A Modular Network Interface Design and Synthesis Outlook

Brahim Attia[1], Abdelkirm Zitouni[1], Wissem Chouchenne[1], Kholdoun Torki[2], and Rached Tourki[1]

[1]Electronics and Microelectronics Laboratory, University of Monastir, Monastir, Tunisia

[2]Circuits Multi-Projets (CMP), 64 Avenue Felix VIALLET, 38031 GRENOBLE cedex, France

Abstract

In recent years, as System on Chip design research is actively conducted, a large number of IPs is included in a system based on a Network on Chip (NoC). Different interfaces' specification of IP cores and different flow controls are used by router. They raise a considerable difficulty for adopting NoC techniques. To facilitate the use of the NoC techniques an efficient design of the network interface (NI) unit that connects the switched network to the IP cores is required. In this paper, we present a new NI architecture for NoC with low latency and jitter constraints. We introduce a new distributed buffer structure that increases area and reduces latency and jitter. We present how we can apply the decoupling between computing and communication by proposing a modular architecture. The low latency and minimal jitter between packets are obtained through the separation between header and payload memories. This separation enables the NI to receive a new packet before the end of the transmission of a previous packet. The modular design is obtained through the separations between injection and extraction path and between IP and network sides. The latter separation allows IPs and NoC to be designed independently from each other. For evaluating the efficiency of this approach, we use AHB standard at the IP side and we use the most three used flow controls in NoC. A performance study was conducted and NI designs were synthesized with ST 0.13µm CMOS technology using four different libraries. Experimental results show that the proposed NIs allow better results in terms of latency, jitter and dissipated power relatively to the current published state-of-the art NI architectures.

Keywords: ***Network Interface, Network on Chip, ASIC, Low latency, Low power.***

1. Introduction

Multi-Processor System-on-Chip (MPSoC) platforms are emerging as an important trend for future SoC design. Such SoCs imply the integration of numerous IP cores performing different functions like processors, DSP, microcontroller, SRAM, DRAM. The number of IP cores embedded in such systems was increased and consequently it has critically increased the amount of on chip communication. The key issue of SoC designs is the communication architecture between heterogeneous components. Most of the communication architectures in current System on Chips are based on buses. However, the bus architecture has its inherent limitations **[1]**, **[2]**, **[3]**.

In order to integrate several heterogeneous components in single SoC, a scalable communication infrastructure must trend to on-chip packet-switched micro-network **[4-5]**, generally known as Network-on- Chip (NoC) architecture. NoC has been proposed to the interconnect problem for highly complex chip. In NoC paradigm a router is used for packet switched communication among on chip cores. Networks are composed of Network Interfaces (NI), which implement the interface to the IP modules, routers, which transport the data from one node to another; and links between routers. IP integration is one of the most challenging works in SoC based NoC design. To cope with the protocol of the interconnect structure, a wrapper or NI must be provided for different IP cores **[6]**. Since different reusable IP cores may not be developed based on the NoC, a NI is required as the interface between IPs and routers.

NIs provides services to the transport layer in the OSI model **[7]**. This is the first layer where offered services are independent from the NoC implementation. It allows IPs and interconnects to be designed independently from each other **[8, 9]**. There exist a number of socket specifications to this end, such as OCP (Open Core Protocol) **[11]**, VCI (Virtual component Interface) **[10]**, AMBA AHB **[12]**, and AMBA AXI (Advanced extensible Interface) **[13]**. Network on Chips are message passing by nature and a Network Interface is then needed. The standardized protocols define the rules for all signaling between IP cores and NoC fabric, while permitting the configuration of specific instances. Our Network on Chip provides a shared-memory abstraction to the IP cores. The Communication is performed using a transaction-based protocol, where the master IP modules issue request messages that are executed by the addressed slave modules, which may respond with a response message. The purpose of NI is the synchronization between IP protocol and router timings, the packaging of IP transactions into flits and vice versa, the computation of routing information, and the buffering of flits to improve performance. Previous studies **[14]** focused to a large

extent on the routing of messages inside the network and proposed many NoC architectures, while little effort has been devoted to optimizing the way the IPs are connected to the network. The fundamental issue in any NI design for NoC is the modularity of the architecture to maximize the reuse and the productivity of SoCs based on NoC. The main reason is related to interoperability support of different protocols. Different IP cores must be connected to the network and the NI logic has to be reused across different core protocols. The specific modules which implement the NoC services must be independent from the type of connected core protocols. Other important issue in NI design is to achieve the modularity with low latency. To decrease the end to end latency between IP cores in SoC based NoC, we must reduce the latency of NoC components (router, Network interface, and Link). The latency of master and slave NIs in request and response data flows and jitter between packets must be kept as low as possible. A low-power design is also essential and important issue for portable or mobile systems. Network on chip will become the main communication platform for this kind of Systems. To address the problem of energy efficient design of NoC, we must decrease the power consumption of NoC components. To reduce NoC consumption, we must reduce the power of NoC components such as NI components. In this paper, we present two novel modular NI architectures with pipelined fashion between IPs and the router of NoC. These NIs allow system designers to send data from IPs to NOC, and vice versa with low latency and power. We present how we can apply the decoupling between computation and communications to achieve the IP modules and interconnections to be designed independently from each other. The proposed NI allows reducing the design time of new systems and hide implementation details of the network. The modularity of design is obtained through a separation between data flows and between IP side and NoC side. The low latency and minimal jitter between packets are obtained through the separation between header and payload memories. Depending on the application constraints, the NoC designer must choose the appropriate technology library to satisfy these constraints. For this reason, we present the synthesis results for area, power, and speed of proposed NIs with different frequency and with different libraries with 130 nm CMOS technology. The paper is organized as follows. In Section 2, the related works are presented. In Section 3, an overview of NoC is given. In Section 4 we describe and detail the architecture of the proposed NIs. Section 5 presents experimental results. Section 6 presents a comparison with other works. Finally in section 7 we conclude the paper.

2. Related Works

There are many works published on the design of novel network architectures as presented by **[14]**, but few publications have addressed particular issues to the design of Network Interface modules.

In **[15]**, the authors compared three schemes of paketization strategy such as software library, on-core and off-core implementation, and related costs in terms of latency and area are projected, showing tradeoffs in these schemes. They insisted that a hardware wrapper implementation has the lowest area overhead and latency.

