A Stoppable clock based Approach for Low Power Network Interface Design in a Network on Chip

Brahim Attia[1], Wissem Chouchenne [1] , Abdelkrim Zitouni[1] , and Rached Tourki[1]

[1] Electronics and Microelectronics Laboratory, Monastir University, Faculty of Sciences, 5019, Tunisia

Abstract

A low-power design is an essential and important issue for portable or mobile systems. Network on chip (NoC) will become the main communication platform for this kind of Systems. To address the problem of an 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 Network Interface (NI) components. The architecture of NIs component must be modular to allow intellectual propriety (IP) module and interconnections to be designed independently from each other and its power must be kept as low as possible. In this paper, we present new modular NI architectures between IPs and router with low power constraints. The modular design is obtained through two separations between data flows and IP side and the network side. The low power is obtained by the implementation of a mechanism based on stoppable clock technique for power saving. The stoppable clock technique allows us to shut down each sub module when it is not running. Experimental results show that the Modularity and the stoppable clock technique aspects integrated in the proposed NI allow a significant reduction in terms of power between stoppable and baseline architectures while increasing at same time the area and decreasing the speed of NIs.

Keywords: *Network on Chip, Network interface, Low power, Low latency, stoppable clock.*

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 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, and synchronization are far more serious than ever. A network which delivers packets between communicating components has been proposed as a solution for SoC design. The network-on-chip (NoC) provides a high performance communication infrastructure. NoC is a new paradigm for integrating a large number of IPs cores to implement a SoC [4-5]. A router-based network is used for packet switched communication among on chip cores. NoCs are composed of routers, which transport the data from one node to another and the links between routers and Network Interfaces (NI) implement the interface to the IP modules. 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. This is a key ingredient in achieving the decoupling between computation and communication [7, 8], which allows IP modules and interconnects to be designed independently from each other. Many socket specifications exist to this end, such as OCP (Open Core Protocol) [9], VCI (Virtual component Interface) [10], AMBA AHB [11], and AMBA AXI (Advanced extensible Interface) [12]. Since most NoCs are message passing by nature, a 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 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 in terms of latency and throughput. There is a number of works published on the design of novel network architectures [13], but few publications have addressed particular

issues to the design of a NI module. Bhojwani and Mahapatra [14] 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. In [15] a NI ASIC implementing standard sockets was presented for the Athereal NoC. Seung [16] presents a generic architecture of network interface and associated wrappers for a networked processor array. In [17] a NI implementing VCI standard interface was presented for the SPIN NoC. In [18], an FPGA implementation of Network interface for an AHB standard was presented for mesh NoC. The NI however, has Low latency in forward and backward direction. In [19] 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 [20] an OCP compliant NIs for the mesh NoC was designed. These NIs have a low area and a low latency and they support only a burst precise mode outstanding read and write transaction. In [21] 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 [22] a NI design for Asynchrony NoC was presented. In [23] Network Interface Sharing Techniques is used for Area Optimized NoC Architectures. In [24], an FPGA implementation of a shared Network Interface architecture is proposed to reduce area and power by sharing the buffering resources. In [25], the authors present a low latency and power ASIC design of Modular network interface for network on chip with pipelined fashion.

A low-power design is an 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. The modularity of the architecture is another important issue to allow the IPs core and NOC to be designed indepdently from each other. In this paper we present a generic architecture model OCP compliant for low power network interface for the mesh 2D NoC. Our contributions include identifying key issues of NI design and developing an efficient and Modular NI architecture with low power constraint. We propose a new architecture of low power network interface that uses the stoppable clock technique. In our knowledge, it is the first time that the stoppable clock technique is used to reduce the power of NI. We evaluate the area, power and performance overheads of implementing NI tasks for NoCs that use credit based or handshake flow control with and without stoppable clock technique using different mode of OCP IP. The paper is organized as follows: Section 2 presents the related works. Section 3 gives an overview of NoC. Section 4 describes and details the two architectures of the proposed NIs. Section 5 presents the experimental results. Section 6 presents a comparison with other works. Finally in section 7 we conclude the paper.

