A Low Latency and Power ASIC Design of Modular Network Interfaces for Network on Chip

Brahim ATTIA1, Abdelkrim ZITOUNI2, Kholdoun TORKI3 and Rached TOURKI4

1Electronics and Microelectronics Laboratory, University of Monastir, Faculty of Science of Monastir, 5019, Tunisia
E-mail: brahim.attia@yahoo.fr abdelkrim.zitouni@fsm.rnu.tn rached.tourki@fsm.rnu.tn

2Circuits Multi-Projets (CMP), 64 Avenue Felix VIALLAET, 38031 GRENOBLE cedex, France
E-mail: kholdoun.torki@imag.fr

Abstract: The implementation of a high-performance Network on Chip (NoC) requires an efficient design of the network interface (NI) unit that connects the switched network to the IP cores. However, different interfaces’ specification of integrated components and different flow control is used by NoC router raises a considerable difficulty for adopting NoC techniques. The architecture of NIs must be modular to allow IPs and interconnections to be designed independently from each other. The power of this NI should be small and its latency must be kept as low as possible. Previous studies show that the decrease of end to end latency between IPs cores can be done by reducing the latency of NoC components and decreasing the jitter between successive packets. In this paper, we present new modular NI architectures between IPs and router with low power and latency. Proposed NIs allows systematic design flow for accelerating the design cycle and hide implementation details of the network. The modular design is obtained through two separations between data flows and IP side and the network side. The low latency and jitter reduction between successive packets are obtained by a separation between header and payload memories. The low power is obtained through the implementation of these NIs using low power library with 130 nm technology. Experimental results show that the proposed NIs outperform conventionally architecture in terms of power and latency.

Keywords: Network on Chip, Network Interface, Low latency.

1. INTRODUCTION

A big challenge of current and future chip design is how to integrate components with millions of transistors and make them operate efficiently. System-on-chip (SoC) designs provide such an integrated solution to various complex applications. One of the key issues of SoC designs is the communication architecture between heterogeneous components. Most of the communication architectures in current SoCs are based on buses. However, the bus architecture has its inherent limitations [1], [2], [3]. For nowadays and the next-generation SoC design, the wiring delay, noise, power dissipation, signal reliability and synchronization are far more serious than ever. A packet-switched network which delivers messages between communicating components has been proposed as the solution for SoC design. Such network-on-chip (NoC) provides a high performance communication infrastructure. NoC is a new paradigm for integrating a large number of IPs cores for implementing a SoC [4-5]. In NoC paradigm a router-based network is used for packet switched communication among on chip cores. Since the communication infrastructure and the cores from one design can be easily reused for a new product, NoC provides high possibility for reusability. Networks are composed of routers, which transport the data from one node to another; links between routers, and Network Interfaces (NI), which implement the interface to the IP modules. The role of NI is similar to the network card required for connecting a PC to the Internet. One of the key components for on-chip networks is the wrapper for different IP cores in the tiles [6]. Since different reusable IP cores may not be developed based on the on-chip network, a wrapper is required as the interface between the IP core and its associated router. NI must provide services at the transport layer in the ISO-OSI reference model [7], because this is the first layer where offered services are independent from the network implementation. This is a key ingredient in achieving the decoupling between computation and communication [8, 9], which allows IP modules and interconnects to be...
designed independently from each other. There exists a number of socket specifications to this end, such as VCI (Virtual component Interface) [10], OCP (Open Core Protocol) [11], AMBA AHB [12], and AMBA AXI (Advanced extensible Interface) [13]. Since most NoCs are message passing by nature, an NI is needed. NOCs have to adhere to standardized protocols so that they can plug and play with IP blocks that were also designed to interface with the same standard. Such standardized protocols define the rules for all signaling between the IP blocks and the communication fabric, while permitting the configuration of specific instances. Our NoC offers a shared-memory abstraction to the IP modules. 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 NoC timings, the packaging of IP transactions into NoC 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, while little effort has been devoted to optimizing the way the IPs are connected to the network. We believe that the fundamental issue in any NI design for NoC is the modularity of the architecture and the micro architecture to maximize the reuse and the productivity of SoCs. The main reason is related to the protocol of interoperability support: different processors or IPs 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 network services must be independent from the type of connected core protocols. Other important issue in network interface design is to achieve the modularity with low latency. To decrease the end to end latency between IPs 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 a two novel modular NI architecture with pipelined fashion between IPs and 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 decoupling between computation and communications to achieve the IP modules and interconnections to be designed independently from each other. The proposed NI allows reducing design time of new systems and hide implementation details of the network. The modularity of design is obtained through 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. The low power design is obtained through the synthesis of the design by using the 0.13 µm technology with low leakage library. The paper is organized as follows. In Section 2, the related works is 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 is a number of works published on the design of network architectures [14], but few publications have addressed particular issues to the design of an NI module. Bhojwani and Mahapatra [15] 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. Baghdadi [16] proposed a generic architecture model which is used as a template throughout the design process accelerating design cycle for SoC based bus. Lyonnard [17] defined 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. In [18] an NI ASIC implementing standard sockets was presented for the Athereal NoC. Seung [19] presents a generic architecture of network interface and associated wrappers for a networked processor.
array. In [20] an NI implementing VCI standard interface was presented for the SPIN NoC. In [21], a FPGA implementation of Network interface for AHB standard was presented for mesh NoC. The NI however, has Low latency in forward and backward direction. 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 transaction. In [23] an OCP compliant NIs for the mesh NoC was designed. These NIs have a low area and low latency and they support only a burst precise mode outstanding read and write transaction. In [24] a generic architecture is presented to provide any mode of NIs compliant OCP for the mesh NoC and it can be used for other topologies. In [25] an NI design for Asynchrony NoC is presented. In [26] Network Interface Sharing Techniques is used for Area Optimized NoC Architectures. In [27], an FPGA implementation of a shared Network Interface architecture is proposed that reduces area and power by sharing the buffering resources and using the stoppable clock technique. In [28] a monitoring run-time system activities in adaptive NoC based on MPSoC platforms through the observation of transactions performed on the communication subsystem. The Collecting Information about system activity is implemented within OCP/IP compliant NIs. 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 [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. Other works [15-20-22-23-25-26-27-28] present a specific implementation for NI for NoC using different protocols. Only the works [18-19-24] present a modular design of NI for NoC. This paper proposes a generic architecture model which is used as a template through the design process for accelerating design cycle for SoC based NoC. It addressed the problem of generating modular NI for NoC with low latency and power constraint.