Another approach is presented in **[16],** where the authors propose a generic architecture model which is used as a template throughout the design process for accelerating the design cycle. The key characteristics of this model are its great modularity and scalability which make it reusable for a large class of applications. Its drawback is that is useful for SoCs based bus only. The authors, in **[17],** define a set of parameters for automatic generation of interface for multiprocessor SoC integration based bus. However, they limited the embedded IP cores to CPUs (ARM7 and MC68000). The designs of wrapper for the application of specific cores still lack generic aspects and only tackle restricted IP cores. An efficient on chip NI ASIC design proposed in **[18],** which offers guaranteed services, shared memory abstraction, and flexible network configuration that implement three standard sockets compatible with aethereal NoC. In **[19],** authors present a generic architecture of network interface and associated wrappers for a networked processor to be used with mesh based NoC architecture. In **[20]**, a micro-network that is a generic, scalable interconnects architecture for system on chip called SPIN is presented. It gives to the system designer a simple view of a single shared address space and provides a variable number of VCI compliant network interfaces for both initiators (masters) and targets (slaves). The authors, in **[21]**, present the design of low latency network interface for mesh based NoC compatible with AHB standard and its associated implementation in Xilinx Virtex5 FPGA board. In **[22],** an OCP compliant NI for the Xpipes NoC was touched upon. The NI has a low area but it supports only a single outstanding read or writes transaction. An OCP compliant NIs for the mesh 2 D NoC was designed in **[23]**. These NIs have a low area and a low latency and they support only a burst precise mode outstanding read and write transaction. The authors, in **[24],** propose a generic and modular architecture of NIs that support many modes of OCP standard and can use Handshake 4phase or Credit-Based flow control for the mesh 2 D NoC and it can be used for other topologies. A NI design for Asynchrony NoC is presented in **[25]**. In **[26]**, authors present a Network Interface Sharing Techniques that can be used to optimize the area of NoC

Architectures. In **[27]**, authors present an FPGA implementation of a shared Network
Interface architecture is proposed to reduce area and power by sharing the buffering resources and using the stoppable clock technique. In **[29],** authors describe a Network Interface design which supports serial-link packet based transmission model for network-on-chip application. The weak point of this design is the highest latency in injection and extraction path. Among the presented works, few approaches as the ones presented in **[16-17]** have proposed a generic architecture model which is used as a template throughout the design process for accelerating design cycle for SoC based bus. Only the works of **[18-19-24]** present a modular design of NI for NoC. This paper proposes a new generic architecture model of network interface which can be used as a template through the design process for accelerating the design cycle for System on Chip that uses a Network on Chip as a communication platform. It addresses the problem of the integration of IP cores in such networks and proposes how we can obtain a modular NI with low latency and power constraint. The proposed modular NI architecture provides a very low Latency in the injection and extraction path for MNIs and SNIs, which is much lower than a software stack implementation and other preview works. It reduces the jitter between two successive packets and allows NI to work without blocking fashion. The NI can receive a new request by Master IP before the end of the transmission of the previous packet. It also allows the reception of a new packet before the end of transaction with the slave IP. We present three different implementations each one uses a specific flow control. Different processors or IP cores must be connected to the network and the NI logic can be reused across different core protocols. When we change the flow control used by the network on chip or the NoC, where the kernel parts are changed and the other components are still unchanged. This paper also presents the impact of the control flow mechanisms and synthesis library on cost and performance of NIs. To prove our concept, we evaluate the cost and performance of the proposed NI architectures by implementing our designs on ASIC. We use the industrial standard for IP cores AMBA AHB as processing elements and a wormhole **[30]** Network on Chip. It is based on the mesh 2D topology and the use of a determinist routing algorithm called XY. We evaluate the area, power, speed, latency, and Throughput of implementing NI tasks that use the most three used flow controls in NoC.

3. Services and Functionalities Provides by the Proposed NoC

The current SoCs predominantly use buses as the one chip interconnects; these standard interfaces have bus based semantics where all nodes connected to the communication medium are defined as masters or slaves, and communicate via transactions. Masters start a transaction by issuing requests; slaves then receive and subsequently process the request. The transaction is completed when the slave responds to the original request. The request-response transaction model matches those used in buses, making it easier to design network interface wrapper around IP blocks that were originally designed to interface with buses. For instance, a processor core will be a master that initiates a new transaction through issuing a write request to a memory module, while the memory module will be the slave that executes the write request and responds with an acknowledgment response. Every transaction in AMBA AHB protocol sends addresses and controls information on the address bus, while data are sent on the data channel on bursts. The write data bus is driven by the bus master during write transfers. The read data bus is driven by the appropriate slave during read transfers. In order to interface the NoC with the tile we utilize a NI, which will have the responsibility of packetizing and depacketizing the cores requests and responses. In the request data flow, the NI has the responsibility of receiving the request from the core interface, preparing the packets and dispatching them to the network logic of the tile. In the response data flow, the NI has the responsibility of receiving the packets from the networking logic and presenting the contents to the core interface. NoC topologies are defined by the connection structure of the switches. We have designed a NoC which is based on the mesh 2D topology **[28]**. The proposed NoC assumes that each switch has a set of bi-directional ports linked to other switches and to an IP core. In the mesh topology used in this work, each switch has a different number of ports, depending on its position with regard to the limits of the network. The use of mesh topologies is justified to facilitate the placement and the routing tasks. We have adopted a synchronous router with five input/output ports (North, East, Local, South and West), having each a bidirectional exchange bus suitable for 2D mesh NoC architecture. The NoC includes 16 nodes and the switching technique used is packet switching. Each switch must have a unique address in the network. To simplify the routing on the network, this address is expressed in XY coordinates, where X represents the horizontal position and Y the vertical position. The data flow through the network is a wormhole routing. This has been chosen due to the small number of buffer required per node and the simplicity of the communication

mechanism (no re-ordering at the destination resources). The NoC uses Credit-Based flow control strategies because it has advantages over Handshake. We have adopted a determinist routing algorithm called XY routing. XY routing algorithm is executed to connect the input port data to the correct output port. A network packet is composed of successive flits. A multi-flit packet is inserted through a header flit, which may be followed by one or more data flits (payload). The first flit of packet includes header information for our case. Each flit is composed of 32 bits data and two control bits, where the 34th bit encodes the beginning of-packet (BOP) and the 33rd bit encodes the end-of-packet (EOP). The header is composed of special fields for the network and special fields for NI and IP. The special fields for adapters and IP will be discussed later.

4. Proposed Network Interfaces

There are two fundamental separations in the NI architecture that enable this modularity: a horizontal one which distinguishes the injection path (request data flow) from the extraction path (response data flow), and a vertical one which distinguishes between the network-dependent and the network-independent (connected component) part. These two parts are referred to as shell and kernel, as proposed in the design of Phillips AEthereal NI [18]. Separation between injection and extraction functions allows easy reuse of dual components in both master and slave NIs, since injection corresponds to packet composition and transmission, while ejection corresponds to packet reception and decoding. Shell and kernel separation through relatively well-defined interfaces is really important for minimizing the effort of supporting different sockets, while keeping a fixed kernel structure and changing only the shell part. Moreover, this separation enables greater flexibility in the packet format that can be configured at instantiation time. Since kernel deals with packet, while shell manages end-to-end protocol transactions, control and data signals are usually driven in parallel. Shell supports flow control to external bus protocols, while kernel handles NoC flow control at hop-by-hop and end to- end level. We have designed two types of NI for AHB based cores for our network-on-chip, named Master Network interface (MNI) attached to master IP and Slave Network Interface (SNI) attached to slaves IP. A master-slave device will need two NIs, an initiator and a target, for operation. Each type of NI is additionally split in two sub modules, one for the request and one for the response data flow or channel (injection and extraction path). These sub modules are loosely coupled: whenever a transaction requiring a response is processed by the request channel, the response channel is notified; whenever the response is received, the request channel is unblocked. The advantage gained by using burst transfers is that the bandwidth is used more effectively, since it is only necessary to send the starting address together with some information about the burst. The longer the burst is the better ratio between data and overhead gets. Another advantage is that the jitter between data flits decreases when adding a burst header to the package, since many flits of data can be sent in sequence. To take advantage of burst transactions the NI needs to package a burst in a package to transmit over the network. However, if a very long burst is packaged into one package, the burst can block a slave core from receiving request from other cores.

