Politecnico di Milano - Department of Electronics, Information and Bioengineering

POLITECNICO
MILANO 1863

RTL Router Design in SystemVerilog

[Coding Project]

Student	Andrea Galimberti
ID	875434
Student	Filippo Testa
ID	875456
Student	Alberto Zeni
ID	884540
Course	Embedded Systems
Academic Year	2016-2017
Advisor	Davide Zoni
Professor	William Fornaciari

November 13, 2017

Contents

1 Introduction 3
- 1.1 Problem statement 3
- 1.2 Summary of the work 3

2 Design and implementation 4
- 2.1 Circular Buffer 4
- 2.2 Input Buffer 4
- 2.3 Route Computation Unit 4
- 2.4 Input Port 5
- 2.5 Crossbar Switch 5
- 2.6 Round-Robin Arbiter 5
- 2.7 Separable Input-First Allocator 5
- 2.8 Virtual Channel Allocator 5
- 2.9 Switch Allocator 5
- 2.10 Router 6
- 2.11 Mesh 6

3 Verification 9
- 3.1 Circular Buffer 9
- 3.2 Input Buffer 9
- 3.3 Route Computation Unit 9
- 3.4 Input Port 10
- 3.5 Crossbar Switch 10
- 3.6 Round-Robin Arbiter 10
- 3.7 Separable Input-First Allocator 11
- 3.8 Virtual Channel Allocator 11
- 3.9 Switch Allocator 11
- 3.10 Router Submodules Integration 12
- 3.11 Router 12
- 3.12 Mesh 12
- 3.13 Verification Scenarios 12
  - 3.13.1 Single Router Scenario 13
  - 3.13.2 Two Routers Scenario 13

4 Router External Interface 14
- 4.1 Inputs and Outputs 14
- 4.2 Interfaces to Other Routers 14
  - 4.2.1 Router-Router Interface Signals 15
  - 4.2.2 Router-Router Interface Modports 15

5 Flit Encoding Scheme 15

6 Conclusions and Future Works 16

1 Introduction

1.1 Problem statement

Interconnection networks are used to interconnect either single or sets of components within a computer system, or multiple computer systems; the interconnection aspect of computer architecture has gained significant importance in recent years, with regard to the impact of different solutions on the overall performance and cost of the whole system.

In addition to providing external connectivity, networks are commonly used to interconnect the components within a single computer at many levels, including the processor micro-architecture; networks have long been used in mainframes, but today such designs can be found in personal computers as well, given the high demand on communication bandwidth needed to enable increased computing power and storage capacity.
Switched networks are replacing buses as the normal means of communication between computers, between I/O devices, between boards, between chips, and even between modules inside chips.

One of the main networking domains, and the subject of this project, is Network-on-Chip (NoC), in which the interconnection network is used for interconnecting micro-architecture functional units, register files, caches, compute tiles and processor and IP cores within chips or multi-chip modules; current and near future NoCs support the connection of a few tens to a few hundreds of such devices with a maximum interconnection distance on the order of centimeters.

The router represents the key component in the Network-on-Chip, thus this project aims to design and implement a simple Network-on-Chip router, with a four stage architecture and supporting virtual channels, and properly verified by means of testbenches.

1.2 Summary of the work

A Network-on-Chip interconnection module has been developed, with a 2D mesh topology, and making it possible to connect computing nodes either in a direct or indirect network, based on how these nodes are connected to the mesh module.

The routers allocate data at flit granularity, implementing a wormhole switching architecture further optimized by the presence of multiple virtual channels per input, avoiding the Head of Line blocking issue and thus allowing an higher average throughput to the network.
Packet routing is driven by the Dimension Order Routing algorithm, independently computed by each router belonging to the mesh.
Flow control is implemented in the switching activity management, which is controlled by a per-router switch allocation unit, and uses the On-Off algorithm, easy to implement but efficient enough for a medium level of traffic in the network.

2 Design and implementation

The Network-on-Chip router has been developed following a bottom-up approach, for easier testing purposes, as simple, lower-level modules were implemented and tested before moving on to higher-level modules.