2. 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. In order to interface the NoC with the tile we utilize a NI, which has the responsibility of packetizing and depacketizing the cores requests and responses. The NI has the responsibility of (i) receiving the contents from the IP core, 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 IP core. We have designed a NoC which is based on the mesh 2D topology. 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. The data flow through the network is a wormhole routing. The NoC uses credit based flow control strategies and we have adopted a determinist routing algorithm called source routing. The Source routing algorithm is executed to connect the input port data to the correct output port. In this routing, the header of packet opens the path between the source and destination units, while the successive data spread along the path and nodes. When the end-of-packet information is received, the packet path is closed and this frees the communication resource for following packets.

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 34^{th} bit encodes the beginning of-packet (BOP) and the 33^{rd} 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 header of request packet is composed by many fields such as:

Path to target: specifies the packet routing path from one source unit to a destination unit.
MCmd: presents the type of command (read, write).
MBlength: presents the burst length.
BSeq: presents the burst sequence (incremental, wrapping, and streaming).
MBprecise: presents the mode used by the IP (precise, imprecise,SRMD).
MBsingreq: indicates if the Single Request Multiple Data mode is used or not.
Destination address: defines the 'local address in slave IP.
Source address: defines the global address of the source router in NOC.

3. Proposed Network Interface

There are two fundamental separations in the NI architecture that enable this modularity: a horizontal one which distinguishes the injection path from the extraction path, and a vertical one which distinguishes between the network and the IP core. These two parts are referred to as shell and kernel, as proposed in the design of Phillips AEthereal NI [15]. The separation between injection and extraction functions allows an 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. 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 master Network Interface (MNI) for OCP IPs based cores for our network-on-chip, named Baseline MNI and Stoppable clock MNI both attached to a master IP. The two proposed MNIs are additionally split in two sub modules, one for the request and the other for the response data flow or channel (injection and extraction path).

OCP Protocol functions according to various modes. Among these modes, we note the burst precise (BP), Burst imprecise (BI) mode or SRMD. 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 it has. 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 a 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 requests from others cores.

In OCP there are three different burst models:

(i) Precise burst: in this model, the burst length is known when the burst is sent. Each data-word is transferred as a normal single transfer, where the address and command are given for each data-word, which has been written or read.

(ii) Imprecise burst: in this model, the burst-length can change within the transaction. The MBurstLength shows estimation on the remaining data-words that will be transferred. Each data-word is transferred as in the precise burst model, with the command and address sent for every data-word.

(iii) Single request multiple data burst: In this model, the command and address fields are only sent once. That is in the beginning of the transaction. This means that the destination core must be able to reconstruct the whole address sequence based on the first address and the MBurstSeq signal.

3.1 Baseline MNI architecture

The master network interface (MNI) transforms an OCP request to a request packet OCP/NoC and a response packet NoC/OCP to an OCP 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 IP cores. Figure 1 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 OCP 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 the 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 data flow is the response data-flow where the network is the source and the core is the destination.

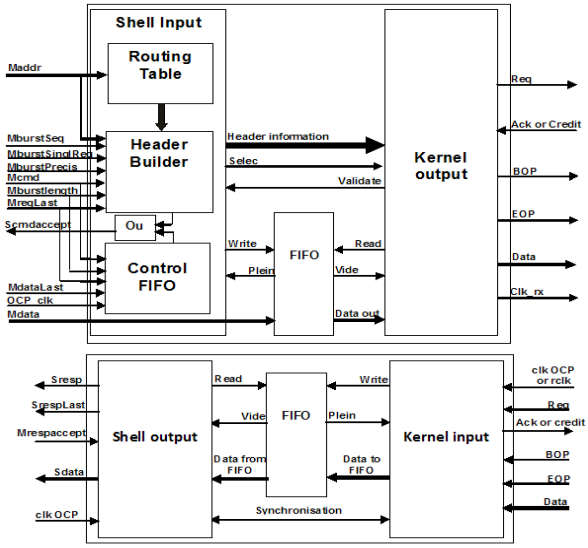

Fig.1 Baseline Master Network interface architecture.

The request data flow called also injection path performs the transformation of the OCP 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 OCP IP master.

1) Injection path

We split the design of injection path into the following parts: the shell input, the kernel output, and payload memory. In this part we will present all the modules that perform the services provided by the injection path to allow the transmission of the request packet flits to the network. We have designed for each mode of the burst, a specific implementation of header builder and control FIFO but we use the same implementation for the two types of flow control. For the request data flow using credit-based control flow, the kernel output implementation is the same for the three burst modes. Also for the request data flow using handshake control flow, the kernel output implementation is the same for the three burst modes.