4.1 Package format Specification

It has been specified that a package is constructed by flits which are 32-bit wide and the flits sent on the network must apply an extra bit to indicate the beginning and the end of a package. The header flit is a 32-bit word located at the beginning of a request or response packet. It contains information used by the routers of the network and the other information used by network interfaces. The information used by routers of the network is useful for the routing of the packet through the network. They are encoded in the 12 least significant bits (address destination, address source). The information used by network interfaces is useful for Decoding Package. They are encoded between 12 and 31-bit number. It depends on the type of the header (request or response). Seen that the MNI and SNI have different behaviors, the information they need is also different. There are two kind of packet used by our NOC: Request packet and Response packet. The request package header is shown in Figure 3 and spans over one flit. The fields address destination and address source present the address XY of the target and source routers. The field Address_cm presents the address of first case memory of the first target memory cell. The address of first target memory cell will be incremented later by the network interface depending on the size of the word and the AHB burst mode used. Hwrite indicates the type of transfer (read or writes) and the priority of the packet in the network is indicated by Priority field.

Fig. 1 Header of request packet.

The size of the transfer and the number of word to be transmitted are indicated by Hsize and Hburst signal. The most important field in Response packet Header is the return address which indicates the source address of the router to route the response packet. The other bits are reserved for future extensions.

4.1 Master Network Interface architecture

The master network interface (MNI) transforms an AHB request to a request packet AHB/NoC and a response packet NoC/AHB to an AHB response. The tasks of the MNI are to receive requests from the master core, encapsulate the request into a package, transmit packages to the network, receive responses from the network, decapsulate responses and transmit responses to the master cores. Figure 2 illustrates the internal architecture diagram of the MNI. The physical division of the interface is distributed in two parts: Shell (IP master side) and Kernel (NoC router side). The Shell part communicates with master IP respecting the AHB protocol and it is divided into two parts: (Shell Input and Shell Output). The Shell Input Part is composed of three modules called respectively: Routing table, Header builder and Controller fifo. This part handles the receipt and encapsulation of the request in one package. The Shell output Part manages the issue of response to the master IP. The shell presents dependent parts of the resource that is, the dependent parts of the IP master. The kernel part is divided into two parts called Kernel Input and Kernel Output. The kernel output part manages the issuance of requests and communication with the local port on the router by using specific flow control. The kernel input part manages the receipt and decapsulation of responses packets. The kernels present the independent part of the resource that is, the dependent part of the network. Clearly, the proposed architecture of the master network interface is built on two data-flows. One data-flow is the request data flow, where the core is the source and the network is the destination. The second dataflow is the response data-flow where the network is the source and the core is the destination. The request data flow called also injection path performs the transformation of the AHB request into a request packet for our NoC. The response data flow called also extraction path performs the transformation of the response packet provided by our NoC into a response for the AHB IP master.

4.1.1 Injection path

We split the design of injection path into the following parts: the shell input, the kernel output, header memory and payload memory. In this part we will present all modules that perform the services provided by the injection path to allow the transmission of the request packet flits to the network. In the case of writing request, the MNI will first receive the necessary information from the master IP for the building of the header via these input signals. A field of the address bus Haddr will be extracted by the routing table module to provide the XY address of the target router. The header builder module will collect the necessary information that is described in figure 1 to build the header. After the building of the header flit and if the FIFO header is not full, then the header builder will activate the write signal for temporary storage of the header flit. In fact, the MNI is available when the data memory and the header memory are not full. The writing of data flits in the data memory is performed by the controller FIFO in the case of writing request and set the Hready and Hresp signals to the appropriate value. If the master IP is in the busy state, the MNI receives data and waits until the master becomes available.

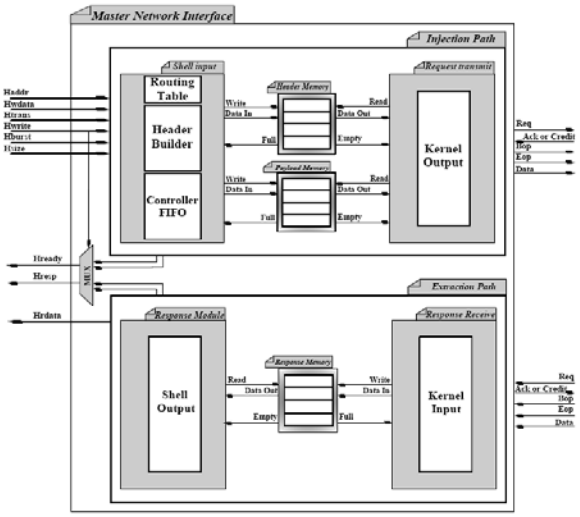

Fig. 2 Master Network interface architecture.

The transactions with the network and the issuance of flits are managed by the kernel output module. First, the transfer begins with the issuance of header flit. If FIFO header is not empty, the kernel output reads a header flit from header memory and transmits it to the local port of the router and set BOP to high state. The kernel output module performs many readings from data memory if it is not empty. The number of readings depends on Hburst and Hsize fields. Two separate pairs of header and payload memories in the MNI injection path are used for temporarily stored flits ready to be injected to the NoC. NI buffers are organized and managed with flit granularity, but the user can decide to continuously inject packets to the network, avoiding wasted cycles. Using separate pairs of header and payload FIFOs decouples the shell from the kernel and provides significant advantages. First, it simplifies size and frequency management that is efficiently implemented through FIFO-based structures.
Second, Actual header size can differ from payload or flit size, so buffering can be optimized and reduced. Third, for components that generate read- or write-only traffic, there is no need to have a payload FIFO in the master NI injection, thus reducing the area complexity depending on the traffic type of the initiator component. Finally, when using a simple flag, the shell is able to simultaneously

store packets while the kernel reads them without mixing flits of different packets in the master NI injection path. In the case of a reading request, the same processing is executed except that the controller FIFO is inactive. We have designed for each flow control, a specific implementation of kernel output module but we use the same implementation for all other modules.

4.1.2 Extraction path

The extraction path is active only when a read command is presented by the IP AHB master. Its task is to receive the response packet that corresponds to the reading request of the master. We split the design of extraction path or response data flow into the following parts: the shell output, the kernel input and the response memory. In this part we will present all modules that perform the services provided by the extraction path to allow the reception of the response packet flits from the network and provide them to the master IP. It is divided into three stages. The first stage presents the kernel input. It is where the data are received from the network. The second stage is the response memory where the data response will be temporary stored. The third stage presents the shell output. It is where the data are transmitted to the master core. After the issuance of response by the network interface slaves, the network routers forward the reply packet to the MNI. The kernel input module presents the dependent part of this network. It manages the reception of flits and the transactions with network according to flow control used by the network. Indeed, the kernel input module writes each received flit until the response memory is not full and returns Ack or credit signal to the local port of the router. The shell output presents the network-independent part. It manages end-toend protocol interactions with the master IP cores directly connected to the NI. When the response memory is not empty, the Shell collects data from response memory and provides it to the master IP and sets hready and hresp signals to the appropriate value. Its role is to manage the emission of data responses to the IP master while taking into account the availability of the response memory and the availability of the IP master. It is responsible for blocking the IP master at the beginning of a read request. After issuing the read request, the master IP is still waiting until the arrival of the response data.