A strong emphasis has been put on simplifying the connections between the submodules of the router, thus the *interface* construct provided by SystemVerilog has been widely adopted; this has also helped in decoupling the functional part of the modules from their I/O specification, making the connections and interactions between them easier to understand even from the source code.

For each virtual channel at each port, a single-bit error output is implemented to signal to the external environment the detection of errors such as inconsistencies in the signals between allocators and input ports or the reception of ill-formed packets at some port of a router.
This mechanism will make it possible to verify that the interconnection network works correctly, with respect to most critical corner cases, also when deployed in a real-life scenario.

2.1 Circular Buffer

A circular buffer is the main responsible of storing flits at intermediate nodes while routing them to their destination.
Simple read/write operations are allowed, and empty and full single-bit outputs signal these two peculiar conditions of the buffer.
The on/off single-bit output gives signals to the connected upstream router whether the buffer can accept more flits or not.

2.2 Input Buffer

The input buffer module encapsulates a circular buffer and surrounds it with status information about the next hop for the current packet and the state of the internal finite state machine, which can be either idle, virtual channel allocation or switch allocation.
The internal FSM of the input buffer is in the idle state when the buffer is available to be allocated to a new incoming packet, in virtual channel allocation state when the buffer has received at least the head flit of the packet and is waiting for the allocation of a buffer at the downstream router, and in the switch allocation state while flits (up to the tail of the packet) are sent to the assigned downstream virtual channel and contend for the access to the crossbar switch with the other buffers from the same input port and the other input ports.
A dedicated single-bit flag signals to the connected upstream router if the buffer is available to be allocated to a new packet.

2.3 Route Computation Unit

A purely combinational logic module, as the implemented routing algorithm requires only information about the position of the current router and of the destination router in the 2D mesh, the route computation unit computes the next hop for each packet as soon as the head flit arrives to the router (as the header of an head flit encodes the position of the recipient of the packet), and sends the result of its computation (i.e., the output port from which the packet will have to go out of the router) to the input buffer in which all the flits of the packet will be stored.

2.4 Input Port

The input port module contains a configurable number of virtual channels and a route computation unit; each input port is connected to the virtual channel allocator, switch allocator and crossbar module of the containing router.
Only one flit per clock cycle can enter the input port from an upstream router, and is stored in the virtual channel corresponding to an identifier properly encoded in the flit itself; in the same way, only one flit can be read from a selected virtual channel at each clock cycle, and sent to the crossbar.

To simplify connections with the interfaces provided by the other modules composing a router (while no functionality is modified at all), input ports are grouped in a **input block**, one per each router, containing a number of ports specified by a parameter (e.g. 5, in the 2D mesh topology case).

2.5 Crossbar Switch

This module enables moving flits from each input port to the outputs of the router, implemented in a way such that each output can propagate at most one of the inputs; the selection of the inputs to be propagated at each output is delegated to a connected auxiliary module, the switch allocator.

2.6 Round-Robin Arbiter

A standard round-robin arbiter is the basic component of more complex allocators, which will be used in the router to solve contention of the virtual channel buffers and the crossbar switch.
This arbiter module can be parametrized for what concerns the maximum number of requests of a shared resource, and the round-robin scheduling algorithm has been chosen for its simplicity, easy implementation and for being starvation-free.

2.7 Separable Input-First Allocator

An allocator allows managing contention of multiple shared resources among multiple agents; the module is implemented such that, first, at each input port round-robin arbitration is performed between the virtual channels requesting for the allocation of any output port (indeed, the allocator is input-first), then, at each output port, round-robin arbitration is performed between input ports requesting for its allocation.
This separable input-first allocator module is used inside the two following allocator units, as it encapsulates all the allocation logic and the containing modules.

2.8 Virtual Channel Allocator

This module produces a request matrix as an input to the contained separable input-first allocator; this matrix is produced based on the availability of virtual channels at the destination downstream input port, i.e., whether at least one input buffer of the port is in idle state.
From the internally computed grant matrix, the virtual channel allocator module produces the identifiers of virtual channels at downstream routers for the input buffers who won contention.

