Networks on Chip, router architectures and performance challenges

Manel Langar  
Department of Electrical Engineering, National Engineering School of Tunis, Tunisia.

Riadh Bourguiba  
Department of Electrical Engineering, National Engineering School of Tunis, Tunisia.

Jaouhar Mouine  
Department of Electrical Engineering, Prince Sattam Bin Abdulaziz University, Saudi Arabia.

Abstract

With the continuous technology scaling, System On Chips (SoCs) have evolved considerably and can integrate an important number of Intellectual Property (IP) cores in the same chip. However, global interconnects are become the main performance limitation of SoCs. The Network on Chip (NoC) paradigm has emerged as an efficient interconnection structure addressing the global wire delay problem. This solution outperforms traditional on chip interconnects, such as Point to Point (P2P) links, bus and bridges. Since the introduction of the NoC paradigm in the last decade, several methodologies have been presented by researchers to enhance its performances in terms of latency, area occupancy and power consumption. In this paper we described the interconnect evolution from point to point links to network on chip paradigm. Then, we focused on the NoC concept and presented some of their principal characteristics. Finally, we proposed a new switch architecture enabling an adaptive inter-port buffers sharing. The proposed design optimizes the virtual channels exploitation which results in an improvement of the NoC zero load latency and throughput without inducing neither area nor power consumption overhead.

Keywords: Network-on-Chip (NoC), System-on-Chip (SoC), switch router, Multi-processors SoC (MPSoC), Buffer sharing, Mesh2D.

Introduction

Since late nineties, embedded systems have entered a new era, where a whole system can be integrate into a single chip. These so called, Systems on Chip, or SoCs, combine software flexibility and hardware computation power to provide real-time performance. Usually, they embarks several processors and coprocessors, with all the memories, peripherals and interfaces required to implement any application chosen in a given domain (video, cell phone, etc.). Two key aspects must be carefully taken in account when designing such devices: software/hardware partitioning and computational power. However, they both would be useless if not supported by a strong communication scheme. In fact, this latter is even of highest importance, since an inadequate communication architecture can considerably decrease the final system performance. To cope with the increased need of communication bandwidth, integrated systems have evolved from simple point-to-point (P2P) links, to crossbars, then buses and finally Networks-on-Chips (NoCs). Submitted to very strong timing constraints, NoCs have been optimized in a way, very specific to SoCs, where time is counted in tens nanoseconds. In this paper, we expose these optimization techniques and we propose a new one, based on buffers sharing.

In the next section, we start by retracing the communication techniques evolution in the SoC domain during the past decades. In the third section, we present the NoC and describe its characteristics. Then, in the fourth section we gradually present the optimizations elaborated to reduce latency and increase throughput of NoCs, before exposing our buffers sharing technique and its experimental results in the fifth section. Finally, the last section concludes this paper.

Communication in SoCs:

The communication function is very important in any embedded system: it can help express the computation power of the whole architecture if well designed, as it can ruin all the efforts if badly conceived. In the context of SoCs, it becomes of highest importance, since you have a very limited visibility on the internal functioning of the system, you can not modify it once it was fabricated, and any correction will require several months and will cost hundreds thousand dollars...

IP block

During the 90’s, with the wide adoption of Hardware Description Languages (HDLs), such as VHDL or Verilog, as the standard way to conceive new digital circuits, a new concept has emerged: Intellectual Property blocs (IP blocs). Based on the fact that hardware can be as reusable as software, and encouraged by design reuse methodologies, this led to the idea that a system should not be built from scratch, but rather built from already verified and proven components. Since then, all parts of a system are not seen as any scattered components anymore, but as highly valuable and reusable IP blocs, or shortly expressed: IPs. So designing a new SoCs mainly signifies choosing among existing IPs, those that fit the needs of the targeted application, and connect them properly.
Only non-existing functionalities are specifically designed, then deeply verified and documented, before being cataloged for future designs.

To foster IPs reuse, secure investments and improve the overall quality, several standard have been specified that precisely define IP interfaces (connectivity and protocol). These later considerably facilitate IPs exchange between design projects and companies.