4.2 Slave Network Interface architecture.

The tasks of the SNI are to receive request packages from the network, decapsulate the request packages, transmit the request to the slave core, receive response from the slave core, encapsulate response and transmit response to the network. Clearly, the proposed architecture of the slave network interface is built on two data-flows. One data flow is the request data flow, where the network is the source and the core is the destination. The second data flow is the response data flow where the core is the source and the network is the destination. The request data flow called also extraction path performs the transformation of the request packet of our NoC to an AHB request. The response data flow called also injection path performs the transformation of the AHB response to a response packet to our NoC. Figure 3 illustrates the internal architecture diagram of the proposed SNI. The physical division of the interface is distributed in two parts: Shell and Kernel. The Shell part communicates with slave IP respecting the AHB protocol. This latter plays the role of a master IP since it takes the same decisions as the master. NI shell connects the slave socket of the component to the NI kernel. It manages responses in the injection path and requests in the extraction path. The kernel part is also divided into two parts called Kernel Input and Kernel Output. The kernel output manages the issuance of responses and communication with the local port on the router by using a specific flow control. The kernel input manages the receipt and decapsulation of request packets. Four memories implemented as FIFO are used for the temporary storage of control information and data. Control information is stored in the header memory and payload is stored in the payload memory. The SNI is divided into seven stages. The first stage is where the data are received and decapsulated by the kernel input from network. The second stage is where header flits are buffered by the header memory. The third stage is where the data are buffered by the payload memory. The fourth stage is where the data are transmitted to the slave core by the shell. The fifth stage is where the address source for reading request is buffered by the address source memory. In the case of a reading request, it provides the way to the source router for the address source memory. The sixth stage is where the response data provided by the slave IP via the response memory. The last stage is where the response packet is transmitted to the local port of the router. With the internal architecture diagram of the proposed SNI, several communications between modules proceed; the modules constituting this entity are described as follows:

Kernel input: In the extraction path, incoming packet flits are received by the kernel input and stored in either the dedicated header or the payload buffers. The hop-by hop flow control is managed depending on the availability of free locations in the NI FIFOs. This module is kept in a waiting state until it receives the beginning of a reading or writing request packet from the local port of destination router. It receives the header flit only if the header memory is available and it extracts the various fields necessary from the header for the reformulation of the

request and it stores them in the header memory. For a burst write, this block will make it possible to let a certain number of words of data to be written in the payload memory; this number of words is defined in the hburst and hsize fields. The kernel input will write necessary fields in the header memory if it is not full and a new request is presented at the local port of router. The NI will write the payload flits in the payload memory when a write request is presented. For a burst read, there is only one flit of the header, and then there are not data to receive in the payload memory. The address source will be stored in the address source memory to be used later by the kernel output for response header building. The kernel input works in no blocking mode. The SNI can receive new packets from the local port of the router before that the slave has finished the previous transaction. It only takes into account the availability of two memories in which it stores the received information. A new reading request can be received only if the two memories are not full.

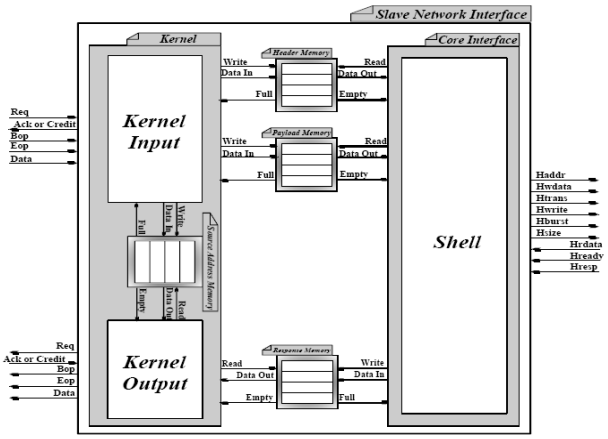

Fig. 3 Slave Network interface architecture.

Shell: it connects the slave socket of the IP component to the NI kernel. It manages responses in the injection path and requests in the extraction path. The shell has to deal with the socket component flow control, address, data and control signals for IP interface. The shell deals with the component data bus size and frequency, while potential adaptation in terms of size and clock speed is handled by the kernel part. In the extraction path the packets coming from the network are organized by the kernel buffering into header and payload, so the shell has to compose the end to end protocol transaction, decode the header field and eventually collect the data. This module plays the role of an IP master interface compared to the IP slave. It rebuilds AHB requests emitted by the initiator IP taking into account the availability of the fields of control in the header memory, as well as the availability of flits in the payload memory in the case of a writing request and the capability of the salve IP to receive a new request. It extracts the necessary fields from the header memory for the reformulation of the request such as (burst type, data size, address, type of command, etc). It provides the slave IP with the necessary phases of address and data to be compatible with AHB standard. The generation of address sequences is obtained by incrementing the first address of memory cell that is provided in the header, incrementing by 1.2, 4 or more depending on the word size. The address and data phases will be extended in the case where slave IPs are not ready to receive a new request. In the case of the presence of control information in header memory, the shell module reads this information and tests on the field Hwrite to determine the type of command. With the burst type, the shell module can specify the number of words to be transmitted on the Hwdata bus or to receive on the Hrdata bus. In the case of a writing request, this module generates a cyclic signal that performs a read from the payload memory to provide writing data bus Hwdata with a new data in case the payload memory is not empty and the IP salve is ready to receive a new data phase. The number of data words to read from this memory is precalculated from the two fields Hburst and Hsize. In the case of a reading request, this module manages the reception of data transmitted by the slave IP. Before beginning a read operation, it must test if the response memory is full or not. If it is not full, it begins the read transfer by storing data temporarily provided by the read data bus Hrdata in response FIFO.

Kernel output: This module has the role of preparing and transmitting the response header, the reading and sending of the response data. The encapsulation of the header is done by the activation of the read signal from the address source memory to get the source address field which will be transmitted with other fields. Then, this module will send the response data already stored in the response memory by the activation of the correspondent read signal. This kernel output will send all the response data which were produced by the IP-AHB and which were stored in the response memory after each beginning of a read request. Moreover, the reading from the response memory is done when it is not empty. The shell implementation is the same for the three implementations in IP side. But, the NI kernel is specific of the flow control to be used.

5. Experimental Results.

In this section the synthesis results will be presented, and a cost analysis of area and power consumption will be made based on the synthesis results. The MNI's performance and

SNI's performance will be evaluated in terms of speed, latency, and throughput. We will present a comparative study of three different implementations for NI. On the IP side the three implementations use AMBA AHB protocol.

The first implementation of NI uses a Handshake 4 phases flow control. The second uses Handshake 2 phases and the third uses the Credit- Based. Master and slave network interfaces with 32 bit AHB data fields and 32 bit network ports have been modeled with VHDL language on RTL level. They were simulated and synthesized respectively by using the ModelSim tool and Synopsys Design Vision tool.

We synthesized these NIs using cell based design with ST 0.13nm CMOS technology using four different libraries (High Speed (HS), High Density High Speed (HDHS), Low Leakage (LL), and High Density Low Leakage (HDLL)). Furthermore, due to the high pin count, the experimental results are based on the circuit simulation of the design instead of the manufactured chip. The synthesis result of the MNI was done with FIFO data and FIFO response having a depth of 4 words of 32 bits and the FIFO header has a depth of 2 words. Each used FIFO has an adjustable depth and width. The synthesis result of the SNI was done with FIFO data and FIFO response having a depth of 4 words of 32 bits and the FIFO header has a depth of 2 words of 19 bits. The FIFO address has a depth of 2 words of 12 bits. For master or slave network interfaces, the Finite States Machine of kernel output and kernel input sub module for each type of control flows is different. The other used sub modules are the same for the three NI versions. Figure 4 and figure 5 show the area of MNIs and SNI for the three implementations with different frequency value. The power consumption results are shown in figure 6 and figure 7. The maximum operating frequency obtained for these NIs implementations is about 1111 MHZ. The result of latency measurement by the simulation of MNIs and SNIs is presented in Table 2. Table 7 shows the measurement of throughput obtained by the simulation of the two versions of the NIs.