The main contributions of this paper are:

(1) A description of a new modular NI architecture which allows accelerating the design cycle and a proposal of a systematic design flow for an application specific interface. It includes two fundamental separations that enable the modularity of the design. 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 kernel and shell part. Different processors or IPs cores must be connected to the network and the NI logic can be reused across different core protocols. The logic to manage the network services is independent from the type of connected core protocols. We must change only the shell part and FIFOs memories parameters. When we change the flow control used by the network on chip, where the kernel parts are changed and the other components are still unchanged. The proposed architecture allows reducing design time of new systems and hides implementation details of the network.

(2) 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 also reduces the jitter between two successive packets and allows NI to work without blocking fashion. 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 request by Master IP before the end of the transmission of previous packet. It also allows the reception of a new packet before the end of transaction with Slave IP. To prove our concept, we evaluate the cost and performance of the proposed NIs architecture 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 deterministic routing algorithm called XY. We evaluate the area, power, speed, latency, and Throughput of implementing NI tasks that use the most three used flow control in NoC.

3. SERVICES AND FUNCTIONALITY PROVIDED BY 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. The NI has the responsibility of (i) receiving the contents from the core interface, preparing the packets and dispatching them to the network logic of the tile and (ii) 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. 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 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 bi-directional 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 INTERFACE

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 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.

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.

(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. 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.

(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

Figure 2: Master Network Interface Architecture
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-to-end 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.

**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 slave 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 pre-calculated 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, throughput, and jitter. 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 [6] tool and Synopsys Design Vision [7] tool.

We synthesized these NIs using cell based design with ST 0.13µm CMOS technology using the Low Leakage library. 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 675 MHZ. The result of latency measurement by the simulation of MNIs and SNIs is presented in Table 1. Table 2 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 and 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).
The maximum operating frequency achieved with Credit based mode module was 666 MHz for the MNI and 625 MHz for the SNI. The maximum operating frequency achieved with handshake 2 or 4 phases mode module was 675 MHz for the MNI and 555 MHz for the SNI.

The area of MNI that uses credit based mode has as area $0.028 \text{ mm}^2$ and the area of MNI that uses handshake modes has as area $0.037 \text{ mm}^2$ at 500 MHz. We show from these results that the area occupied by the MNI that uses credit based control flow is the most reduced compared to the other modes. The two other modes (handshake 2 and 4 phase) have approximately the same area. The area increase is about 32% between 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.