2.9 Switch Allocator

The switch allocator module solves the contention between input buffers for the access to the crossbar switch, by encapsulating a separable input-first allocator and thus sending control signals to the cross-

bar module and to the input buffers.
Access to the shared crossbar switch depends on previous grants to the resources (with a Round-Robin policy) and on the availability, from the flow control point of view, of the downstream virtual channel that has been assigned to the upstream input buffer.

2.10 Router

Each router module contains a variable number of input ports, a crossbar, a virtual channel allocator and a switch allocator, all properly connected to each other through the related interfaces, as shown in Figure 1.

At each port, two interfaces (one for the router to act as the downstream, and the other to act as the upstream of the communication) are provided to connect the router to another one; the interface exposed by each router module is explained in detail in Section 4, and a schematic view can be seen in Figure 2.

2.11 Mesh

The mesh module contains a variable number of routers connected in a direct network with a 2D mesh topology, and both dimensions of the mesh can be defined as parameters of the module; another parameter of the mesh that has to be set at this final stage is the size of the input buffers (i.e., the number of flits that can be stored at each virtual channel).

This module provides an interface to which external nodes can be connected in order to communicate with each other by means of a Network-on-Chip interconnection network.

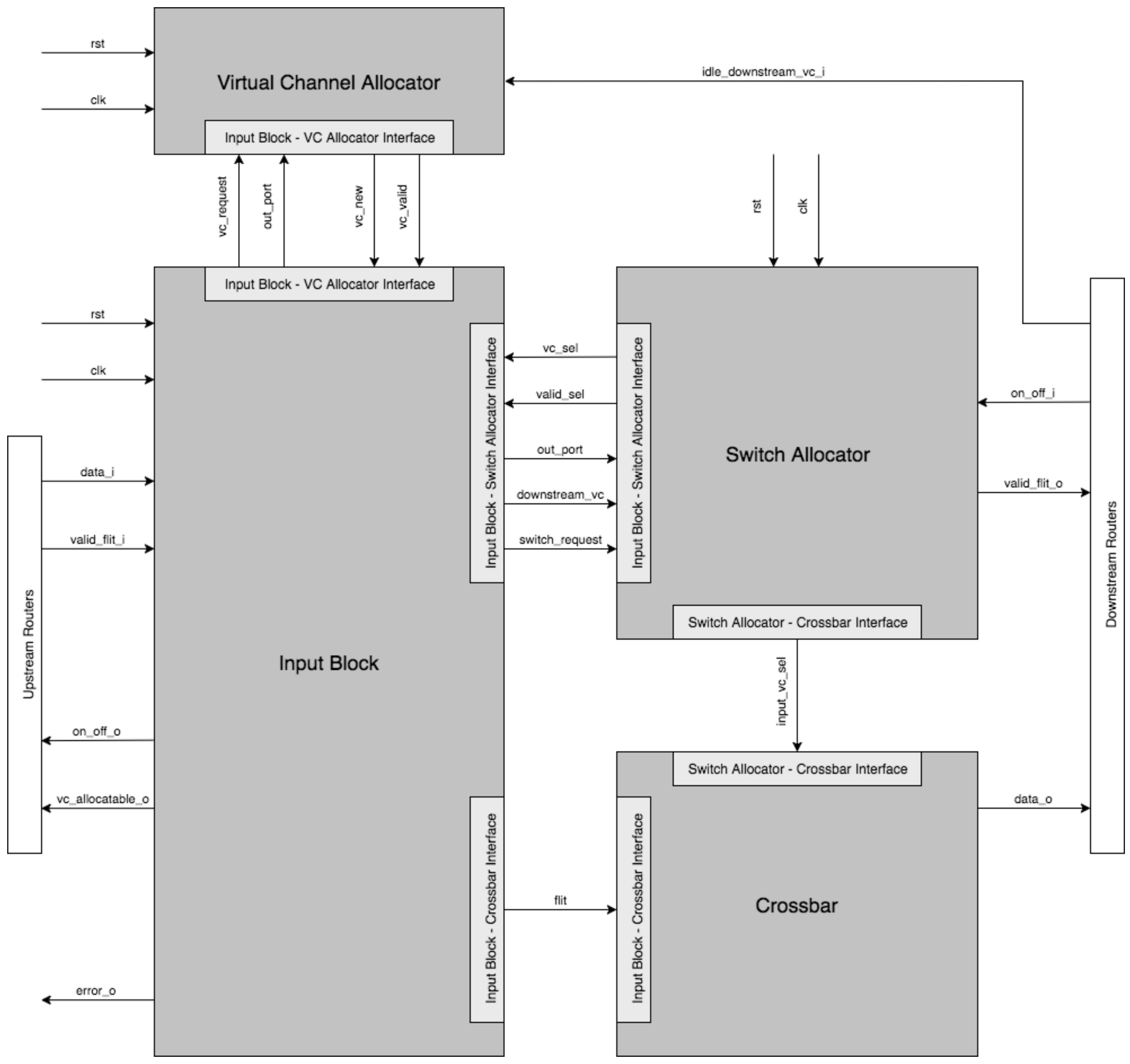
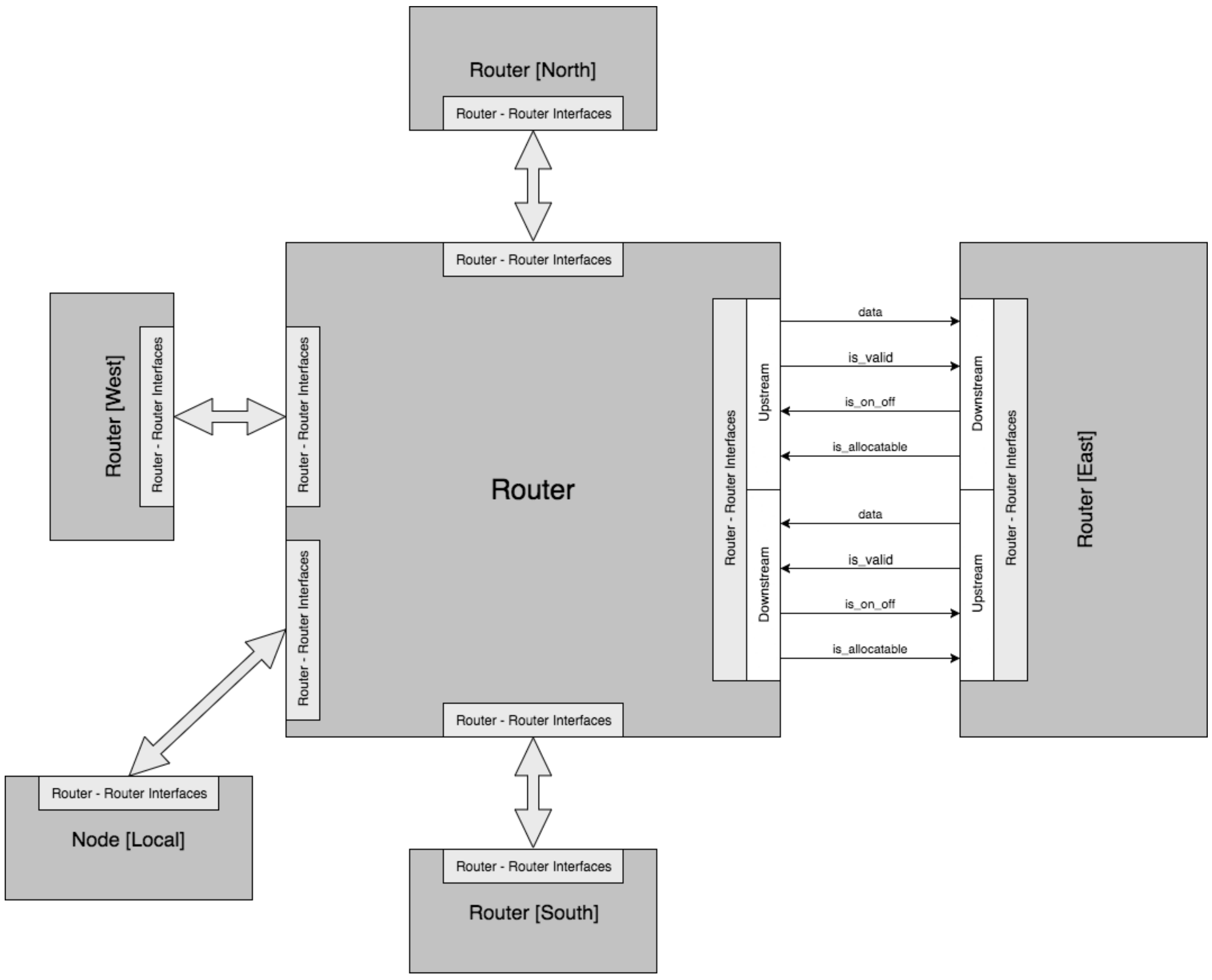

