NOC: DESIGN AND IMPLEMENTATION OF HARDWARE NETWORK INTERFACE WITH IMPROVED COMMUNICATION RELIABILITY
RAJESH B V1, SHIVAPUTRA 2
Department of ECE, DR.AIT, Bangalore, INDIA
1raju2702@gmail.com, 2putrauvce@gmail.com

ABSTRACT
In Network-on-Chip (NoC), interconnection between IP cores is a subject of major concern, and is crucial factor for a successful product. In this paper, we present the design and Implementation of a Network Interface (NI) for Network on Chip (NoC) infrastructure with improved communication functionalities: store & forward transmission, bit flip detector and corrector, error management, power management, ordering handling, security, QoS management, end-to-end protocol interoperability, Error transaction handler. These improved Communication features can be designed on top of a basic NI core implementing data packetization and conversion of protocols, frequency and data size between the connected IP (Intellectual Property) core and the on chip network (NOC). The NI can be configured to reach the desired tradeoffs between supported services and circuit complexity.

Keywords: Network-on-Chip (NoC), Network-Interface (NI), VLSI architectures, Intellectual Property (IP), Multi Processor System-on-Chip (MPSoC).

1 INTRODUCTION
The next generation of Multiprocessor Systems-on-Chip (MPSoCs) will be composed of a combination of a high number of processing and storage elements interconnected by complex communication architectures. Processing elements will include general purpose processors and specialized cores, such as digital signal processors (DSPs), very long instruction word (VLIW) cores, programmable cores (FPGAs), application specific processors, analog front end, and peripheral devices [1]-[3].

The advance of the technology is raising the level of integration of intellectual property (IP) and scalability issue for communication architectures in very large-scale integration systems. Since traditional buses do not scale well in the system-on-chip (SoC) platforms, this trend has driven bus-based architecture toward networks-on-chip (NoCs). Networks-on-chip (NoC) is an emerging design paradigm for building scalable packet switched communication infrastructures, connecting hundreds of IP cells, in Multi-Processor System-on-Chip (MPSoC). NoCs provide a methodology for designing an interconnect architecture independently from the connected cores that can be general purpose processors, Application Specific Instruction set Processors (ASIP), Digital Signal Processors (DSP), memories or peripherals. NoCs will be a key component also for the success of future 3D SoC. A key element of a NoC is the Network Interface (NI) which allows IP macro cells to be connected to the on-chip communication backbone in a Plug-and-Play fashion [4], [5]. The NIs are the peripheral building blocks of the NoC. Basically, Network Interface are in charge of traffic packetization/depacketization to/from the NoC.

Fig. 1 An example SoC that has a 2-D mesh NoC with 9 resources. Network interface is denoted with NI.
For illustrative purposes, Fig. 1 shows an example SoC with a NoC and nine heterogeneous IP blocks that are CPUs, memories, input/output devices, and HW accelerator [10]. A NoC packet includes a header and a data payload which is physically split in units called flits. All flits of a packet are routed through the same path across the network. The header field is composed of both a Network Layer Header (NLH), whose content is determined by the NI according to the node map network configuration, and a Transport Layer Header (TLH) containing information used by the NIs for end-to-end transaction management. Some NI designs proposed in
literature also implement the conversion of data size, frequency and protocol between the original IP bus and the NoC. In recent literature some NIs have been presented that add to the basic IP-NoC interface functionalities some features such as handling of out-of-order transactions [6],[7], detection of error transactions[8], secure memory access control[9]. However, the literature does not present a design integrating all the above mentioned advanced features in the same NI. With respect to the above NI functionalities other services may be useful to support in hardware such as end-to-end interoperability, management of pending transactions when powering down/up the IP to increase energy efficiency, remapping for master IP cores. To overcome these limitation, here we presents the design and Implementation of a Network Interface (NI) for on-chip communication infrastructure with improved networking functionalities: store & forward transmission, bit flip detector and corrector, error management, power management, ordering handling, security, QoS management, end-to-end protocol interoperability, Error transaction handler. The NI can be configured to reach the desired trade-off between supported services and circuit complexity.

2. FUNDAMENTAL NETWORK INTERFACE DESIGN

IP cores in a NoC infrastructure are commonly classified into Master and Slave IPs: master IP generates the request transactions and receives responses, where in slave IP receives and elaborates the requests and then sends back proper responses to the master IP.