**P2P link**

In early devices, communication was quite simple: a producer synchronizes with a consumer using a simple handshake, in order to transmit data. Today, IPs still implement the same principles, but adapt them to the SoC flavour. Interfaces can be master or slave, depending on who order data exchange. Also, signals have specific meanings and can be optional, depending on the kind of IP (for example, a processor can eventually have cache management signals in its interface). Given the complexity of real components, often IPs have multiple interfaces dedicated to different usages. Thus, a Direct Memory Access (DMA) coprocessor can have both a slave interface to receive instruction from the main processor, and a master interface to read/write data from/to memory.

**Crossbar**

P2P links are not adapted to resource sharing. For example, if two processors share two memories and a serial communication controller, P2P links would be of poor help, while a crossbar would perfectly fit the need. This latter provides simultaneous access to different shared slave IPs, together with arbitration between master IPs when conflict accesses attempts occur.

Unfortunately, their ease of use and their great bandwidth is paid a high price, as it implies complex routing schemes, that are not scalable with an increasing number of masters and slaves in the system.

**Shared Bus**

In order to reduce the routing complexity of SoCs, while still maintaining resource sharing, system architects have integrated communication buses. With this kind of architecture, IPs are connected to the bus through their master and/or slave interface(s), if natively compatible with the bus protocol.

Otherwise, an adapter will bridge the gap between between the IP’s interface protocol and the bus protocol. Again, to improve design reuse and interoperability, standard protocols have been specified. Most of them suppose two bus architectures: one dedicated to high performance IPs (processors, co-processors, SRAM, DRAM controller,...) and another one to slow peripherals (input/output controllers, general purpose I/O, sensors interfaces,...). Both have an address decoder to select a slave, but usually only the first one has an arbitration unit to take a decision when several masters try to access the bus simultaneously. A SoC can embed as many buses as required from each type, that can be connected with bridges.

A bus transfer passes by four phases:

1. Bus request: the master behind the transfer notifies to the arbiter that it needs to access the bus.

2. Arbitration: the arbiter analyzes the requests of all the masters connected to the bus and designates a winner, depending on a specific arbitration policy (fixed priority, round robin, time division multiple access,...).

3. Address decoding: the winning master places an address on the bus for the decoder. This latter decodes the high order bits of the address to select a slave, which in turn decodes the low order bits of the address to select an internal storage location and prepares itself for responding.

4. Data exchange: Depending on the transfer direction (read or write), data is either placed on the bus by the slave and read by the master, or placed by the master and written in the slave.

In practice, these four phases do not take place sequentially in the same clock cycle, but are rather distributed on four consecutive cycles, in a pipelined fashion. With this technique, overall latency stays the same, but the throughput is almost multiplied by four, since four data transfers can overlay at a single cycle. Pipelined transfers are supported by a panel of burst transfers that avoids pipeline stalls during re-arbitration.

**Network on Chip**

In 2002, Luca Benini and Giovanni De Micheli introduced a new interconnect structure: the Network on Chip (NoC) [1]. NoCs outperform bus based interconnects by allowing a high number of IPs to communicate simultaneously. They provide higher bandwidth, higher scalability and better modularity. Packet switched NoCs are composed of three main components:

- Network Interfaces (NI): adapt NoC protocol to that of IP cores and vice versa.
- Switches (routers): route messages from source nodes to destination nodes according to a predefined routing algorithm.
- Links: physical links that connect switches and NIs.

Figure 1 below illustrates an example of a 2D mesh NoC.
NoC characteristics:
NoCs are characterized by several design parameters: topology, routing algorithm, flow control and switch architecture.

Topology
The topology describes the way in which switches, network interfaces and links are organized. There are several topologies for NOC architecture like mesh, torus, ring, butterfly, octagon, fat tree, heterogeneous [4]. The NoC topology impacts its implementation cost. If the number of connections per switch is high, the total bandwidth of the system improves, but results in an overhead in terms of implementation cost. Then, not all network topologies are suited for a silicon implementation of a NoC. Mesh topology is one of the most used topologies due to its regular structure and its simple architecture (Figure 1).