Virtual Channel Allocator
rst
clk
idle_downstream_vc_i
Input Block - VC Allocator Interface
vc_request
out_port
vc_new
vc_valid
rst
clk
Input Block - VC Allocator Interface
rst
clk
Input Block - Switch Allocator Interface
vc_sel
valid_sel
out_port
downstream_vc
switch_request
Input Block - Switch Allocator Interface
Switch Allocator
on_off_i
valid_flit_o
Upstream Routers
data_i
valid_flit_i
Input Block
on_off_o
vc_allocatable_o
error_o
Switch Allocator - Crossbar Interface
input_vc_sel
Switch Allocator - Crossbar Interface
Downstream Routers
data_o
Input Block - Crossbar Interface
flit
Input Block - Crossbar Interface
Crossbar

Figure 1: Detailed view of the connections between internal submodules



Router [North]
Router - Router Interfaces
Router - Router Interfaces
Router [West]
Router - Router Interfaces
Router - Router Interfaces
Router
Router - Router Interfaces
Router - Router Interfaces
Upstream
Downstream
data
is_valid
is_on_off
is_allocatable
data
is_valid
is_on_off
is_allocatable
Downstream
Upstream
Router - Router Interfaces
Router [East]
Router - Router Interfaces
Router - Router Interfaces
Node [Local]
Router - Router Interfaces
Router [South]

Figure 2: Detailed view of the external interface

3 Verification

The verification phase has not been approached from a complete functional testing point of view, as developing a complete testbench for the whole developed system would have required a disproportionate effort with respect to the desired quality level.
Instead, testing properly chosen corner cases has been deemed enough for the verification purposes of this project; thus, a testbench module is used in order to test each router submodule, the router and the mesh.

In particular, each testbench contains the module to test, later on simply referred as design under test (DUT), and the necessary logic to both steer the input signals of the DUT and check its output signals.
Each testbench verifies the correctness of the DUT output signals by checking them against the expected output values inside a scoreboard structure.

Testing of single SystemVerilog modules and of multiple modules interacting with each other has been performed with a bottom-up approach, alongside the development phase, both by manually analyzing the VCD waveforms produced by simulations in the Vivado environment and by means of testbenches self-evaluating the results of the same simulations.

All the tests have been executed in the Vivado HL WebPack IDE (2017.2 version) on Linux machines.

3.1 Circular Buffer

For this basic element of the NoC router, we designed a simple testbench that supports three different operations: flit read, flit write and simultaneous read and write of different flits. Then, for a fixed number of times, an operation is picked up in a random way and synchronously executed on the DUT. The written flits are only of head type, for the sake of simplicity, and all the other fields are filled with a progressive integer because at this early stage of development it is sufficient to check the correct behavior of the circular buffer with respect to the singular flits, that is the flits have to exit the buffer (when reading) in the same order as they have been written.

3.2 Input Buffer

The verification of this module includes also the handling of well-formed packets; thus, it is more complex with respect to the previous circular buffer and both reading and writing operations now take care of packets rather than single flits.
The test inserts different packets and, through the scoreboard, it ensures the flits exit from the DUT in the expected order and with correct field values. Simultaneously, a parallel check to the flits operations aims to verify the correct values of the output signals related to the VC management.
The testbench for the input buffer is quite simple and limited, as a deeper verification will be carried out on the input port module; at this step of verification, it is deemed enough to prove the correct behavior of the single input buffer just for single packets.