NI connected to the master IP is known as Initiator NI, similarly Target NIs is connected to slave IPs. Initiator NIs performs two processes firstly convert IP request transactions into NoC traffic, and also translate the packets received from NoC into IP response transactions. Target NIs present a mirrored architecture: requests are decoded from NoC; responses are encoded. Fig. 2 shows the complete overview of NoC platform. Conversion features must be implemented in the two directions, called request path (from Master to Slave IPs, blue paths in Fig. 2) and response path (from Slave to Master IPs, red paths in Fig. 2) respectively. Fig. 3 indicates a more detailed view of the core NI Initiator architecture with a clear distinction between request and response paths. The NoC interface presents an Upstream (US) section, to send packets to the interconnect and a Downstream (DS) section, receiving packets from the NoC. Before passing data on to the Kernel, the Shell also builds the Network and Transport Layer headers, needed by subsequent NoC components (i.e., routers and target NIs) for forwarding the packet and decoding it at destination. The NI Kernel part manages buffering and other services. FIFO interface technique is used to kernel to the shell. Shell module encodes the incoming header data; these data are stored in two kernel FIFO’s, an header FIFO (holding transport layer and network layer headers) and a payload FIFO (holding bus raw data). Each FIFO has its own read and write managers which update FIFO pointers and status, and provides frequency conversion mechanisms. The Kernel is connected to the NoC interface stage through two additional FSMs. In the request path, an output FSM (OFSM) reads headers and payloads and converts them into packets according to the NoC protocol. In the response path, an input FSM (IFSM) collects packets and splits header and payload fits into their respective FIFOs. To be noted that the NI encodes both the TLH and the NLH, while in the decoding action only the TLH is taken into account because the packet has reached its destination and routing data are not needed.
Fig. 4. Upsize conversion

Data size conversion between IP and NoC domain is also possible, since read (RD) and write (WR) managers can access a FIFO. The conversion is managed by exploiting FIFO rows and columns concepts. A FIFO column is sized according to the larger data size between data in and data out; a FIFO row is sized according to the smaller data size between data in and data out. Up-size conversion is accomplished by writing by rows and reading by columns; down-size conversion is exactly the opposite. Fig 4 shows design of Up-size conversion. For example, consider large amount of payload data generated by an IP with data bus size of 32 bits and connected to a NoC with flit size of 128 bits: four 32-bit data write accesses by the IP are necessary to fill a 128-bit payload FIFO location and make it available to the NoC to read it. 32 bit payload is stored in 4 rows by incrementing the row pointer at a specific column with the help of the write manager. Finally up conversion is obtained by reading the entire column with the help of the read manager, which selects specific column.

3. ADVANCED INTERFACE FEATURES

These sections describe the configurable services available as special features in the NI design.

3.1 Store & Forward (S&F)

Both Request and Response directions FIFOs contain flits, either encoded from a bus transaction and to be transmitted over the interconnect, or received from the NoC. Default NI behaviour is that a flit is extracted from the FIFO as soon as it is available i.e., read pointer and write pointer are next each other. Hence, if the original traffic at a NoC or bus interface has an irregular nature, such a shape is reflected also into the other bus or NoC interface. When Store & Forward is enabled, flits are kept into the internal kernel FIFOs until a) the previous whole packet is encoded and transmitted (request path), b) the previous pay load and header flits are received successfully by master IP. This way, an irregular traffic is changed to a bubble-free traffic thus improving overall system performance.

3.2 Bit flip detector and corrector

As we know that the FIFO is also a small storage unit, which preserves the information for retrieval whenever required. Due to increase in soft error rate in logic circuits such as encoder, decoder & in FIFO blocks, which reduces the reliability & overall NI performance. Therefore here we include a Bit flip detector and corrector. Bit flip corrector at head portion of NI, which is capable of detecting and correcting of single bit flip, similarly bit flip detector at payload portion of NI, which is capable of only detecting these is because usually size payload will be much larger than header, correcting module for these increases the size of NI. There after Error payload packet is flagged in its header as an error packet, later EMU builds a response packet to master IP requesting for proper payload.

3.3 Error Management Unit (EMU).

The EMU can be placed between kernel and the NoC interface. The EMU behaviour is different in Initiator and Target NIs. In an Initiator NI, EMU can handle bad address errors or security violations. When the address of the Master IP transaction is not in the range of the assigned memory map, or when the transaction is trying to access a protected memory zone without having the rights, the packet is flagged in its header as an error packet [8]. The EMU then filters the packet directed to NoC US interface to avoid it to enter the network, and builds a response packet re-mapping the request header on a new response header. The response packet is sent back to the Master IP in order to be compliant with protocol rules. The EMU of a Target NI When the Power Manager is enabled, the EMU is also in charge of properly managing incoming traffic at DS NoC interface during power down mode. All the traffic received in request during power down mode is flushed by the EMU, so that it never reaches the Slave IP. The EMU itself generates an error response to the Master originating the request. The EMU is composed by 3 blocks as in Fig. 5 showing EMU and Power Manager Blocks in a Target NI:

i Error Detector, in Target NIs all incoming packets are flushed if the connected Slave IP is in power down mode;

ii Error Encoder, which assembles a new NoC packet to be channelled in the response path;
iii Error Write Manager, avoids simultaneous traffic to the US NoC Interface from Kernel Response and from the EMU in a Target NI, while in an Initiator NI it avoids interference between the DS NoC Interface and the EMU both trying to access the Kernel Response.

