Modeling and Simulating Network-on-Chip Designs: A Case Study of Fat Tree Interconnection Architecture

Azeddien M. Sllame and Asma Alasar

***Abstract*—In this paper we describe a fat-tree based Network-on-Chip (NOC) system that composed of processing nodes and communication switches. The IP node contains message generator and buffering. The switch uses wormhole technique which improved by virtual channel mechanism. The switch includes the following essential units: the router, input/output link controller units and arbitration unit. A discrete event simulator has been developed in C++ to analyze the proposed architecture. The obtained results clearly demonstrate both the efficiency and the applicability of fat tree structure to NOC design. In addition, VHDL code for the proposed algorithms has been prototyped in FPGA technology.**

***Index Terms*—Network-on-chip, routing, switching, fat tree.**

I. Introduction

One of the important key design issues in the *multiprocessing system-on-chips* (MPSOC) paradigm is the interconnect topology. In the last decades point-to-point communication links were used because the design model was based on almost a single processor with support of small number of *application specific integrated circuits* (ASICs). Nowadays, this is not working any more because of the latest advances in semiconductor technologies which enable integrating many multiprocessing elements (IP cores) in a single chip. So instead of connecting the top-level SOC modules by routing dedicated wires, they are connected to a network that routes packets between them; see Fig. 1.

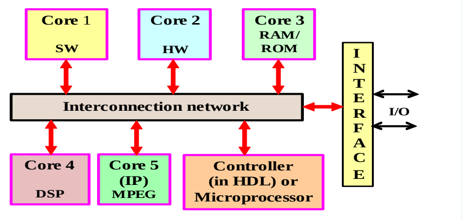


Fig. 1. Generic SOC structure

This approach has the benefits of being modular, well-structured, flexible, and has efficient performance. In addition, interconnection networks are already used in many super-computers and parallel systems in industry and academia for many years. Moreover, point-to-point wiring between IP cores have the following disadvantages; power dissipation, cross talk delays due to routing inside the chip, and slow propagation velocity. Whereas, interconnection networks had structured wiring that reduces the mentioned above disadvantages. Furthermore, in interconnection networks, when one IP block is idle, other IP blocks continue to make use of the network resources. Butterfly fat tree (BFT) and MESH architectures are typical examples of those interconnection networks [1], [2]. Interconnection networks can be classified according to different characteristics. Their topologies fall into two classes *static* (or *direct*) and *dynamic* (or *indirect*). In the static interconnection networks, point to point links interconnect the network nodes in some fixed regular topology such as mesh or hypercube. The dynamic interconnection networks allow the interconnection pattern between the network nodes to be varied dynamically: this is accomplished by using some form of switching. Examples of dynamic networks include fat trees and multistage networks. In such systems, routing algorithms and switching techniques are the two main factors that control network latency and throughput, and realize the overall network performance [2].

In this paper, we will describe a simulator for NOC system based on fat tree interconnection network architecture. The system clearly illustrates the traffic movement on the flit-level step-by-step between IP nodes through the communication switches. The simulator can be used to evaluate the effect of: routing algorithm; virtual channel mechanism; buffer management, message latency and system performance. The results obtained obviously demonstrate that: the fat tree interconnection networks can offer an attractive alternative solution for NOC interconnection because of its scalable structure and the bandwidth available.

This paper is organized as follows: definitions and terms are given in Section II. Commonly used switching techniques are briefly described in Section III. Related work is presented in Section IV. Fat tree construction is given in Section V. Section VI explains some details of our proposed switch. Section VII outlines the simulator structure. Finally, results are given in Section VII.

II. Definitions and Terms

The following terms and definitions are needed to understand the paper context [1], [3], [4]:

Core (node): defined as any reusable design block, i.e. can be used as building blocks within chip designs in hardware or a sub-component in software programs. We use the term IP (Intellectual Property) to refer to copyrights. In this paper we are using the following names interchangeably (IP node, IP block, IP core, logic core, component, processing element).

Switch: it is responsible for forwarding (switching and routing) packets from sender to the intended destination using suitable techniques to guarantee this function with

Manuscript received October 2, 2012; revised January 27, 2013.

The authors are with the Computer Department, Faculty of Science, Tripoli University, Tripoli, Libya (e-mail: aziz239@yahoo.com, asma44441@yahoo.com).

proper flow control and reasonable quality of services.