For SNI, The three modes have approximately the same area of $0.051 \text{ mm}^2$ at 500 MHz. In low frequency at 200 MHz, we have a little difference between 2Ph (area about $0.047 \text{ mm}^2$) and other modes (area about $0.045 \text{ mm}^2$). The area increase is about 4%. It should be noted that a large area of the slave network interface is occupied 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 area constraint without decreasing the maximum operating frequency.

5.2. Power Estimation of Network Interfaces

In power consumption there are two main components; dynamic and static. The following equation shows the dynamic power component:

$$P_d = \alpha C_L V_{dd}^2 f$$  \hspace{1cm} (1)

The first term denotes the dynamic power, $\alpha$ is the activity of the circuit, $C_L$ is the parasitic capacitance, $V_{dd}$ is the power supply and $f$ is the operating frequency. The dynamic component is a quadratic function of $V_{dd}$ and the sub threshold component is an exponential function of $Vt$ which makes it crucial for the coming deep submicron technologies. By reducing $V_{dd}$ one can drastically reduce the dynamic component, but unfortunately this is at the expense of speed degradation.

The power consumption results are from 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 Power estimation of MNI that uses credit based mode has as power 3.8 mw and the estimated power of MNI that uses handshake modes is in average equal to 8 mw at 500 MHz. We show from these results that the power consumption of the MNI that uses credit based control flow is the most reduced compared to the other modes. The two other modes (handshake 2 and 4 phase) have approximately the same power.

The power increase is about 110% between 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 MNI that uses handshake 4 phases flow control has as power 7.7 mw and the MNI that uses handshake 2 phases modes are equal to 8.4 mw at 500 MHz. The power increase is about 9 % between handshake 4 phases and handshake 2 phases implementations. We conclude that the power consumption of NI that uses 4 phase-handshake mode is lower than that of 2 phase- modes.

For SNI, The three modes have approximately the same power. For example, when the frequency
is fixed at 200 MHz, we have a little difference between 2Ph (power about 7.8 mw), 4Ph (power about 8 mw) and credit based (power about 8.2 mw). The power increase is about 4% between 2Ph and credit based implementations. The power increase is about 2% between 2Ph and 4 Ph implementations. 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 the Request data flow when the request packet 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 response transaction 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:

\[ \text{Packet Delay} = FD + M(N-1) \text{ clocks / packet} \]

FD: flit delay indicated in table 1.
M: time in cycle to forward a new flit.
N: packet length.

For 4ph handshake flow control M is equal to 4. For 2ph handshake flow control M is equal to 2 and for credit based M is equal to 1.

For example, for write request of SNI with the packet length is equal to 5.

\[ \text{Packet Delay (4ph)} = 6 + 4(5-1) \text{ clocks/packet} = 22\text{clocks/packet} \]
\[ \text{Packet Delay (2ph)} = 4 + 2(5-1) \text{ clocks/packet} = 12\text{clocks/packet} \]
\[ \text{Packet Delay (cb)} = 3 + 1(5-1) \text{ clocks/packet} = 7\text{clocks/packet} \]

For example, for write request of MNI with the packet length is equal to 5.

\[ \text{Packet Delay (4ph)} = 3 + 4(5-1) \text{ clocks/packet} = 19\text{clocks/packet} \]
\[ \text{Packet Delay (2ph)} = 3 + 2(5-1) \text{ clocks/packet} = 11\text{clocks/packet} \]
\[ \text{Packet Delay (cb)} = 3 + 1(5-1) \text{ clocks/packet} = 7\text{clocks/packet} \]

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

<table>
<thead>
<tr>
<th>Throughput (Gbits/s)</th>
<th>4Ph</th>
<th>2Ph</th>
<th>Credit based</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNI Forward direction</td>
<td>5,333</td>
<td>5,333</td>
<td>5,333</td>
</tr>
<tr>
<td>Reverse direction</td>
<td>2,285</td>
<td>3,2</td>
<td></td>
</tr>
<tr>
<td>SNI Write request</td>
<td>2,666</td>
<td>45,333</td>
<td>5,333</td>
</tr>
<tr>
<td>Read request</td>
<td>16</td>
<td>5,333</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 shows the throughput in forward and reverse direction with clock frequency \( F = 500 \) MHz for MNI and SNI.