Routing table: it is a local memory in the MNI. It stores the route paths to other slave cores in the NoC. This route path is needed as part of the packet header, since all packets are source-base routed. This means that all the routing information is stored in the path to target field which shows the routing nodes where to route the packet at each hop. The Routing table is not globally memory mapped and cannot be addressed by other cores. The table is configured and the entries are set at NI instantiation time.

Header builder: It takes in entry some essential OCP signals during the transfer and the field provided from a routing table which shows the path to the target. It encapsulates this information for building two header flits. If the command in the request is a write command, a payload should be added to the package. After the creation of the two header flits, the header builder module sends these flits to the kernel output Module using a simple module protocol.

Control FIFO: This module is responsible for the management of FIFO writing. It can also put an end or suspend the writing if it receives a high state on the signal full of FIFO. When a data is well written in the FIFO and FIFO is not full, then the controller is ready to accept any request, so it asserts SCmdAccept signal.

Or gate: This component takes in entry two signals. The first comes from the controls FIFO module (in the case of a write request) and the second is that of the header builder (in the case of a read request).

Payload memory: It is designed for the temporary storage of the data flits. The writing command in the payload memory is performed by the controller FIFO module. The reading commands from payload memory are performed by the kernel output.

Kernel output: It is the synchronizer between the NI and the network. It receives package flits from the Header builder or from the payload memory and sends it out from the NI to the network. Then it transmits the flits to the network using the four handshake phases or credit based flow control. It makes a packet transfer to the destination router. A network packet is composed of successive flits. A packet is always composed of header flits, which may be followed by one or more data flits. Whenever the header builder has two flits ready by the activation of *validate* signal, the kernel output module receives the header flits and informs the source router that the flit is ready on the data bus by asserting the *Req* signal, it puts the same signal *BOP* to a high level indicating the start of sending a new package with header as first flit. The module then waits for the

issuance acknowledgment signal (*Ack* or *Credit*) from the router to start sending data flits. Sending the last data flit (tail) will be joined by the set of *EOP* signal indicating the end of the transfer package.

2) Extraction path

The extraction path is divided into three stages. The first stage is where the data are received by the kernel input Module from the network via the NI. The second stage is the FIFO response where the data are temporary stored. The third stage is where the data are transmitted to the master core by the Shell output Module. Within the extraction path presented in Figure 2, several communications between modules proceed; the modules constituting this entity are described as follows:

Kernel input: it is the synchronizer between the NI and the network. It receives flits from the network using the four phase handshake or credit based protocol, and writes the response flits to a FIFO. At the time of the reception of the data on the bus Data, this module starts to make a temporary storage of these flits in the FIFO response to be read by the Shell output Module. The writing of the data is controlled by two signals *write* and *Full*. If the FIFO is full then this module does not assert the *Ack* signal from low to high for the router (*Credit* must be put at a low level for credit based). For the writing of the last data in the FIFO, the signal *Last Data* emitted towards the Shell output Module is put at a high level so that this latter knows the number of data remaining in the FIFO.

Shell output: its task is to transmit the response back to the master core. This module handles the response phase of the IP protocol. This module reads the data from FIFO, and then transmits it to the master IP. The Shell output module implementation is the same for the three burst modes and for the two implementations using handshake and credit based flow control.

3.2. Stoppable MNI architecture

The difference between baseline and stoppable architecture is the insertion of a stoppable clock module in the injection and the extraction path. In the injection path, the stoppable clock module can transfer or stop the local input clock to the two sub modules control FIFO and header builder as described in Figure 2. This module's role is to distinguish the type of command issued by the IP for a given transaction. Then, it allows the transfer of the input clock to the output clock (*clk header*, *clk controller FIFO*) respectively for the two sub modules header builder and control FIFO. Indeed, through OCP signal (*Mcmd*, *Mdatalast*) and signal generated by the Kernel output (*validate*) the formalism of local Stoppable clock is achieved:

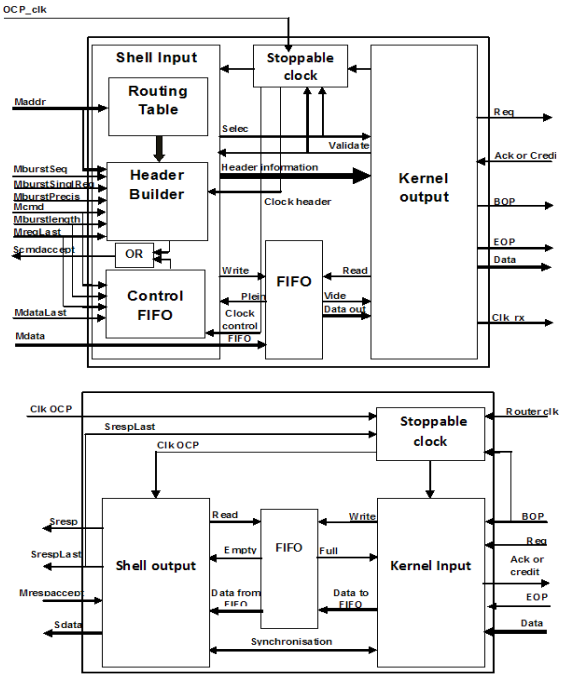

Fig. 2 stoppable clock Master Network interface architecture.

a) The *Mcmd* signal (Idle coded 000, Write coded 001, Read coded 010) can identify the type of request.
b) *Mdatalast* signal indicates whether the current write data transfer is the last in a burst.
c) *validate* is a signal that is provided by the kernel output to indicate the reception of the header by the local router port .

The sequences of phases of transfer or the stopping of the clock for a reading or writing operation are:

i) The first phase represents the beginning of transfer and the building of header flits for a read or write request. For a write transaction, the activation of control FIFO module is necessary to allow writing data in the FIFO.

ii) Once the header flits are received by the NI, *validate* will be activated until the two flits header will be transmitted to the local port of the router.

iii) Once done, a read transaction is completed by disabling the *validate* signal. In this phase, *Clk header* will be stopped for a read or a write transaction. On the other hand for a write transaction, control FIFO module continues its execution while *Mdatalast* is not asserted.

iiii) The desertion of OCP signal *Mdatalast* leads to the deactivation of *clk header* and clk *controller FIFO*.

In the extraction path, the Stoppable clock module detects the presence of response packet when *BOP*=1. It activates the kernel input module by transferring the clock of transmitter router if the used flow control is credit based. On the other side if the used flow control is handshake, it transfers the clock of the OCP IP. For the activation of the Shell output module, it transfers the clock of the OCP IP. (Handshake or credit based) when the presence of the first data placing in the FIFO. Upon detection the end of the transaction through *SrespLast* signal (*SrespLast*=1), it stops clocks of the Kernel input and Shell output modules if there is not a new response packet.

4. 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 different implementations for NI. On the IP side the three implementations use OCP IP protocol. The first implementation of NI uses a handshake 4 phases flow control and the second uses the credit based. Master and slave network interfaces with 32 bit OCP data fields and 32 bit network ports have been modeled with VHDL language on RTL level for baseline and stoppable MNI architectures. They were simulated and synthesized respectively by using the ModelSim tool and ISE tool from Xilinx. The synthesis result of the MNIs was done with FIFO data and FIFO response having a depth of 4 words of 32 bits. Each used FIFO has an adjustable depth and width. For master 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 two NI versions. Table 1, Table 2 and Table 3 show the area of baseline and stoppable MNIs using different flow control and OCP modes. The power consumption results are shown in Table 4. The maximum operating frequency obtained for these NIs implementations are shown in Table 5. The result of latency measurement by the simulation of MNIs is presented in Table 6. Table 7 shows the measurement of throughput obtained by the simulation of the two versions of the NIs.

4.1. Area of Network Interfaces

As 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 OCP data fields and 32 bit network ports using respectively credit based and handshake flow control. Tables 1, 2, and 3 present the area produced by the synthesis of baseline and stoppable MNI architectures for the three modes used by the OCP IPs using Handshake and Credit-Based flow control and showing the FPGA resources used. NSR presents the number of slice registers and NSLUTs present the number of slice LUTs. Table 1 presents MNI that uses the PB mode for Baseline and stoppable clock architecture using Handshake and Credit-Based flow control.