Message: is a unit of information from the programmer's perspective. The size is limited only by the user's memory space. See Fig. 2.

Packet: is the smallest unit of communication containing routing information (e.g., destination address) and the sequencing information in its header. Its size is of order of hundreds or thousands of bytes or words. It consists of header flit and data flits.

Flit: the smallest unit of information *at link layer* and its size of one of several words. Flits can be several types and flit exchange protocol typically requires several cycles. See Fig. 2

Phit: it is the smallest unit of information *at physical layer*, which is transferred across one physical channel in one cycle.

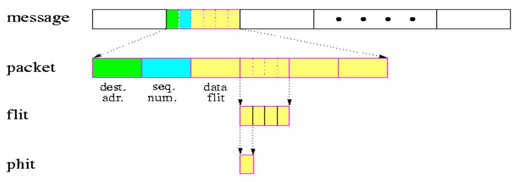


Fig. 2. Message structure (packet, flit, phit)

Routing algorithm: it determines the path selected by a packet to reach its destination, it must decide within each intermediate router which output channel(s) are to be selected to forward incoming packets.

Switching mechanism: it determines how network resources are allocated for data transmission, it is the actual mechanism that removes data from input channels and places them on the output channels.

Flow control: it defines the synchronization protocol between sender and receiver nodes which determines actions to be taken in case of full buffers, busy output channels, faults, deadlocks, etc. Flow control has two levels:

• *Packet flow control*: it performs synchronization between sender and receiver at the level of packet, ensuring successfully transfer and availability of buffer space at the receiver.

• *Physical channel flow control*: it implements the multi-cycle packet flow control and it breaks packets into flits. Here, even the flit may take several cycles to transfer; hence, the most elementary unit of information is the phit (physically).

Latency: is defined as the time elapses between the injection of header flit of a certain message into the network at source node and the arrival of the tale flit of the same message at the destination node. Average message latency can be calculated by the relation:

$$\text{Average Latency} = \frac{\sum_{i=1}^{i=P} Li}{P}$$

where *P* is the total number of messages reaching their destinations and *Li* is the latency of each message [1].

III. Switching Techniques

In this paragraph we are going to describe commonly used switching mechanisms. In c*ircuit switching* as in [2] ,[3], the process starts with transmitting routing probe into the network, which contains destination address and other control information to reserve the physical channel between source and destination, as it is transmitted through intermediate routers the path is setup. And when it reaches its destination, the source starts full message transmission at the full bandwidth of the reserved path. The disadvantage of this technique is the inefficient use of network resources because of path reservation between the sender and the receiver. In *packet switching* [1], [2], the message is divided into several fixed length packets, every packet consists of several flits, starting with the header flit, every channel has input and output buffer for one entire packet and each packet is routed individually from source to destination. Routing decisions are made by each intermediate router only after the whole packet was completely buffered in its input buffer. This is mechanism is also called *store and forward* (SAF). The disadvantage of this technique is its high storage requirements. In *virtual cut through* [1], [2], (VCT) messages are split into packets and routers have buffers for the whole packets as in packet switching. However, instead of waiting for the whole packet to be buffered, the packet is effectively pipelined through successive routers as a loose chain of flits. The *wormhole switching* [1], [2] works as a VCT scheme. The main difference is that, every router has a small buffer for one or few flits. The sequence of buffers and links occupied by flits of a given packet form a *wormhole* in the network. Wormhole routing allows building simple, small, cheap, and fast routers. Therefore, it is the most common switching technique used nowadays in commercial machines [5], [6]. The problem of degradation of throughput in wormhole switching is solved by the *virtual channel* concept [1], [2]. A physical channel may support several virtual channels multiplexed (time-multiplex) across the physical channel. They will all have their own buffers, but they will share one single physical channel medium. Each unidirectional virtual channel can hold, for example (see Fig. 3), four flits of the same packet, mixing flits from different packets is not allowed. Packets can share the physical channel on a flit-by-flit basis; the physical channel protocol must be able to distinguish between the virtual channels.

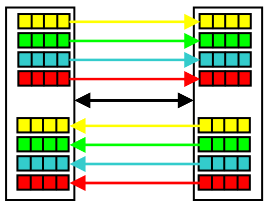


Fig. 3. Physical channel divided into four (yellow, green, blue and red) virtual channels

However, keeping adding virtual channels to further reduce the blocking; will result in increased network throughput in flits/second, due to increased physical channel utilization. However, increasing channel multiplexing reduces the data rate of individual message and increasing message latency. Nevertheless, general network throughput will be increased, if the number of virtual channels is reasonable, as we will see in our experiments graphs.