Fig. 5 EMU and Power Manager Blocks in a Target NI

3.4 Power Manager (PM)
This feature is available only for Target NIs connected to Slaves which may be turned off to save power. A simple req/ack protocol controls the power up/down state of the NI, by means of a dedicated interface: each request (req set to 1) acknowledged by the PM unit (ack set to 1) makes the NI power state switch from UP to DOWN and vice versa. It may happen that a request for power down is sent to the PM while the Slave IP is still elaborating a number of pending transactions. In this case the Target NI stops accepting packets from the network and waits for all pending transactions to be processed (see the counter of outstanding transactions in Fig. 6) before acknowledging the request and switching to power down mode.

3.5 Security
The security service can be implemented in NI Initiators as a hardware firewall mechanism (see Fig. 7), it introduces a security rules Master IP transactions must satisfy to gain access to the network. Security rules are applied during packet encoding [9]. If a test fails the security check, the corresponding transaction is marked as an error in the NLH and it is detected by the EMU. The illegal packet is then discarded and does not consume network bandwidth, and the error response to the Master IP is directly generated by the EMU itself. These rules involve listing of memory zone under access control, listing of Master IPs that may have access to a certain memory region and listing of access types (i.e., Read, Write and Execution permissions, RWX) for a certain IP on a certain memory region.

Fig 6 shows the operation of the protected memory map, in this map a number of memory regions are defined, and associated to Region IDs (Protected Memory Zones in Fig. 5). The same map also defines how these regions can be accessed. Access to these zones can be allowed only to Master IPs belonging to specified Groups. Finally, a table of read/write/execution permissions is given for each group and for each protected region. The implemented security mechanism supports up to 8 protected memory regions, up to 8 group access rules with Read/Write/Execution permissions, up to 16 Source to classify Master IP.
3.6 Ordering Handler

Each request transaction generated by a Master IP is consisting of a destination address and an identification number. The destination address identifies a specific Slave to access. The identification number characterizes the Master itself: this information is used by Target NIs to encode response packets to be routed back [6]. Typically, bus protocol rules impose that transactions generated by a single Master IP get their responses with the same order of the requests. Once a transaction is forwarded from master IP to the network nothing can be said about the response time of slave. In general, each Slave has its own list of requests (from several Masters) to respond to, and it may happen that a Slave receiving request n from a Master is slower to reply than the Slave receiving request n+1 from the same Master, due to its longer requests list to handle. As a consequence, there is no guarantee that responses will get back to the corresponding Initiator NIs in the correct order. This is a problem when transactions with the same ID (same Master) but different destinations (different Slaves) hang around the network.

When an Initiator NI receives the response transactions it cannot distinguish from which specific Slave it comes because the ID field is the same (and, generally, the address information is not available in the response path) [7]. To avoid the risks of this situation the proposed NI may be configured to support the Ordering Handler feature which is placed in the Initiator NI Shell (see Fig. 7). A Master IP can access a generic number of Slave IPs via NoC but where ordering filter just filters out the transactions with the same master ID which are directed to different destinations from accessing the network simultaneously. But the transactions with the same master ID are accepted only if the intended destination is the same of still pending transactions. This mechanism uses a buffer to store the history of the Master pending transactions. A single buffer entry is represented in Fig. 8. The entry is allocated upon reception of a request transaction with a new ID/destination pair. Any new transaction with the same pair increments the outstanding transactions counter in the associated entry (ISSCAP field in Fig. 8, whose size is configurable). The counter is decremented upon delivery to the Master of a corresponding response packet (characterized by the same ID). When the counter is zero, there are no more pending transactions, and the entry is again available for other ID/destination pairs.