Table 1: BP area results and overhead

BP		Baseline	Stoppable	Overhead
Handshake	NSR	601	743	-23.6%
	NSLUTs	336	398	-18.4%
Credit Based	NSR	590	602	-2%
	NSLUTs	395	382	+3.3%

We show from these results that the resource used for Baseline MNI using handshake and Credit-Based are approximately equal. The resource used for stoppable architecture using Handshake mode is greater than Credit-Based architecture. Experimental results show that the overhead in terms of NSR and NSLUTs between Baseline and stoppable architecture is important between handshake implementations and poor between Credit-Based implementations. Table 2 and table 3 present MNI that uses the BI and SRMD modes for Baseline and stoppable clock architecture using Handshake and Credit-Based flow control.

Table 2: BI area results and overhead

BI		Baseline	Stoppable	Overhead
Handshake	NSR	692	661	+4.4%
	NSLUTs	399	378	+5.5%
Credit Based	NSR	697	707	-1.4%
	NSLUTs	432	416	+3.7%

The use of stoppable clock technique allows a little gain in area saving compared to Baseline architecture. This reduction is obtained because the

synchronization between the kernel and shell part is performed by the stoppable clock module which reduces the complexity of kernel and shell parts.

Table 3: SRMD area results and overhead

SRMD		Baseline	Stoppable	Overhead
Handshake	NSR	566	575	-1.6%
	NSLUTs	300	305	-1.6%
Credit-Based	NSR	587	596	-1.5%
	NSLUTs	338	327	+3.2%

We conclude that the use of the stoppable clock introduces an important overhead in BP mode and a little overhead in SRMD mode. For BI mode, the use of the stoppable clock architecture introduces a little gain in terms of area or resource used.

4.2. Power estimation of Network Interfaces

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

$$Pd = \alpha C_L V_{dd}^2 f \quad (1)$$

The first term denotes the dynamic power, α is the activity of the circuit, C_L is the parasitic capacitance, Vdd is the power supply and f is the operating frequency. By reducing Vdd one can drastically reduce the dynamic component, but unfortunately this is at the expense of speed degradation. The power consumption results are from ISE tool from Xilinx (XPower) and are based on an estimate where the clock frequency is set to 200MHz and the switching activity estimation is done by using vcd file simulation. The first exploration was performed for three NI implementations with 32 bits OCP data fields and 32 bits network ports using respectively credit based, handshake 4 phases with Baseline architecture. The second exploration was performed for three NI implementations with 32 bits OCP data fields and 32 bits network ports using respectively credit based, handshake 4 phases with stoppable clock based architecture. We display in table 4 the power estimation of baseline and stoppable MNI architectures for the three modes used by the OCP IPs using Handshake and Credit-Based flow control. When using the Handshake flow control for Base line and stoppable architectures, the BI mode is the lowest while the BP mode is the greatest. When using the Credit-Based flow control for Baseline architecture, the SRMD mode is the lowest while the BP mode is the greatest. When using the Credit-Based flow control for stoppable architecture, the SRMD mode is the lowest while the BP and BI mode is the greatest and are equal. The results show also that the power of handshake implementations is lower than the Credit-Based implementations for the modes of OCP IPs and for Baseline and stoppable architectures.

Table 4: power estimation of baseline and stoppable MNI architectures

Power (mW)		Baseline	Stoppage	Gain
BP	Handshake	35	26	25.7%
	Credit-Based	95	40	57.8%
BI	Handshake	12	7	41.6%
	Credit-Based	81	40	50.6%
SRMD	Handshake	25	11	56%
	Credit-Based	36	32	11.1%

Experimental results show that the power consumed by the stoppable architecture is lower than base line architecture. For the Handshake implementations, the gain obtained between the MNI Baseline architecture and the MNI stoppable architecture are 25%, 41%, and 56% respectively for BP, BI, and SRMD modes. For the Credit-Based implementations, the gain obtained between the MNI Baseline architecture and the MNI stoppable architecture are 57%, 50%, and 11% respectively for BP, BI, and SRMD modes. These results show that the use of the stoppable clock technique was benefic for designing a low power network interface. The use of gated clock technique reduces the activity of the MNI (α parameter) which reduces the dynamic power of this latter.