IV. Related Work

In 1985, Leiserson had proved formally that fat tree is the most cost-efficient for VLSI realizations [7]. Since then fat tree has got great attention and has been appeared in some super-computer architectures [1], [2]. In [3] ,[8] authors have been investigated how butterfly fat tree (BFT) as a structured network-based design paradigm can be easily meet specific clock cycle requirements when used as the overall MP-SOC interconnect architecture. The work illustrated that this type of interconnection networks can offer an attractive alternative solution for SOC interconnection that does not suffer from the non-scalability aspect of the buses in regards to the clock cycle problems [8]. However, their work in that paper is concentrated on wiring and clock constraints of the system. Whereas in [3] they described how the use of virtual channels can improve the system throughput with an extra increase of switches silicon area. Our work differs with the one described in [3] in the arbitration method, and routing algorithm and in the modular design of our switch. Our design also, includes separate set of algorithms for internal switch functionality such as input/output link controller and virtual channel management procedures. Moreover, our design differs also in the way of instantiating fat tree overall structure [4].

V. Fat Tree Interconnection Architecture

The Fat tree is a type of interconnection network, where the processors (processing cores or IP cores) are interconnected by a tree structure, in which the IP cores are at the leaves of the tree, and the interior nodes are switches. An advantage of a tree structure is that communication distances are short for local communication patterns. Moreover, the fat tree is a tree structure with redundant interconnections on its branches; the number of interconnections increases as the root is reached. The purpose is to increase the bandwidth at higher levels, where it is most needed. Because it is not feasible to provide a channel between every pair of nodes, the network channels are shared among the IP nodes. Messages are used to communicate between sending and receiving nodes, which means construction of paths that consisting some intermediate switches (for routing purposes) along the specified paths from the sources to the destinations. Fig. 4 shows a butterfly fat tree with 64 IP blocks (cores) interconnected by suitable number of switches in intermediate levels. The IP nodes are placed at leaves in zero level and switches are placed in higher levels. We can calculate number of levels by the relation:

$L = \log_4 N$, Where N is the number of IP nodes.

In our network we have 3 levels, and the switches are placed in levels ranging from $l > 0$ and $l >= L$. Each IP node is denoted by pair (i, 0) where i is ranging from (0-63) which denotes the index of the IP node in the level zero, each IP node has two ports to connect with its parent switches, each port has two unidirectional physical links. Each level of the network has the number $\frac{1}{2^{(l-1)}} \times \frac{N}{4}$ of the switches and the total number of switches in the network is the summation of the number of switches in each level. Each switch is represented by a pair of coordinates (i, l), where i represents the index of the switch in the level and l represents the level of the switch, the pair (5,2) represents switch no. 5 in the level no. 2. Each IP node at the coordinate *(i, 0)* has the parent at coordinate (p, 1) and $p=i/4$. For example if we have the IP Node (62, 0), it has the parent switch (15, 1). Each switch has two parent (p) coordinates ($p1$, $l+1$) and ($p2$, $l+1$)

$$p1 = \frac{i}{2^{(l+1)}} \times 2^{l} + i \bmod 2^{(2-l)}$$

$$p2 = p1 + 2^{(l-1)}$$

For example, if we have the switch (15, 1), then from the above relations it has the parents:

P1=6 and *p2*=7. And each switch has four children (switches or nodes).

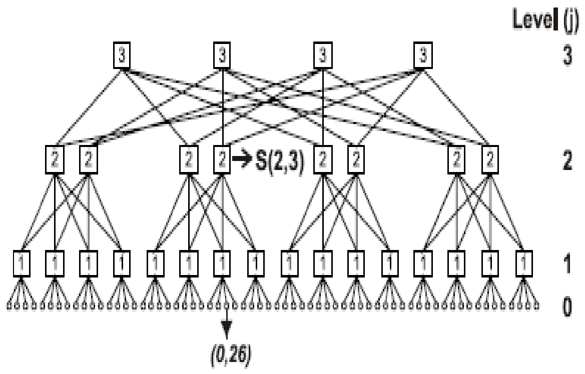


Fig. 4. Butterfly fat tree structure with 64 IP cores

VI. Switch Design

The switch is the basic component of the fat tree NOC and it performs the functions of routing and switching. The switch also ensures the storing of packets (the packet consists group of flits) to be transferred to other intermediate IP cores in the fat tree network. Each switch has two parent switches except for the top level switches and four children switches except for the lowest level switches. Lowest level switches have four children IP nodes. Each switch is bidirectional; each port is associated with a pair of opposite unidirectional channels, one for inputs and one for outputs. Fig. 5 illustrates the main components of the proposed switch.