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

**Throughput** = \( \frac{1}{\text{latency (Flits / Clock)}} \) (3)

Example:

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

\[
\text{Throughput} = \frac{1}{(3\times(1/(500\times10^6)))} = 100 \text{ MFlits / Second} = 5333 \text{ Mbits / Second}
\]

5.5. Jitter of Network Interfaces

For burst read and write, the jitter is measured as the latency between data-word in the burst on the NOC side (the delta time between two data-words). The jitter is measured for a write request and a read response. This is the same as the time it takes to make the synchronization in the NI. We can see that the disadvantage of using handshake 4 phases mechanism is that it requires at least two cycles of clock to be carried out and at least 2 cycles for jitter. On the other hand, this mechanism is simple to set up. The use of handshake 2 phases mechanism requires at least one cycle of clock to be carried out and at least one cycle for jitter. As long as the receiver is free, the credit based flow control requires one clock cycle to transmit a data and zero cycle for jitter. The jitter between two successive packets for master or slave NI is about 2 cycles for all flow control modes.

Table 4

<table>
<thead>
<tr>
<th>Comparative Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>[29] [25] [22] [18] [19] This work</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>NA</th>
<th>OCP</th>
<th>OCP</th>
<th>OCP, AXI, DTL</th>
<th>specific</th>
<th>AMBA, AHB</th>
</tr>
</thead>
<tbody>
<tr>
<td>technologies</td>
<td>0,13\mu m</td>
<td>0,13\mu m</td>
<td>0,13\mu m</td>
<td>0,13\mu m</td>
<td>90 nm</td>
<td>0,13\mu m</td>
</tr>
<tr>
<td>Frequency (Mhz)</td>
<td>NA</td>
<td>MNI:725</td>
<td>MNI:1000</td>
<td>MNI:0.036</td>
<td>NI:169</td>
<td>MNI:0.031</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>NI: 0.43</td>
<td>MNI:0.058</td>
<td>MNI:0.045</td>
<td>MNI:0.169</td>
<td>NI: 0.053</td>
<td>MNI:0.055</td>
</tr>
<tr>
<td>Power (mw)</td>
<td>NA</td>
<td>NA</td>
<td>MNI:33.5</td>
<td>NA</td>
<td>NI: 15</td>
<td>MNI: 5.2</td>
</tr>
<tr>
<td>Latency (cycles)</td>
<td>[8,10]</td>
<td>[4,6]</td>
<td>MNI:5</td>
<td>[4,10]</td>
<td>[4,5]</td>
<td>MNI: [3,4]</td>
</tr>
</tbody>
</table>

6. COMPARISON OF THE PROPOSED ARCHITECTURE WITH OTHER NI WORKS

In this section we compare our proposed NI 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 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 the credit based flow control have the best performance compared to the handshake flow control implementations. For this reason we will use this implementation to compare our NI with other works reported in table 4.

The maximum operating frequency for our design is about 666 MHz in 0.13 \mu m technology. The maximum frequency of our work outperforms the work of [18]. The latter supports three IPs at the same time but our work supports only a single IP.
The speed of [25-22-19] outperforms our design but if we synthesize our work in 90 nm technology we can outperform the work of [19]. The area of proposed MNI is smaller than [29-25-18-19]. The area of Xpipe [22] for Master and slave NI are obtained for 1 GHz frequency. The area of the same design at 666 MHz will be smaller than our NIs. The area of proposed SNI is bigger than [25-22]. The power results presented in table 4 show that the proposed NI has the most reduced value compared to other works. For exact comparisons, we suppose that all these designs run at 500 MHz and consequently our design will consume about 4 mw whereas that of [19] and [22] will consume about 10.43 mw and 16.75 mw. We can conclude that our work outperforms the presented other works in terms of power consumption. This is due to the fact that we use a low Leakage library which has a power constraint that affects the speed and area. The result of latency of the NI of [29] is between eight and ten 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 AETERAL 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. In this work, the latency in injection and extraction path is respectively 3 and 4 for MNI and 3 for SNI. These results show that our NI outperforms all other architectures in terms of latency in injection and extraction path for MNI and SNI.

7. CONCLUSIONS

In this paper, we have presented a new modular network interface architecture which offers high-level services with low latency and power. We have presented how we can apply the decoupling between computation and communications to achieve the IP modules and the interconnections to be designed independently from each other. The logic to manage the network services is independent from the type of the connected core protocols. We must only change the shell part and FIFOs memories parameters. When we change the flow control used by the network on chip, the kernel parts are changed and the other components are unchanged. To evaluate this approach, we have presented three implementations using respectively a 4 handshake phases, 2 handshake phases and a credit based control flow. The result shows that NIs implementations using credit based control flow present better implementations in terms of area, power, and latency than handshake versions. A comparative study has been conducted with other works. The obtained result shows that the proposed NI outperforms other works in term 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 and NoC parameters like (flow control, routing algorithm, flit width, queuing technique,..).

REFERENCES