4.3. Speed of Network Interfaces

The speed results are obtained from the ISE tool from Xilinx and prototyped with Xilinx Virtex5 FPGA device XCVLX30. We present in table 5 the speed results of the Baseline and the stoppable clock based architectures for the three modes used by the OCP IPs using Handshake and Credit-Based control flow. We show that for any mode used by OCP IPs, the Handshake implementation is faster than Credit-Based implementation. This is true for Baseline and stoppable implementations. For Baseline or stoppable clock architectures, the BI implementation is faster than other implementations using Handshake or Credit-Based flow control. For Baseline and stoppable architectures, we see also that BP and SRMD speeds are the same.

Table 5: Speed results and degradation between baseline and stoppable MNI

Speed (MHz)		Baseline	Stoppage	degradation
BP	Handshake	444	372	16.2%
	Credit-Based	328	264	19.5%
BI	Handshake	459	407	11.3%
	Credit-Based	377	277	26.5%
SRMD	Handshake	444	372	16.2%
	Credit-Based	328	264	19.5%

For Baseline architecture, the maximum operating frequency obtained for these MNI implementations is about 459MHz in BI mode using the Handshake flow control. The stoppable clock architectures are slower than the Baseline architectures. For BP and SRMD modes, the speed degradation between Baseline and the stoppable clock architecture in Handshake and Credit-Based flow control are respectively 16% and 19%. For BI mode, the speed degradation between Baseline and the stoppable clock architecture in Handshake and Credit-Based flow control are respectively 11% and 26%. This study shows that the use of the stoppable clock technique reduces the maximum operating frequency of the Design.

4.4. 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 injection path when the request is presented at the OCP 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 OCP interface.

Table 6: Latency results

Latency (cycles)		BP	BI	SRMD
Handshake	Write request	3	3	3
	Read request	3	5	3
	Read response	7	7	7
Credit-Based	Write request	3	3	3
	Read request	3	5	3
	Read response	3	3	3

The MNI designs are tested and verified in two phases. In the first phase, the communication from the IP to the router was tested. In the second phase, the communication from the router to the IP was tested. The number of clocks to transfer a flit from an OCP IP to the router is calculated at different stages and the results are presented in table 6. 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)
FD: flit delay indicated in table 6.
M: time in cycle to forward a new flit.
N: packet length.
For 4ph handshake flow control M is equal to 4 and for credit based M is equal to 1.

For example, for write request of MNI with the packet length is equal to 8.

Packet Delay (cb) =3+1(8-1) clocks/packet
= 10 clocks/packet

4.5. 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. It depends on the Latency and the maximal clock frequency of each design. 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)
Throughput = 1 / FD (1 / Fmax))) (4)

Where FD presents the flit delay or latency indicated in table 6 and Fmax presents the maximal operating frequency.

Example: The flit throughput for Baseline MNI in forward direction using Handshake flow control and BI mode can be calculated as follows:

Throughput = 1 / (3*(1/(459*10^6)))
= 153 MFlits / Second
=4,896 Gbits / Second

Table 7 shows the throughput in forward and reverse direction with maximal clock frequency for Baseline MNI and stoppable MNI for the three modes used by the OCP IPs using Handshake and Credit-Based control flow. The experimental results show that the throughput of stoppable clock MNI is lower than the throughput of the Baseline MNI.

Table 7: minimal throughput results

Throughput(Gbit/s)		Direction	BP or SRMD	BI
Baseline	Handshake	Forward	4.736	4.896
		Reverse	2.029	2.098
	Credit Based	Forward	3.498	4.021
		Reverse	3.498	4.021
Stoppable	Handshake	Forward	3.968	4.341
		Reverse	1.7	1.860
	Credit Based	Forward	2.816	2.954
		Reverse	2.816	2.954

This is due mainly to the fact that the maximal speed of Baseline MNI is greater than Stoppable MNI and the latency or flit delay is the same for the two types of architectures.

5. Conclusion

In order to reduce power dissipation in a NoC, we have presented VLSI architecture of a new network interface. This architecture is based on a stoppable clock technique that allows shutting down each sub module when it is not running. The advantage of the proposed architecture relatively to the baseline architecture is that the power reduction is performed with the same latency and the speed degradation is between 11.3% and 16.2% using handshake and between 19.5% and 26.5% using credit based mode. The stoppable clock architecture allows 41.6% and 50.6% of gain in terms of power reduction for MNI respectively in Handshake and Credit-Based compared to baseline architectures.