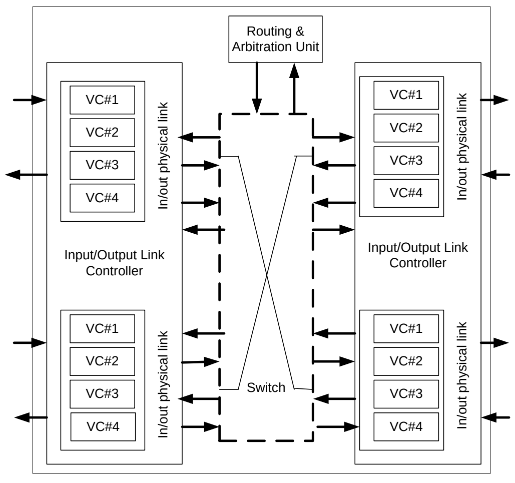


Fig. 5. The top-level of the switch structure

Each switch is constructed from the following entities:

A. Arbitration Unit

The switch includes an arbitration unit that is used to let one of the competing input messages grant access to the output port. The proposed arbitration unit used round-robin strategy with the help of *mapping table*. The mapping table is a data structure that is used to aid the arbitration process, and it is composed of an array of integers having the size of *noOfPhysicalLink* × *noOfVirtualLink*. This array is used for switching in crossbar switch i.e. it determines the path from an input buffer to the output buffer. At index *i* of this array a value *x* (>=0) is stored for the path selection. Here *x* is the output physical link identifier and *i* is the input virtual channel identifier. That means the data from input port *i* is switched (transferred) to the output port *x*. the input and output physical and virtual channels are identified as below:
Input physical channel number = *i* DIV number of virtual link
Input virtual channel number = *i* MOD number of virtual link
Output physical channel number = *x*

After the output port has been granted by the message, the message will be switched to that output port by an established link of the switch between the input link and the output link. When multiple messages simultaneously request the same link, the arbitration component must provide arbitration between them. If the requested link is busy, the incoming message remains in the current link. The arbitration unit is invoked whenever the arbitration is needed on a certain link.

```
The arbitration algorithm
Input: PriorityArray, Physical link number which needs arbitration
PriorityArray Dimension: NoOfPhysicalLinks × NoOfVirtualLinks
Output: Selected Virtual Channel (vc)
Procedure:
Start =Pysical link number×NoOfVirtualLinks
Min=PriorityArray[Start]
vc=0
for i= start+1 To i = start +
        NoOfVirtualLinks;
    I = i+1;
    if PriorityArray[i]<min then
       min=PriorityArray[i];
       vc=i MOD NoOfVirtualLinks;
    endif
endfor
return vc
End procedure
```

B. Input Link Controller Unit

Link controllers interconnect switches and define the fat tree topology, link controllers can be divided into input link controllers and output link controllers. Input link controller unit is responsible for receiving incoming flits from different IP’s and forwarding them to the associated units, with the help of using virtual channel technique. Moreover, it controls the input buffer, which composed of a FIFO memory for storing one or more flits; that are required for storing transferred data until the next channel is available. As we are using wormhole switching, virtual channel technique is implemented to avoid deadlock, in wormhole switching input buffer unit holds as many buffer objects as many virtual channels are required. Fig. 6 illustrates the flowchart of the procedure that is used in the designed switch to move flits inside the switch from the input buffers to the output buffers, making them ready to be sent out of the switch. However, the input link controller is responsible for:-

1) Checking the availability of free input virtual channel, if exists then it returns free virtual channel number;
2) Managing the sending out of flit that is available in input virtual channel buffer;
3) Helping do routing function by setting the outgoing physical link number that the flit occupying in the virtual channel must follow to reach the destination;
4) Keeping track of the outgoing physical link number that is used by flit occupying the virtual channel now;
5) Setting the path up using the outgoing virtual channel number for the flit occupying the virtual channel ;
6) Sending the flit by passing the front flit in the specified input virtual channel on the corresponding input physical link, and.
7) Doing Buffer management.