3.7 QoS Scheme

Since NOC can interoperate in different traffic classes, a QoS is necessary to avoid interference among them. In addition, in an interconnect NoC there is a need for an easy real time reconfigurability of slave bandwidth allocation. The proposed solution the FBA (Fair Bandwidth Allocation) scheme for real-time bandwidth allocation. The basic principle of FBA arbitration is to share the Slave available bandwidth among the Masters during peak request period and to apply a faction tag at packet injection (i.e. in the NI), and to keep together, in the same faction round. This scheme should not be confused with the TDMA approach: the faction round duration is variable; if only packets belonging to a new tag are received, they win the arbitration, so that there is not wasted bandwidth. For example, while in Faction round i in all IPs (1,2,3,4) are using their reserved bandwidth, in Faction round i+1 IP1 is not producing traffic and the total bandwidth is redistributed among the other IPs(2,3,4). The NI tags the packets with a faction Identifier (if needed with their priority) and each injected flow specifies the requested bandwidth. The requested bandwidth corresponds to a threshold that must be reached by a counter to switch the faction identifier bit. The counter (inside an Initiator NI shell) computes (from the opcode) the number of bytes that flow to each target and enables the faction bit switching when the threshold is reached. The proposed FBA scheme offers a benefit where we need to program only the NIs with the requested bandwidth value, by means of a tagging mechanism based on a simple counter.
3.8 Interoperability and End-to-end Size Conversion

Fig. 10. (a) Reshuffling in the Byte Lanes Matrix and (b) Byte Lane Matrix coupled to the Keep/Pass logic.

The proposed NI are capable to handle the protocol, size and frequency conversion not only at IP-to-NoC level (and vice versa), but also at end-to-end level, obviously only for the supported IP protocols. This way, from an end-to-end point of view, the NI performs the protocol, frequency and data size conversion between Master and Slave IPs. Each NI collects IP traffic from the core it is connected to and then converts such traffic into NoC packets sending them to the network of routers; upon arrival at the destination NI, the NoC packets are translated into the correct IP transactions according to protocol, frequency and bus size of the destination IP. This is achieved by enabling Interoperability and End-to-end Size Conversion support if the Master and Slaves do not use the same bus protocol. With no restriction on opcode but the guarantee of addresses aligned to the Slave data bus size it is possible to enable a simplified end-to-end size conversion hardware based on a Byte Lane Matrix for reshuffling correctly the 32-bit pieces of payload in the transfer, depending on address and opcode. Fig. 10(a). In other cases when the limitations cannot be applied, a specific support may be required for address realignment coupled to payload cells reshuffling through specific Byte Lane Matrix and Keep/Pass logic (Fig. 10 (b)): while the Byte Lane Matrix changes the Byte Lane position within the same transfer (vertical reshuffling), the Keep/Pass logic changes the Byte Lane position between two transfers (horizontal reshuffling), which might be needed for some wrap operations.

3.9 Error transaction handler

Retransmission of master IP transaction is required due to increase in soft error rate in logic circuits, bad address errors or security violations. In all above situation EMU builds a response packet on a new response header and sends to Error transaction handler. Fig. 11. Shows placement of Error transaction handler in a NI Initiator. Error transaction handler, requests the master IP for Retransmission (in order to be compliant with protocol rules or soft error rate), meanwhile it holds target (slave) ID and TLH header in buffer entry along with the Configurable Retransmission Timer/Counter. Fig 11. Error transaction handler in a NI Initiator

This mechanism also uses a buffer to store the history of the error transactions. A single buffer entry is represented in Fig. 12. If the retransmission of the master IP transaction is successfully done, then the corresponding buffer entry will be cleared or else Configurable Retransmission Timer/Counter starts decrementing. When the timer expires (from the predefined value), Error transaction handler once again requests the master IP for Retransmission.
4. CMOS IMPLEMENTATION RESULTS

The proposed NI is designed to support a wide configuration space, see Table II. By changing the configuration set different trade-offs between performance and complexity are achieved. Figure 13 shows simulation result of implemented NI.

![Fig. 13 Simulation Results](image)

CONCLUSION

In this paper, we presented network interface (NI) design for on chip communication. The proposed Network interface supports a wide set of improved communication functionalities: store & forward transmission, bit flip detector and corrector, error management, power management, ordering handling, security, QoS management and interoperability. Error transaction handler. The capability to support all these features in the single hardware represents a novelty with respect to the state of the art. Furthermore, we can change the Network interface (NI) configuration to reach the desired trade-off between supported services and circuit complexity. The proposed NI represents a complete solution that can be adopted for different scenarios, from multimedia to real-time applications.

ACKNOWLEDGMENT

I express my special thanks to the Dr. G.V.Jayaramaiah, Dr. J.S. baligar and Mr. Chetan for providing opportunity and valuable guidance. I would like to take this opportunity to thank my parents for this support and patience. I am also thanking to my friends for their continued moral and material support throughout the work.

REFERENCES