References

[1] J. Liang, S. Swaminathan, and R. Tessier, "A SoC: a scalable, single-chip communication architecture", in international conference on Parallel architecture and compilation techniques, 2000, pp. 37-46.

[2] A. Mello, L. Tedesco, N. Calazans, F. Moraes, " Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC", in SBCCI, 2005, pp. 178-18.

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

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

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

[6] 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.

[7] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. S. Vincentelli, "Addressing the system-on-a-chip interconnect woes through communication-based design", In. DAC, 2001, pp. 667-672.

[8] Open Core Protocol Specification, Release 2.0, 2003, www.ocpip.org, OCP-IP Association.

[9] Virtual component interface standard - draft specification, v. 2.2.0, 1997, http://www.vsia.com; August 1997.

[10] ARM, AMBA AHB Protocol Specification, version 2.0,1999, www.arm.com, ARM.

[11] ARM, AMBA AXI Protocol Specification, version 1.0, 2004, www.arm.com, ARM.

[12] E. Salminen et al. "Survey of Network on Chip proposal". White paper, OCP IP, Mars 2008.

[13] P. Bhojwani, and R. Mahapatra, "Interfacing cores with on chip packet switched networks", In VLSID, 2003, pp. 382-387.

[14] 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.

[15] E. L. Seung, H.B. Jun , S. Y. Yoon , and N. Bagherzadeh, "A Generic Network Interface Architecture for a Networked Processor Array (NePA) ", in ARCS, 2008, pp. 247–260.

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

[17] 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.

[18] S. Stergiou, and al, "Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips", in DAC, 2005, pp. 559-564.

[19] 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.

[20] 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.

[21] T. Bjerregaard, S. Mahadevan, R. Olsen, and J. Sparso, "An OCP compliant network adapter for GALS-based SoC design using the MANGO network-on-chip", In ISSOC,2005, pp.171-174.

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

[23] 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.

[24] B. Attia, A. Zitouni, K. Torki and R. Tourki, "A low Latency and Power ASIC Design of Modular Network Interfaces for Network on Chip", in IJCSES, vol. 5, no 4, pp.257-270.

ah Att a received the MSc degree in analysis and processing of electronics systems from Faculty of Sciences of Tunisia, Tunisia in 2006. Since 2007 he is a Ph.D student in the Faculty of Sciences of Monastir. From 2007 to 2012 he has joined the Institut Supérieur des Sciences Appliquées et de Technologie de Sousse at Université of Sousse, as Assistant Professor in the department of Computer Science. He has published several papers in international scientific journals and conferences proceedings. He obtains the best paper award in 23th International conference in microelectronics. He is member of the Electronics and microelectronics Laboratory at FSM and is an associated researcher in Communication synthesis team. His researches interests include the network on chip design flow and automatic synthesis of Network interface for SoC based NoC with various constraints, Image and Video Compression, and low power design.

Wissem Chouchenne was born in Sousse, Tunisia on August 27 1984. He received the Master degree in Physics (Electronics option) from Faculty of Sciences of Monastir, Tunisia in 2011.His is a PhD student in ENIM Monastir, tunisia and lifl lille, France. His researches interests are communication synthesis for SoC and reconfigurable 3D Network on Chip.

A de t n was born in Gabès, Tunisia on October 06 1970. He received the D.E.A and the Ph.D. degree in Physics (Electronics option) from Faculty of Sciences of Monastir, Tunisia in 1996 and 2001 respectively. Since 2009 he has been recruited as Professor in Electronics and Microelectronics with the Physics department in the Faculty of Sciences of Monastir. His researches interests are communication synthesis for SoC and asynchronous system design.

a hed was born in Tunis, on May 13 1948. He received the B.S. degree in Physics (Electronics option) from Tunis University, in 1970; the M.S. and the Ph.D. in Electronics from Orsay Electronic Institute, Paris-south University in 1971 and 1973 respectively. From 1973 to 1974 he served as Microelectronics Engineer in Thomson-CSF. He received the Doctorat d'etat in Physics from Nice University in 1979. Since this date he has been Professor in Microelectronics and Microprocessors with the Physics department in the Faculty des of Sciences of Monastir. His researches interests are digital signal processing and Hardware–software codesign for rapid prototyping in telecommunications.