5.1 Area of Network Interfaces

The size of the NIs is an important metric because it facilitates calculating the interconnection overhead introduced by the NoC. As a Slave NI or a Master NI should be instantiated for each IP core connected to the network, it is desired that the area is smaller than the IP cores. An exploration of the area/frequency trade off was performed for three NI implementations with 32 bit AHB data fields and 32 bit network ports using respectively Credit-Based, Handshake 2 and 4 phases. By varying the target synthesis clock, different area results were reported (Figure 4 and 5) using four different libraries. The maximum operating frequency achieved with Credit-Based mode module was 1GHz for the MNI and 833MHz for the SNI with High Speed library. The maximum operating frequency achieved with Handshake 2 or 4 phases mode module was 1,111GHz for the MNI and 714MHz for the SNI with High Speed and High Density High Speed Library. The minimal operating frequency achieved with Credit-Based mode module was 588MHz for the MNI and 500MHz for the SNI with High Density Low Leakage library. The minimal operating frequency achieved with Handshake 2 or 4 phases mode module was 600MHz for the MNI and 416MHz for the SNI with High Density High Speed Library.

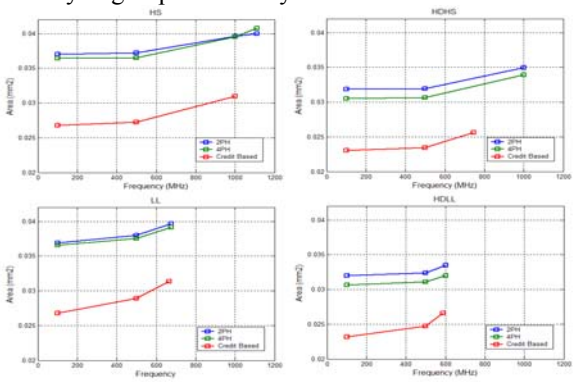

Fig. 4 Area of Master Network interfaces.

The results presented in figure 4 show that the area occupied by the MNI that uses Credit-Based control flow is the most reduced compared to the other modes with the four libraries.

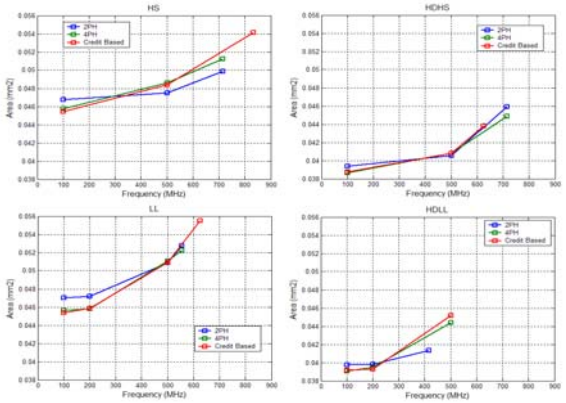

Fig. 5 Area of Slave Network interfaces.

The MNI that uses the 2 phases control flow is the greatest compared to other modes. This is due mainly to the fact that the number of states of the two sub-modules Kernel input and Kernel output in Handshake modes is higher than the number of states of the Credit-Based mode. The area increase is about 32% between Handshake and Credit-Based implementations for LL (Low Leakage) library. The HDHS and HDLL libraries allow obtaining the minimal area compared to other libraries. The HDHS library permits to obtain a small MNI area with high

maximum operating frequency witch achieve 1GHz. The area of MNI that uses Credit-Based mode has as an area 0.025mm^2 and the area of MNI that uses 2 and 4 phases Handshake modes has as areas respectively 0.031mm^2 and 0.030mm^2 at 500MHz. The HDLL library permits to obtain a small MNI area but with low maximum operating frequency witch achieves 600MHz. The area of MNI that uses Credit-Based mode has as an area 0.024mm^2 and the area of MNI that uses 2 and 4 phases Handshake modes has as areas respectively 0.032mm^2 and 0.031mm^2 at 500MHz. By using HS or LL libraries, the area of MNI that uses Credit-Based mode has as an area about 0.027mm^2 and the area of MNI that uses Handshake modes have as area 0.037mm^2 at 500MHz.

For SNI, The results presented in figure 5 show that the area occupied by the SNIs for the three control flow has approximately the same area by using these 4 libraries. The 2 phases control flow has a little difference in terms of area with other modes. It should be noted that a large area of the slave network interface is occupied by the four FIFOs. The HDHS and HDLL libraries allow to obtain the minimal area compared to others libraries. The HDHS library permits to obtain a small SNI area with high maximum operating frequency which achieves 714MHz. The area of SNI has as an area 0.040mm^2 at 500MHz. The HDLL library permits to get a small SNI area but with low maximum operating frequency which achieves 500MHz. The area of SNI that uses Credit-Based mode has as an area 0.045mm^2 and the area of SNI that uses 2 and 4 phases Handshake modes has as areas respectively 0.041mm^2 at 416MHz and 0.044mm^2 at 500MHz. By using HS library, the area of SNI that uses Credit-Based mode has as an area about 0.0483 mm^2 and the area of SNI that uses 2 Phases and 4 phases Handshake modes has as areas respectively 0.0474mm^2 and 0.0486mm^2 at 500MHz. We conclude that the Credit-Based flow control is the best choice for NoC designer to have a NI with a low area constraint without decreasing the maximum operating frequency. These results show that HDHS library allows obtaining a low area with high speed.

5.2 Power estimation of Network Interfaces

The power consumption results are from the Synopsys Design Vision (Power Compiler). An exploration of the power/frequency trade off was performed for three NI implementations with 32 bits AHB data fields and 32 bits network ports using respectively Credit-Based, Handshake 2 and 4 phases. By varying the target synthesis clock, different power estimation results were reported (Figure 6 and 7). When we increase the operation frequencies the dynamic power is automatically increased. The results presented in figure 6 show that the power consumption of the MNI that uses Credit-Based control flow is the most reduced compared to the other modes. The MNI that uses Handshake 2 phases consumes more than other modes. The synthesis with HDLL library permits to obtain the lowest power consumption compared to other libraries. The synthesis with HS library permits to get the highest power consumption compared to other libraries and with important leakage power. The power estimation obtained by synthesis with HDLL library of MNI that uses Credit-Based mode has as power 3.43mW and the estimated power of MNI that uses 2 and 4 phases Handshake modes has respectively 7.67mW and 7mW at 500MHz. By using the HDHS library, we obtain 3, 65mW for credit based mode and 8.35mW and 7.59mW for respectively 2 and 4 phase's modes.