3.3 Route Computation Unit

The testbench for the route computation unit has a simple structure; indeed, it does not require any scoreboard mechanism but it uses only a dedicated function for comparisons. Using arbitrary values as mesh dimensions and a fixed X,Y location for the DUT, the main task relies on nested loops to compute all the possible destinations; for each of them a simple comparison is done between the effective out port from the DUT and the expected outcome, that is the out port computed by the testbench.

3.4 Input Port

The testbench for this module is more structured than the previous ones because it involves the simultaneous verification of multiple input buffers, considered as Virtual Channels contained within an input port.
The designed testbench allows some degree of freedom to specify different parameters for every test that will be executed: it is possible to choose which VCs to use, the precise structure of the packet (size and number of different type of flits), possible delays between the arrival of flits and the timing for the signals about Allocation phases.
Multiple tests are provided with the objective to verify the behavior of the DUT in some corner cases, such as packets without body flits, longer than the buffer length, with multiple head flits, without leading head flit and single-flit packet. Also, the DUT is tested with simultaneous operations involving all available VCs.

3.5 Crossbar Switch

The verification of the crossbar would require also the deployment of input block and switch allocator modules, therefore the following testbench would be quite complex to implement and not very effective for detecting possible errors in the crossbar design. For this reason, we used a Mock module that has as inputs the signals of both input block and switch allocator, thus allowing us to design a simplified logic to steer the DUT.
Therefore, for this testbench there is no need for the usage of a scoreboard structure and the test verifies that all possible combinations between input and output ports return the correct flit.

3.6 Round-Robin Arbiter

The arbiter allocates resource it monitors to one of the agents that requested it, each time by upgrading and following the priority of each agent; in order to check the correctness of the round robin arbiter we need to be sure that the grants have been asserted only to the agents that requested them, and we also need to check that in case of conflict only one agent has received the grant, and it is the one with highest priority of those in conflict.

The testbench thus checks for each clock cycle that the agent which has been granted is unique and that it has requested the access to the resource, also that it was the one with the highest priority with respect to the ones that requested the resource; a simple starvation control is included, i.e., whenever one agent requests a resource but it does not receive the grant a counter for that agent is incremented, and if this happens more than a fixed number of times the starvation signal is raised for the correspondent agent.
The test fails whenever the granted signals are more than one or when the grant is assigned to an agent that didn't request the resource or when the granted agent is not the one with highest priority among the requesting ones.

3.7 Separable Input-First Allocator

The testbench for the separable input-first allocator is pretty straightforward, as it only checks the correctness of the grants for the given inputs by simulating the module's logic.
The testbench takes the requests for output ports from all the virtual channels from all the input ports and generates two sets of grants, each of those with respect to the entity that requested them; when both of those two grants sets have been computed, then the grant array for the output ports is generated. The test fails whenever there's a mismatch with the grants for the output port in the module.

3.8 Virtual Channel Allocator

The test of the virtual channel allocator would require the deployment of the input block module, therefore we used a mock module, such as in the crossbar testbench, for convenience; the module has as inputs the signals of the input block, allowing us to design a simplified logic to steer the DUT.
For this testbench we decided to test some of the basic tasks that the allocator has to perform and also some corner cases. The testbench also implements the logic of an input-first allocator, in order to check the correctness of the virtual channel allocations.

The core of the testbench simulates the logic of the virtual channel allocator and then tests its results with the ones of the real module. The testbench takes the signals for the request of virtual channels and the ones for their availability, which are given inputs that we control, and arbitrates those through the input-first allocator logic. The allocation signals are then asserted based on the grants of the input-first allocator. The availability of the virtual channels and the outputs signals of their assignment are compared with the ones of the module, and the test obviously fails when there's a mismatch on this signals.

