the slave. But while it is sending the second, the slave want to send the confirmation of the first task, that means it stops to read data from the RAM until it can send the whole packet. So the rest of the flits that come from the source are stored in the RAM. But to start sending the confirmation it needs to receive the acknowledge as flow control protocol is implemented. If the RAM is too small it gets full before to receive the acknowledge and the system is blocked. That is why the RAM has 128 positions depth.

Continuing with the latency, the next case that shows Fig. 4.2 is when the packet is smaller than the first case, this time it only has 6 flits and in any case the RAM gets full. While HeMPS needs 180ns to transmit the whole packet, XHiNoC needs 440ns. The yellow arrow shows all the way that XHiNoC has to do before transmit.

(a)

(b) Fig. 4.2: Latency of one packet of 6 flits
Also we can see how affectively the latency of XHiNoC is more than double the HeMPS. Anyway XHiNoC has some features that haven’t been demonstrated until now. This is the aim of the following point. Fig. 4.3a shows the structure that has made and Fig. 4.3b the specific scenery that has simulated.

With this scenery and with the follow simulation we can see how XHiNoC can transmit simultaneously both transmissions due to it doesn’t reserve the network link end to end while HeMPS cannot. HeMPS sends one packet and when it finishes then sends the second. The two sceneries are shown in the Fig. 4.4.
Fig. 4.4: Simulation of scenery of Fig. 4.3
Just to end this latency section, do a simple comparison between the number of flits of different packets and the time that each system spend to transmit it. Through the graph we can observe how for small packets the time that XHiNoC spends for every flit is more than double the time that HeMPS spends. But as the packets are bigger these times are balancing. Of course these results are directly conditioned by the RAM dimensions. If we increased the HeMPS RAM the difference between the two systems would be bigger.

<table>
<thead>
<tr>
<th>Number of flits</th>
<th>HeMPS latency (ns)</th>
<th>XHiNoC latency (ns)</th>
<th>HeMPS: Flit/ns</th>
<th>XHiNoC: Flit/ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>180</td>
<td>440</td>
<td>30</td>
<td>73.3</td>
</tr>
<tr>
<td>12</td>
<td>290</td>
<td>680</td>
<td>24.16</td>
<td>56.6</td>
</tr>
<tr>
<td>408</td>
<td>7180</td>
<td>8950</td>
<td>17.59</td>
<td>24.14</td>
</tr>
<tr>
<td>738</td>
<td>11400</td>
<td>15730</td>
<td>15.44</td>
<td>21.31</td>
</tr>
<tr>
<td>882</td>
<td>10800</td>
<td>18320</td>
<td>12.24</td>
<td>20.77</td>
</tr>
</tbody>
</table>
4.2 Execution time

The execution time or CPU time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services on its behalf. There are a lot of factors that influence the execution time such as the application, implementation defined, the tasks distribution and the multi-tasking as well.

To check the execution time we have done an experiment which consists in choosing an application, for instance Communication application. Communication is divided into 4 tasks: A, B, C and D. Some task like A or B needs to communicate with C before to finish its work. That means the communication is not only master-slave or slave-master but also there are small communication slave-slave.

This experiment involves changing the place of the various tasks using a 2x3 mesh NoC topology and evaluated the execution time. We can see the result below. The last case is an exception that has made to corroborate the conclusion of this section.

\[
HeMPS = 1130115 \text{ ns} \\
- \quad XHiNoC = 1151465 \text{ ns} \\
\underline{21350} \text{ ns}
\]

\[
HeMPS = 1091005 \text{ ns} \\
- \quad XHiNoC = 1112515 \text{ ns} \\
\underline{21510} \text{ ns}
\]
\[ HeMPS = 1082905 \text{ ns} \]
\[ - \quad XHiNoC = 1112515 \text{ ns} \]
\[ 29610 \text{ ns} \]

\[ HeMPS = 1148735 \text{ ns} \]
\[ - \quad XHiNoC = 1168335 \text{ ns} \]
\[ 19600 \text{ ns} \]

\[ HeMPS = 1311555 \text{ ns} \]
\[ - \quad XHiNoC = 1354195 \text{ ns} \]
\[ 42020 \text{ ns} \]

\[ HeMPS = 1145755 \text{ ns} \]
\[ - \quad XHiNoC = 1164925 \text{ ns} \]
\[ 19170 \text{ ns} \]
Making a look at previous results, quickly we can see that always HeMPS is faster than XHiNoC. It is not something new because it could theoretically predict as well as with latency results. However it is interesting to see how, in cases when all the tasks are in the same column, the execution time is smaller than when the tasks are divided into two columns. That is because there is only one way where all the communications must use. In this case XHiNoC is better. Except on the case that there are more than 2 tasks in one core where it isn’t able to work parallel and then the execution time increases.
Chapter 5

Conclusion

The main purpose of this work has been to study two Networks-on-chip and understand the working each, focus on the advantages and disadvantages. From this point began the design chapter that consisted of replacing the HeMPS NoC by XHiNoC and adapt it. It allow us, first, commissioning XHiNoC within an evaluate platform for MPSoC and second, do a comparison between the two intercommunication systems.

Obviously, the decisions taken during the design chapter have affected directly the results and therefore also in the conclusions.

It is true that, theoretically, XHiNoC provides some features which make it better for application that require high bandwidth, as long as the interleaved problem could be solved. Thus it allows us do simultaneous transmissions from different sources to the same destination, either using the same network link or different.

Of course our adopted solution solves the interleaved problem but maybe not the best form to increase the NoC performance. Because it doesn’t allow that different sources can transmit simultaneously to the same receiver but allows they can use the same link. The new system need to request permission and wait for the acknowledge to start the communication. In addition XHiNoC duty cycle is 50%. All this makes that the latency and the execution time as well are bigger than the original network on chip of HeMPS as seen in chapter 4.

Seen the characteristics that XHiNoC gives in opposition of HeMPS and the range of applications that HeMPS evaluation platform contains, I dare say that it hasn’t been the best way to evaluate the functioning of the new NoC because of the few application which were available, the communication of almost all tasks were between Master-Slave or Slave-Master. This means we cannot see the XHiNoC advantages. As we can see in the last part of measures, the system is more efficient when there are simultaneous transmissions.
Anyway is inevitable to use more FPGA resources when you want adapt the XHiNoC. In our case affects the increase of RAM but any case, the fact to solve the interleaved involve a complex processing system.

Another experiment that has done but it haven’t mentioned previously is to packet in one flit 32 bits of data instead of 16. Remember that in our design we left 16 bits for extra information that so far have been wasted. So the conclusion is not because the transmission is faster the core read before. That means the time between you alert the core that there are data to read and the core starts to read is exactly the same. So, we improve on latency but lose on FPGA resource because the RAM memory should be double to avoid the locked situation.

To conclude this thesis, I would like to discuss some directions for future works.

As we seen, our solution is not the most optimal or appropriate for the computational time. So, one way to improve the performance would be solving the interleaved problem in a way more efficient. Although the resource used are bigger.

We could see also, we were limited to the HeMPS platform and its applications because the measures have been done using an old HeMPS version. Therefore, it would be better commissioning the update version of HeMPS. I am sure it incorporates more and complex applications which require a great bandwidth and a great processing capacity.
Bibliography