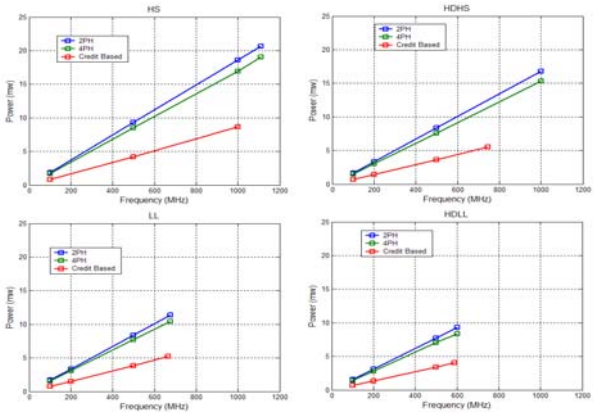

Fig. 6 Power of Master Network interfaces.

For HDHS library, the power increase is about 128% between 2 phase Handshake and Credit-Based implementations.

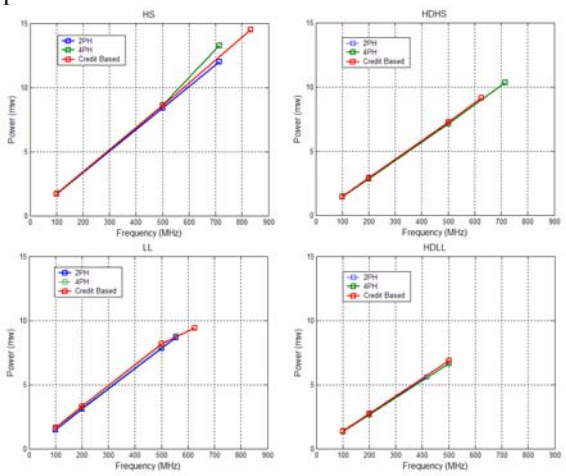

Fig. 7 Power of Slave Network interfaces.

The power increase is about 107% between 4 phase Handshake and Credit-Based implementations. This is due

mainly to the fact that the number of states of the two sub-modules Kernel input and Kernel output in Handshake modes is higher than the number of states of the Credit-Based mode and it has less switching in control signal than Handshake. The power increase is about 10% between Handshake 2 phases and Handshake 4 phases' implementations. For SNI, The three modes have approximately the same power. The synthesis with LL library permits to have the lowest power consumption compared to other libraries. The synthesis with HS library permits to obtain the highest power consumption compared to other libraries and with important leakage power. It should be noted that the power of the slave network interface is dominated by the power consumed by the four FIFOs. We conclude that the Credit-Based flow control is the best choice for NoC designer to have a NI with low power constraint without decreasing the maximum operating frequency.

5.3 Latency of Network Interfaces

For Master Network Interface, the latency for a write or a read request transaction is defined as the number of cycles needed by injection path when the request is presented at the AHB interface to the time when the first flit of the packet leaves the NI. The latency for a read response transaction is defined as the number of cycles needed by the extraction path when the response packet is presented at the local port of the router to the time when the first response appears at the AHB interface.

Table 1: Latency Results

Latency (cycles)		4Ph	2Ph	CB
MNI	Write request	3	3	3
	read request	3	3	3
	read response	7	5	4
SNI	Write request	6	4	3
	read request	2	3	3
	read response	1	3	3

For Slave NI, the latency for a write or a read request transaction is defined as the number of cycles needed by the Request data flow when the request packet is presented at the local port of the router to the time when the first request appears at the AHB interface. The latency for a read response transaction is defined as the number of cycles needed by the Response Data flow when the response is presented at the AHB interface to the time when the first flit of the response packet quits the SNI. The MNI and SNI designs are tested and verified in two phases. In the first phase, the communication from IP to router was tested. In the second phase, the communication from router to IP was tested. The number of clocks to transfer a flit from IP AHB to the router is calculated at different stages and the results are presented in table 2.

Therefore, the time to transfer a complete packet from IP to the router and vice versa is:

Packet Delay = FD + M (N-1) clocks / packet (2)

Where FD present flit delay indicated in table 1, M present the time in cycle to forward a new flit and N present packet length.

Figure 8 presents the Master and Slave Packet delay in clock cycles for different packets lengths respectively for the three different flow control without congestion. These results show that Credit-Based implementations have always the lowest packet delay and the 4 phase have the highest packet delay. This is evident because the time in cycles to forward a new flit (M) for 4ph Handshake flow control is equal to 4, 2 for 2ph Handshake flow control and equal to 1 for Credit-Based.

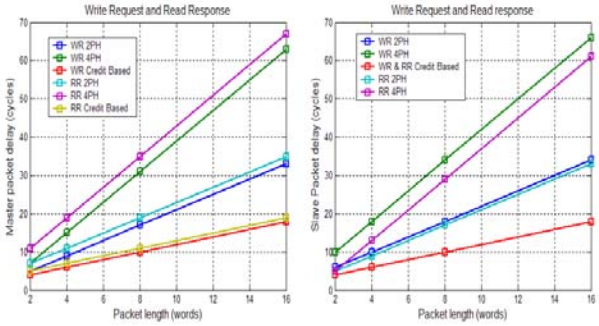

Fig. 8 Packet delay of MNI and SNI for write request and read response.

5.4 Throughput of Network Interfaces

The NI is a bridge between the IP and the NoC. Therefore, the throughput for the NI can be in two directions: the forward direction, from the core to the NoC, and the reverse direction, from the NoC to the core. Table 2 shows the throughput in forward and reverse direction with clock frequency F = 500MHz for MNI and SNI.

Table 2: Minimal throughput Results

Throughput (Gbits/s)		4Ph	2Ph	CB
MNI	Forward direction	5,333	5,333	5,333
	Reverse direction	2,285	3,2	4
SNI	Forward direction	2,666	4	5,333
	Reverse direction	16	5,333	5,333

The throughput for NI in forward direction or reverse direction is defined as the total number of flits processed by NI per second.

Throughput = 1 / latency (Flits / Clock) (3)

Example:

The flit throughput for MNI in forward direction can be calculated as follows:

Throughput = 1 / (3*(1/(500*106)))

= 100MFlits / Second=5333Mbits / Second

5.5 Physical design of 4ph MNI using High Speed (HS) standard-cells library

We designed the master network interface IP using the VHDL language. For logic synthesis and physical ASIC design we used the Synopsys Design Vision and the Cadence Encounter environments respectively. The resulting netlist is used as an input to Cadence in order to perform mapping and routing with a 130nm CMOS technology. The layout of the MNI using 4 phase flow control with HS library is shown in figure 9. The results obtained from these operations are reported in table 3. The latter is a synchronous circuit that operates with 1GHz as the clock rate, which makes it more suitable for real time communications. The MNI occupies a 0,040mm^2 silicon area and integrates 6686 equivalent gates. The correctness of the network interface functionality is verified by using the Synopsys simulator tool.

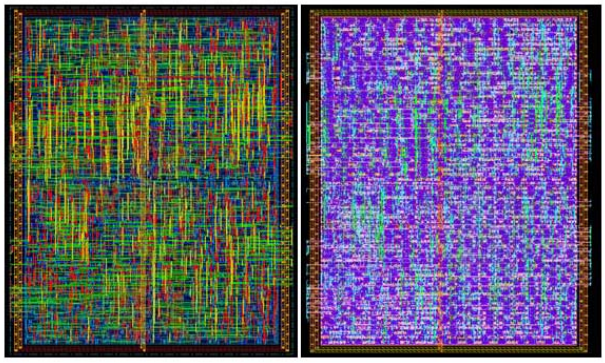

Fig. 9 Layout result of the 4PH MNI routed (a)and Virtuso layout.