Aside from simulating the logic of the tested module, the testbench contains different scenarios, each one covering various parts of the allocator behaviour. One scenario performs randomly generated requests to the module, and also assigns random availability to each of the virtual channel; this scenario covers only basic tasks of the allocator and we used it to check that the main logic were correct.
The other scenarios cover some corner cases which will be surely encountered by our allocator when it will be used in a real NoC; they cover the situations where all the upstream virtual channels request the same port, all the downstream virtual channels are not in idle for five clock cycles and then they start to return available, a reset before the operations.
All the scenarios are tested with the simulation and compared to the real module to check any incoherence.

3.9 Switch Allocator

In order to check the correct behaviour of this module it would be necessary to also have already deployed both the crossbar and input block module, therefore the testbench requires the usage of a module mocking these two modules.

The testbench simulates the logic of a separable input-first allocator, in order to check the correctness of the grants associated with the given inputs, then it simulates the assignments and the related control signals that the switch allocator generates from the computed grant matrix; after generating those values, a check is performed comparing the values produced by the module and the ones produced by the testbench.

The test produces a randomly generated set of inputs in order to check the main behavior of the module, since this will cover the majority of the cases that our module will encounter in a real NoC, and the test will fail whenever the values from the testbench and the values from the module differ.

3.10 Router Submodules Integration

Once that all modules of the Router have been tested alone, it is possible to proceed with an integration step and verify the overall behavior of these modules together; this stage of verification then relies on a testbench that involves Crossbar, Switch Allocator, VC Allocator and Input Block.
It is important to remark that we chose to design this testbench, before the one for the complete Router, because we wanted to ensure the correct matching between the inner signals.
For the reason just mentioned, the testbench simply tests whether a well-formed packet goes through the Input Block and come out of the Crossbar in the correct order with respect to the one stored in the scoreboard. A more comprehensive testbench is provided for the complete Router module.

3.11 Router

The testbench for the Router requires also the implementation of a Mock module to establish the proper connections with the interfaces. Using a similar structure to the one used for the Input Port, the testbench verifies the DUT behavior with well-formed packet and under some specific corner cases such as: full packet with delay between flits, packet without body flits, long packet that exceeds the length of the internal buffer, ill-formed packet with multiple head flits and packet without the head flit and, finally, with single flit packet.

Also for this module, the verification is done both using the scoreboard, for checking the correct order of the flits, and manually analyzing the VCD, for spotting any eventual inconsistency with the expected behavior.

3.12 Mesh

This testbench aims to verify the communication between only two routers and then through the mesh; the verification step of two connected routers is crucial for the verification of the complete mesh because once it is ensured that the communication between each pair of routers works properly, as a consequence the communication across the entire Mesh will work as well.
In the former case, two connected routers are tested in an exhaustive way, ensuring that all possible combinations among input ports of the first one and output ports on the second returns the correct ordering of the flits according to the ones stored in the scoreboard.
In the latter case, instead, a well-formed packet is sent through a 2x3 bidimensional mesh, and the outcome of test is inspected simply analyzing the VCD.

3.13 Verification Scenarios

This section briefly reports, in a compact and structured way, all the tests that have been carried out on both a Router module alone and a pair of connected Routers.
As already mentioned in the first part of the Verification section, it is important to remark that the implemented tests do not ensure an extensive full coverage of all the possible cases, but instead they focus on some particular corner cases.

3.13.1 Single Router Scenario

In a single Router scenario, the tests aim to ensure the correct management of packets, and their flits, with respect to the Virtual Channels but also to check the internal components as well.

Test No.	Test Description
1	Single, well-formed packet with 4 flits
2	Simultaneous insertion of two well-formed packets (4 flits) with different destination addresses, that means different outports
3	Simultaneous insertion of two well-formed packets (4 flits) with the same destination address, that means same outport
4	Simultaneous insertion of two well-formed packets (4 flits) with the same destination address and with delay between flits arrival
5	Simultaneous insertion of two well-formed short packets composed only of HEAD and TAIL flits, with the same destination address
6	Simultaneous insertion of two well-formed packets with different sizes that exceed the buffer length and with the same destination address
7	Simultaneous insertion of two ill-formed packets, that is with multiple HEAD flits
8	Simultaneous insertion of two different single-flit packets
9	Insertion of an ill-formed packet without the leading HEAD flit

3.13.2 Two Routers Scenario

This verification step, which is fundamental for the test of the complete mesh, checks in an exhaustive way the communication between two attached routers.