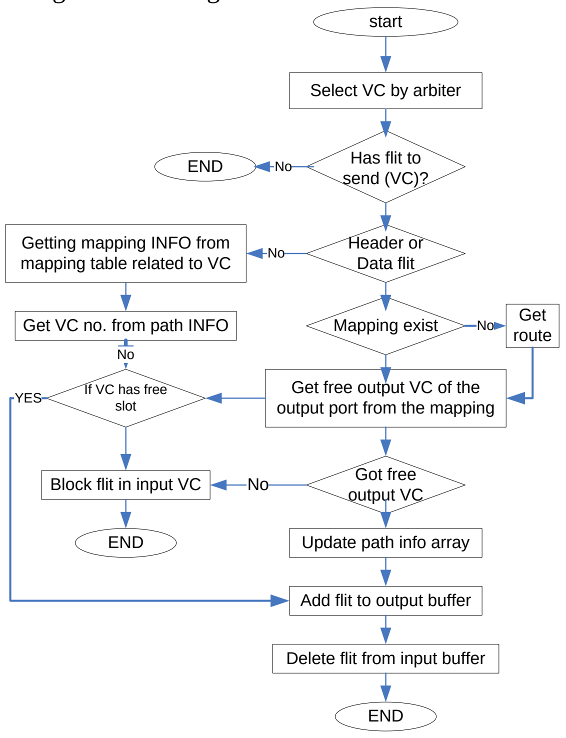


Fig. 6. Procedure of moving flit from switch input buffer to switch output buffer

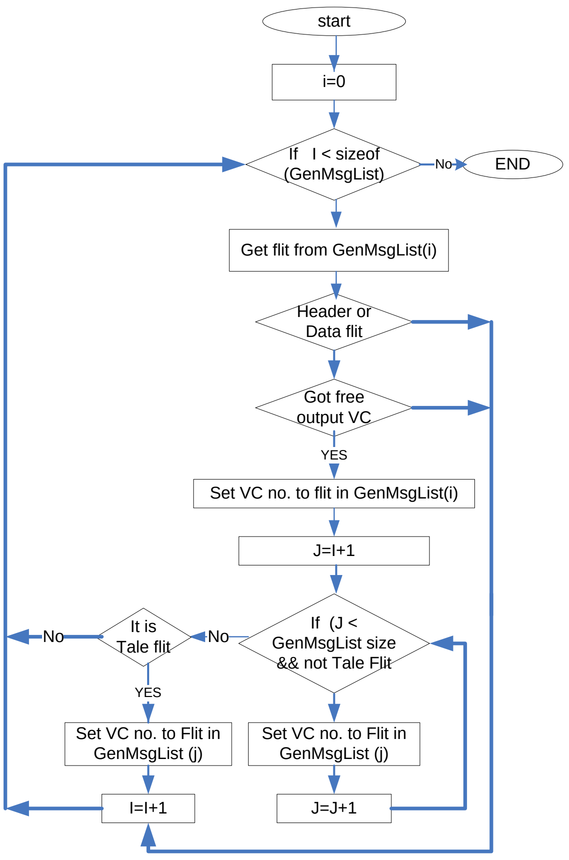
C. Output Link Controller Unit

This unit is responsible for receiving the incoming flits from the input controlling unit (after determining the appropriate output link controller number). Then, it forwards them to destination or to other switches, with the help of using virtual channel technique. Buffers at a specified virtual channel are used to help output link in performing its functions.

D. Routing Unit

It is the basic component of the switch that is responsible for applying the designed routing algorithm described below on the incoming flits to decide which output port to be used in the next step for forwarding (moving) the flit to the next switch or to the destination IP block. *Least common ancestor* *algorithm* is used in our design as a routing algorithm, as seen below [1], [4]. In addition, a set of functions and procedures have been developed to assign virtual channels to the flits in the generated message (packets), and to do flits movement internally/externally, in the switch to support the efficiency of the routing algorithm. However, Fig. 7 describes the steps of the procedure that is used to assign virtual channels to the transmitted message's flits list.

```
Routing algorithm
Input: SwitchLevel, SwitchIndex, Destination
Output: Selected output channel
Procedure:
Get switch‘s index
    Get Switch’s Level
Calculate low range and high range using the formula
        $Low\ Range = \frac{SwitchIndex}{2^{(SwitchLevel-1)}} \times (2 \times SwitchLevel)^2$
        $High\ Range = Low\ Range + (2 \times SwitchLevel)^2 - 1$
    If destination>=Low Range and Destination <=High
Range then $NumberofIP = High\ Range - Low\ Range + 1$
Childport=NumberIP/NumberChildPorts
Count=1
While Destination >=(Low Range + count ×Child Port)
   Count = Count + 1
While end
    Count = Count + 1
  Return Count
Else
  Get Random Parent Port Number
Endif
End Procedure
```