The total Input/output is equal to 166. The core dimension of the MNI is about 0.190mm x 0.352mm, and the core is about 0.0575mm^2. In Fig. 10 (a) we show a detailed view of the area consumption of different parts of the master NI.

Table 3: Chip characteristic

Technology	130nm
Buffer size	4 flits
Total Input/output	166
Flow control	Handshake 4 phase
Clock frequency	1GHz
Operation Voltage	1.2V
Power Consumption	Dynamic:16.92mW Cell Leakage: 1mW
Chip Dimension	0.190mm x 0.352mm
Total area of Core	0.0575mm^2
Total area of Chip	0.0682mm^2

The figures show a NI with 2 FIFOs having four words depth and one FIFO having a two words depth. Each FIFO payload and FIFO response has an area of 0.0076mm^2, corresponding to 18.7% of the MNI. The FIFO header has an area of 0.0042mm^2 which presents 10.5% of the MNI. One can note that for this NI instance, a large part of the total MNI area is consumed by the FIFOs (47.9%) as presented in the floor-plan of figure 10(b).The Shell part presents 23.1% of NI area, The Shell input and the Shell output have respectively the area of 0.0018mm^2 and 0.0074mm^2, corresponding to 4.6% and 18.5% of the MNI area. The Kernel part presents 28.15%. The Kernel input and Kernel output have respectively the area of 0.0034 and 0.0079mm^2, corresponding to 8.4% and 19.75% of the MNI area.

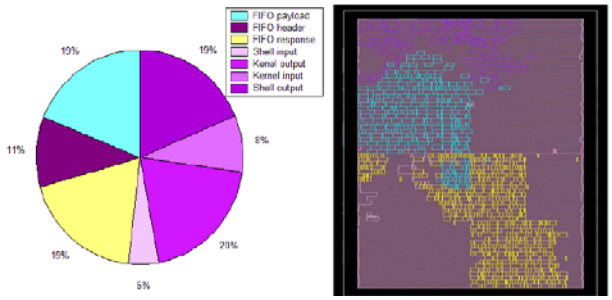

Fig. 10 MNI area detail (a) and FIFO floorplan (b).

We will describe a SoC based on 4x4 mesh 2D NoC. This SoC is composed of 32 IPs (Masters and Slaves), 16 MNIs, 16 SNIs, and 16 routers with five input/output ports. The number of gates count of a single router is about 15191. The number of gates counts of a single MNI and SNI are respectively 6686 and 8034. The gates count of the NoC composed of 16 routers, 16 MNI, and 16 SNI are equal 478576 as shown in equation 4.

16. Router Gate count+ 16. (MNI Gate count + MNI Gate count)=

(4)

16. 15191+16. (6686 + 8034)= 243056+235520=478576

The 32 NIs occupied about 50% of NoC gate count and other gates count are from these 16 routers. Authors in [31] have estimated PEs with complexity of about 50−100Kgates. We will estimate the SoC gate count with different IP gate count and we estimate the percentage of gate count that will be occupied by the NoC. For IP gates count equal to 50Kgates, the NoC will occupy about 23% of SoC gate count. For IP gate count equal to 75Kgates, the NoC will occupy about 17% of SoC gate count. For IP gate count equal to 100Kgates, the NoC will occupy about 13% of SoC gate count.

6. Comparative study

In this section we compare some of our proposed NIs with other architectures. This comparison is presented in Table 4. An exact comparison is complicated due to the fact that these architectures have been implemented with different technologies and exhibit variations in their specifications, protocols and capabilities. Nevertheless, it will be noted that the design presented exhibits the highest level of modularity and flexibility for supporting other standards. The proposed MNI and SNI implementations that use HDHS synthesis library have the best performance compared to the implementations using the other libraries. For this reason we will use this implementation to compare our NIs with other works reported in table 4. For NI in Handshake mode, the maximum operating frequency is about 1GHz for MNI and about 714MHz for SNI in 130nm technology. The maximum frequency of proposed MNI is equal to the work of [22] and outperforms all other works. The maximum frequency of proposed SNI is less than [22-25] and outperforms the works of [18-19]. The area of proposed MNI in Handshake modes is smaller than other works. The area of the proposed SNI is approximately equal to the work of [22], smaller than [18-19-29], and finally bigger than [25].

Table 1: Comparative study

	[29]	[25]	[22]	[18]	[19]	This work (HDHS)		
Protocol	*NA*	*OCP*	*OCP*	*OCP, AXI, DTL*	*NA*	*AHB 2PH*	*AHB 4PH*	*AHB Credit based*
Λ(μm)	*0,13*	*0,13*	*0,13*	*0,13*	*0,09*	*0,13*	*0,13*	*0,13*
Frequency (Mhz)	*NA*	*MNI: 725 SNI: 1086*	*MNI: 1000 SNI: 1000*	*NI: 500*	*NI: 719*	*MNI: 1000 SNI: 714*	*MNI: 1000 SNI: 714*	*MNI: 746 SNI: 625*
Area (mm^2)	*NI: 0.43*	*MNI: 0.058 SNI: 0.020*	*MNI: 0.036 SNI: 0.045*	*NI: 0.169*	*NI: 0.053*	*MNI: 0.034 SNI: 0.045*	*MNI: 0.033 SNI: 0.044*	*MNI: 0.025 SNI: 0.043*
Power (mw)	*NA*	*NA*	*MNI: 33.5 SNI: 36.9*	*NA*	*NI: 15*	*MNI: 16.8 SNI: 10.34*	*MNI: 15.31 SNI: 10.33*	*MNI: 5.55 SNI: 9.14*
Latency (cycles)	*[8,10]*	*[4,6]*	*MNI: 6 SNI : 10*	*[4,10]*	*[4,5]*	*MNI : [3,5] SNI : [3,3]*	*MNI : [3,7] SNI : [6,1]*	*MNI : [3,4] SNI : [3,3]*

For NI in Credit-Based mode, the maximum operating frequency is about 746MHz for MNI and about 625MHz for SNI. The maximum frequency of proposed MNI is smaller than in the work of [22] and outperforms all other works. The maximum frequency of proposed SNI outperforms the works of [18] and smaller than other works. The area of MNI in Credit-Based mode is smaller than other works. The area of proposed SNI is approximately smaller than [18-19-29], and finally bigger than [22-25]. As we also can show then the power consumption of Handshake mode is approximately the one fourth of the Xpipes NI power [22]. We can conclude that our work outperforms the presented other works in terms of power consumption.

The result of latency of the NI of [29] is between 8 and 10 cycles. The latency of [25] NI in injection and extraction path is respectively 4 and 6. In [22], the latency of MNI and SNI are respectively 6 and 10. The latency of AETHEREAL NI [18] is between 4 and 10 cycles. Finally, the Latency of [19] for SINGLE and BLOCK transmission in NI in the injection path are 4 and 5 cycles, respectively. Its latency in extraction path is 5 cycles. For MNI in three modes, the latency in injection path for write or read request is equal to 3 cycles. The latency of extraction path is between 4 and 7. The latency results of the proposed MNI and SNI presented in table 4 shows that the proposed NIs outperforms all other architectures in terms of latency in injection and extraction path for MNI and SNI.