Routing algorithm
In packet switching NoCs, the routing algorithm describes how a packet is routed to its destination node. There are two main types of routing algorithms:
- Deterministic routing: all the packets of a source-destination couple will traverse the same path.
- Adaptive routing: packets of a source-destination couple can traverse different paths. The routing path can be modified depending on a metric such as the network congestion.

Adaptive algorithms can reduce congestion situations and enhances the reliability when compared to deterministic ones. However, when using adaptive algorithms, the reception order of packets can be different to that by which there were sent. So reordering buffers are needed to reorder the received packets and their area is important.

During the routing task, some failure situations can occur:
- Deadlock: a packet is waiting for an event that can not happen.
- Livelock: a packet never arrives at its destination and remain indefinitely in the NoC.

The XY routing algorithm is one of the most known deterministic deadlock-free algorithms used by 2D mesh topology. It routes packets in the horizontal direction, then in vertical.

Switching techniques
The Switching technique describes how the packets are transmitted between switches from source to destination nodes.

There are three main switching techniques:
- Store-and-forward: packets are transferred to the next switch only if the whole packet has been received [19]. So switches must have enough buffer resources to receive the longest packet circulating in the NoC. Also, packets latency depends on their length.
- Virtual cut-through: It is similar to the store-and-forward but it allows the packet to be transferred to the next switch once its header is received. This technique reduces the packet latency and the required buffering resources [10].
- Wormhole: It is similar to the virtual cut-through but the packet is slit into sub-packets called flits (Flow Control Unit). This technique has the best performances in terms of packet latency and required buffer resources [4].

Flow Control
The flow control manages network resources, such as channel bandwidth, buffer slots, which are allocated to a packet circulating in the NoC. The flow control can be either buffered or bufferless. The Bufferless Flow Control is not widely used because it has more latency and fewer throughput when compared to the Buffered Flow Control. There are many Buffered Flow Control such as Credit Based [3], ACK/NACK[2] and Handshaking Signal based Flow Control [22].

Implementation Techniques
Implementing a NoC implies constraints that are very specific to the microelectronic context. In this domain, time and latency are measured in nanosecond, which prevents using sophisticated algorithms for routing or resource allocation. Contrariwise, performance can be found by promoting simplicity and regularity, which in turn allows reducing of silicon area and increasing the throughput, thanks to pipelining.

Basic architecture
Each flit arriving at an input port of the switch is buffered, then routed, before being authorized to pass the crossbar to its destination. So a basic implementation of a NoC switch must inevitably provide a FIFO buffer for each port, plus a router, a crossbar and an arbiter for managing accesses conflict to the same output port (Figure 2). As we use the wormhole flow control, the FIFO buffer can be quite smaller than the packet length.

Usually, the FIFO depth is four to eight flits and it stores data synchronously with the clock rising-edge. The router must implement a strategy that can give a response within a clock cycle. Nonetheless, routing is made only for the header flit, as consecutive flits of the same packet follow the same way.
Pipeline
With the previous architecture, in most favorable conditions, a flit has a single cycle to be routed, to get access to the output port, to pass the crossbar and to cross the link to the next switch where it will be written. Since the delay for all these operations is quite long, it is better to detach them into separate operations that would be performed in different clock cycles, at an increased clock frequency, in a pipelined manner. Typically, the pipeline is made of three stages: one for routing and switch allocation, another for bridging the crossbar, and the last one for passing the link to the input buffer of the next switch (Figure 3). As for the pipelined buses, this technique leaves the latency untouched, but multiplies the maximum theoretical throughput by three.

Look-ahead routing
With Virtual channels, the execution time of the second stage of the pipeline has been extended with the delay necessary to allocate a virtual channel, which results in unbalanced pipeline stages and a decreased throughput. To solve this problem, the solution is to anticipate the routing of the packet at the previous switch. By doing so, a packet is already routed when it arrives at the input port. Thus virtual channel allocation, followed by switch allocation, can take place immediately after extracting the flit from the FIFO buffer (Figure 5). Of course, routing for the next switch still has to be done, but since it depends only on the destination node and the position of the current node in the NoC, it can be done in parallel. However, this restricts the choice for the routing algorithm.