Test No.	Test Description
1-16	Each test inserts for every pair of input port of the first Router (more precisely Local, North, West and South) and output port of the second (that are Local, North, East and South) a well-formed packet with a proper and precomputed destination address to reach the output port currently under test.

4 Router External Interface

In the following section, an overview of the interface exposed by the Router is provided, detailing the inputs and outputs of the module and the interfaces that enable connecting one router to another, in order to obtain a 2D mesh structure.

4.1 Inputs and Outputs

	Signal	Size (bits)	Description
Inputs	*clk*	1	Clock signal input to the Router module
	rst	1	Asynchronous reset signal, resets the Router to the initial state, in which all the buffers are empty and the internal allocators have no previous history
Outputs	*error_o*	# ports x # virtual channels	Error flags, each corresponding to an Input Port - Virtual Channel pair, signal errors for the flits stored at the related buffer

4.2 Interfaces to Other Routers

	Interfaces	Description
Upstream *router2router.upstream* *router_if_[link]_up*	*router_if_local_up* *router_if_north_up* *router_if_south_up* *router_if_west_up* *router_if_east_up*	Router-Router interfaces (respectively at the Local, North, South, West and East links), where the Router acts as the upstream entity (i.e., it sends the packet flits to the downstream Router connected on the other end of the link)
Downstream *router2router.downstream* *router_if_[link]_down*	*router_if_local_down* *router_if_north_down* *router_if_south_down* *router_if_west_down* *router_if_east_down*	Router-Router interfaces (respectively at the Local, North, South, West and East links), where the Router acts as the downstream entity (i.e., it receives the packet flits from the upstream Router connected on the other end of the link)

4.2.1 Router-Router Interface Signals

Signal	Size (bits)	Description
data	# bits per flit	Data lines transporting one flit at a time from the upstream Router to the downstream one *(see the Flit Encoding Scheme section for more detailed information about how the flit_t type has been defined)*
is_valid	1	Flag signaling the validity of the flit currently on the data line
is_on_off	# virtual channels	Flag representing the availability, from the on/off flow control point of view, of the virtual channels of the downstream router's input port to receive more flits
is_allocatable	# virtual channels	Flag representing the availability of the virtual channels of the downstream router's input port to be assigned to an upstream virtual channel in order to receive a packet (i.e., from the head flit to the corresponding tail flit)

4.2.2 Router-Router Interface Modports

Modport	Inputs	Outputs
Upstream	*is_on_off* *is_allocatable*	*data* *is_valid*
Downstream	*data* *is_valid*	*is_on_off* *is_allocatable*

5 Flit Encoding Scheme

Section		Subsection	Description
flit_label			Label representing the type of the flit, which can be either HEAD, BODY, TAIL or HEADTAIL *(see flit_label_t type definition)*
vc_id			Identifies the virtual channel in which the flit has to be stored at the downstream input port
data, which is either	*head_data*	*x_dest*	X coordinate in the 2D mesh of the recipient Router
		y_dest	Y coordinate in the 2D mesh of the recipient Router
		head_pl	Payload of the flit
or	*bt_pl*		Payload of the flit (larger than the HEAD flit one, as the recipient address is not needed in BODY and TAIL flits)

6 Conclusions and Future Works

The Network-on-Chip has been designed, implemented and then verified on some more complex corner cases, manually defined as the most critical and significant ones, alongside with other simple random tests.

Some improvements can be seen as a natural continuation of this project in the future.
From the design point of view, an higher performance could be obtained by adding speculation to the router, i.e., speculating the outcome of the virtual channel allocation in order to carry it out at the same clock cycle as the switch allocation for the head flit of each packet.
Meanwhile, from the verification perspective, a more complex testbench could be implemented in the future for a more complete functional coverage of the design under test, with random tests being subsequently generated from the feedback of previous ones.