6. Conclusion

This paper presents new network interface architectures that allow IP cores and NoC to be designed independently from each other. The proposed NIs includes three fundamental separations. The first separation is horizontal, one which distinguishes the injection path from the extraction path. The second separation is vertical, one which distinguishes between the IP core side and the NoC side. The last separation is between header and payload memories. The proposed NIs allows the reduction of the end to end latency and packets jitter between IP cores. Three MNI and three SNI implementations were proposed to study the impact of flow control in terms of cost and performance and to prove that if we change the network part we need only to change the kernel part without changing the shell part. It uses respectively Handshake 2 phase, Handshake 4 phase, and Credit-Based flow control. The cost and performance of the proposed NIs are evaluated in terms of area, power, speed, latency, jitter, and throughput. We synthesized these NIs using standard cells based design with ST 130nm CMOS technology using four different libraries. The results demonstrate that HDHS library permits to obtain better results than other libraries in terms of area, power, and speed. The Credit-Based flow control permits to obtain the best performance in terms of area, power, latency, jitter, and throughput. The 2 and 4 phase's implementation allow obtaining the best speed compared to the credit based implementation. We present an instance of ASIC design of NI that uses 4phase flow control and synthesized by the High Speed library, which shows that the cost of implementing our NI in hardware is small ($0.057mm^2$ after layout in a 130nm technology, running at 1 GHz and consume about 18mW). A comparative study has been conducted with other works .The obtained results show that the proposed NIs outperforms other works in terms of latency and power. The long-term objective is to develop a tool that

automatically generates a specific application of NI which accepts as inputs the IP core interface specifications(OCP,AHB,AXI,DTL) and NoC parameters like (flow control, routing algorithm, flit width, queuing technique,..).

References

[1] J. Liang, S. Swaminathan, and R. Tessier, "ASOC: a scalable, single chip communication architecture", 9th International Conference on Parallel Architectures and Compilation techniques, 2000, pp. 37-46.

[2] W.J. Dally, "Virtual channel flow control," IEEE Trans. on Parallel and Distributed Systems., Vol.3, No. 2, 1992, pp. 194-205.

[3] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on Chips: Implementation and evaluation on Hermes NoC", in SBCCI, 2005, pp. 178-183.

[4] L. Benini, and D.G. Micheli, "Network on Chips: A New SoC paradigm," IEEE Computer., Vol.35, No. 1, 2002, pp. 70-78.

[5] S. Kumar, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A Network on Chip Architecture and Design Methodology", in IEEE Computer Society Annual Symposium on VLSI, 2002, pp. 117-124.

[6] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, and A.A. Jerraya, "A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design", in CODES, 2001, pp. 195-200.

[7] M.T. Rose, The Open Book: A practical Perspective on OSI, Prentice Hall, 1990, Upper Saddle River, NJ, USA.

[8] K. Keutzer, A.R. Newton, J.M. Rabaey, and A.S. Vincentelli, "System-level design: Orthogonalization of concerns and Platform-based design," IEEE Trans. On CAD of Integrated Circuits and Systems, Vol. 19, No 12, 2000, pp. 1523-1543.

[9] M. Sgroi, M. Sheets, M. Mihal, K. Keutzer, S. Malik, J. Rabaey, and V.A. Sangiovanni, "Addressing System on Chip interconnect woes through communication based design", in DAC, 2001, pp. 667-672.

[10]Virtual Socket Interface Alliance., Virtual Component Interface, draft specification, 1997, V 2.2, htpp://www.vsia.com.

[11]OCP-IP Association., Open Core Protocol specification, 2003, Release 2.0, htpp://www.ocpip.org.

[12]ARM Corporation. AMBA AHB Protocol specification, Version 2.0, 1999, htpp://www.arm.com.

[13]ARM Corporation. AMBA AXI Protocol specification, Version 1.0, 2004,htpp://www.arm.com.

[14] E. Salminen, A. Kulmala, and T.D. Hamalainen, "Survey of Network-on-Chip Proposals", OCP IP association white paper, 2008.

[15] P. Bhojwani, and R. Mahapatra, "Interfacing cores with on chip packet switched Networks", in VLSI, 2003, pp.382-387.

[16] A. Baghdadi, D. Lyonnard, N. Zergainoh, and A.A. Jerraya, "An efficient architecture model for systematic design of application-specific multiprocessor SoC", in DATE, 2001, pp. 55-62.

[17] D. Lyonnard, S. Yoo, A. Baghdadi, and A.A. Jerraya, "Automatic generation of application specific architecture for heterogeneous multiprocessor System on Chip", in DAC, 2001, pp. 518-523.

[18] A. Radulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage, "An efficient on-chip network interface offering guaranteed services, shared memory abstraction, and flexible network configuration", in DATE, 2004, pp. 878-883.

[19] E.L. Seung, H.B. Jun, S.Y. Yoon, and N. Bagherzadeh, "A Generic Network Interface Architecture for a Networked Processor Array (NePA)," ACS, Lecture Note in Computer Sciences, 2008, pp. 247-260.

[20] A. Adriahantenaina, H. Charlery, A. Greiner, and L. Mortiez, "SPIN: A scalable, packet switched, on-chip micro network", in DATE, 2003, pp. 70-73.

[21] B. Attia, W. Chouchene, A. Zitouni, N. Abid, and R. Tourki, R., "Design and implementation of low latency network interface for Network on Chip", in IEEE International Design & Test Workshop, 2010, pp. 37-42.

[22] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. De Micheli, "Xpipes Lite: A Synthesis Oriented Design Library for Networks on Chips", in DATE, 2005, pp. 1188-1193.

[23] B. Attia, A. Zitouni, and R. Tourki, "Design and implementation of network interface compatible OCP for packet based NoC", in IEEE International Conference on Design and Technology of Integrated Systems on Nanosacale Era, 2010, pp. 1-8.

[24] B. Attia, A. Zitouni, N. Abid, and R. Tourki, "A Modular network interface adapter design for OCP compatibles NoCs," International Journal of Computer and Network Security (IJCNS), Vol. 1, No 2,pp 101-109.

[25] T. Bjerregaard, S. Mahadevan, R.G. Olsen, and J. Sparso, "An OCP Compliant Network Adapter for GALS based SoC Design Using the MANGO Network-on- Chip", in SoC, 2005, pp. 171-174.

[26] A. Ferrante, S. Medardoni, and D. Bertozzi, "Network Interface Sharing techniques for Area Optimized NoC Architectures", in DSD, 2008, pp. 10-17.

[27] B. Attia, W. Chouchene, A. Zitouni, and R. Tourki, "Network interface Sharing for SoCs based NoC", in International Conference on Communications, Computing and Control Applications, 2011, pp. 1-6.

[28] B. Attia, W. Chouchene, A. Zitouni, N. Abid, and R. Tourki, "A Modular Router Architecture Design For Network on Chip", in 8th International Multi-Conference on Systems, Signals and Devices, 2011, pp. 1-6.

[29] Y.L. Lai, S.W. Yang, M.H. Sheu, Y.T. Hyang, H.Y. Tang, and P.Z. Huang, "A High-Speed Network Interface Design for Packet-Based NoC", in ICCCS, 2006, pp. 2667-2671.

[30] S. Felperin, P. Raghavan, and E. Upfal, "A theory of wormhole Routing," IEEE Trans. on Computer, Vol. 45, No 6, pp. 704-713.

[31] D. Sylvester, and K. Keutzer, "Impact of small process geometries on micro architectures in systems on chip," in proceedings of the IEEE., Vol. 89, No 4, pp. 467-489.