Virtual channels
With wormhole switching, a blocked packet can block all the switches over which it is spanned, resulting in severe performance decrease. To reduce this risk, one solution consists in providing a bypass to get around the blocked packet. For this purpose, few FIFO buffers are added to the previous single one, in each input port. Thus each FIFO buffer, or so called virtual channel, will play the role of a highway lane, that would let the fast packets to exceed slow ones (Figure 4).

However, virtual channel raises a new problem: virtual channel management, or more precisely virtual channel allocation. This is done by a new hardware unit, that takes place in the second stage of the pipeline, after routing and before switch allocation.

Speculative switch allocation
Another technique used to reduce the execution time of the second pipeline stage of a VC switch speculates about the results of the virtual channel allocation. This gives the opportunity to the header flit of a new packet to try getting access to an output port, before having effectively acquired an output VC [17]. Hence, virtual channel allocation and switch allocation can be done in parallel during a single clock cycle (Figure 6).

Intra-port buffer sharing
The memory used for FIFO buffers can represent about 40% of the total switch area. So two ways have been explored to reduce it, based on memory sharing. One [16] reconfigures FIFO buffer size, while the other [7] proposes dynamically shared buffers. The saved silicon area can be regarded as an
economy, or can be re-invested to increase the number of virtual channels, and enhance the switch throughput at limited costs.

**Inter-ports buffer sharing**

In this paper, we propose another technique to reduce the memory area, based on sharing unused queues between neighboring input ports (Figure 7). So, when an input port doesn’t use a FIFO queue, this latter can be lent to another input port. It is particularly beneficial, because in a real NoC, the local data traffic is never isotropic, but rather oriented in a specific direction, that evolves with time. Inter-ports buffer sharing lets the switch re-organize its input FIFO buffers to absorb the data traffic, as the sail of a ship that would rotate to capture the wind.

![Figure 7: Example of inter-ports buffer sharing](image)

**Experimental Results**

To evaluate performance of the proposed inter-port buffers sharing NoC switch and the typical speculative VC switch [17], we developed two cycle-accurate simulators. We considered a NoC with a regular 2D mesh topology using the XY deterministic routing algorithm. Both switches implement the virtual channel wormhole flow control with credit based protocol. They have five bi-directional ports: four ports (east, west, north and south) connected to neighbor switches and a local port connected to an IP via the network interface (Figure 1). Each input port has 2 VCs with 4 flit buffers per VC.

The packet is split in four 32 bits flits. Simulation of each switch runs firstly for 10,000 warm-up cycles. Then, 100,000 packets are injected to the NoC and the simulation progresses till the reception of all these packets.

Latency of a packet is measured from the time its head flit is created by the source IP core, to the time its tail flit is consumed by the destination. Average network latency is the mean of all packet latencies in the network.

Experiments are performed over uniform random traffic pattern across a 8-by-8 mesh network. Figure 8 bellow represents the latency-throughput curves of the proposed switch and VC2 switch over uniform random traffic.

As shown by the curves, our proposed switch improves the saturation throughput (0.29 flits/cycle) when compared to the VC2 switch (0.28 flits/cycle). Also, it reduces the zero-load latency by 3 cycles (33 cycles versus 36 cycles).

As for the power and area consumption, the two switches have very similar architectures with the same buffers size, so their performances are very close.

The two routers were written in VHDL code and simulated using Mentor Graphics Modelsim. We implemented them using TSMC 65nm technology. The synthesis of the VHDL code is done using Cadence RTL Compiler. We constrained the designs and verified that they work properly at 500MHz target frequency.

**Conclusion**

Progress in silicon technology has enabled the integration of a whole electronic system into a single chip. Systems on chip combine software and hardware to provide both adaptability and power computation for demanding embedded applications. In this paper, we started by relating the on chip interconnect evolution. Then we introduced the network on chip paradigm and described its main characteristics and design parameters. Finally, we proposed a new switch architecture allowing adaptive virtual channel buffers sharing among different input ports, in order to improve buffer usage of switches in 2D Mesh NoCs. A simulation platform was developed and experiments over uniform traffic pattern have shown significant improvements of the zero load latency and saturation throughput metrics.

**References**