start
i=0
If I < sizeof (GenMsgList)
No
END
Get flit from GenMsgList(i)
Header or Data flit
Got free output VC
YES
Set VC no. to flit in GenMsgList(i)
J=I+1
If (J < GenMsgList size && not Tale Flit
No
It is Tale flit
No
YES
Set VC no. to Flit in GenMsgList (j)
Set VC no. to Flit in GenMsgList (j)
I=I+1
J=J+1

Fig. 7. Flowchart of procedure of assigning virtual channels to the flits of the generated message list

VII. Building the Fat Tree Simulator Structure

The network is the main entity in which all nodes and switches are connected to perform the functionality of generating, transferring, routing, switching and receiving of messages in the form of flits, see Fig. 8, which illustrates flit types and their structures. Network object contains list of IP nodes and a list of communicating switches. The network module is performing the task of generating the fat-tree NOC architecture, setting adjacent switches, moving flits through the network from output buffer to parent switches input buffer, moving flits from switches output buffers to nodes input buffers.

A. IP Node Structure

The IP nodes of the fat tree network are placed at the leaves in the level zero and connected with parent switches with two unidirectional physical links. In our proposed fat tree each IP node generates its own messages that are required to be sent to certain destinations. Those messages pass through the fat tree to reach the desired destinations. Each message has random data and it is generated at different random time stamps and has random message lengths. Each switch has six physical links, two for parent ports and four for children ports, each physical link has four virtual channels and each channel can hold of four flits per virtual channel. Each flit contains a field denotes the *flit type*, namely *header*, *data* or *tail*. The second field contains the virtual channel identifier (VCID). The third field contains packet length information, i.e., the number of flits in the corresponding packet. The next two fields give *source* and *destination* addresses. Header flit contents differentiate from data or tale flits, header flit contains the control information required to establish the path of the message from source to destination.

Header Flit

H	VCID	SRC	DEST	No. of flits

Data Flit

D	VCID	Data

Tale Flit

T	VCID	

Fig. 8. Flit types

B. Packet Generator

For simulation purposes we have created a unit in the C++ code named as *a packet* generator in the IP node architecture. The packet generator unit is in charge of generating data packets in random lengths. The packets are then passed from nodes to other nodes through using different switches of the fat tree NOC model.

VIII. Results

A discrete event simulator has been developed in C++ to analyze the proposed fat tree NOC architecture. In addition, a complete VHDL code for the proposed algorithms has been also written to demonstrate the correctness of the proposed system in FPGA chip (partially). Fig. 9 illustrates an example of packets (flits) flowing in the simulator when we send a message from IP node (3) to IP node (6) in the fat tree NOC example of 16 nodes for simplicity only (the system can handle even 64 nodes). For every simulation cycle the simulator performs the following operations:

1) Move flits from every node to adjacent switches;
2) Move flits from input buffer to output buffer of the switches;
3) Move flits from output buffer of the switch to the input buffer of the adjacent switches and/or nodes;
4) Move flits from the input buffer of the resource node to the received message list of that node;
5) At the end of the execution, the simulator calculates the network throughput on the level of switch;

Of course the simulator displays other parameters during the simulation such as number of IP nodes of fat tree structure, traversed switches, virtual channels/physical channels and the status of buffering inside the switches. At the end, the system calculates message latency, and network throughput. The simulator illustrates also traffic movement on the flit level by showing it step by step, for every IP node and switch. In traffic movement we can see many details about some flits from a particular packet (from a message) such as type of flit, current position of the flit i.e. all details of switching and routing (switch no., switch level, virtual channel no., physical channel no., received flit status, flit movement inside the switch from its input to its output ports, etc). Fig. 10 shows the relation between the system throughput and the number of virtual channels employed in the switch. We have tested the performance of the network throughput having a single buffer for each physical channel and having several buffers per physical channel. The results have shown increasing of switch throughput when the switch has two and four buffers per physical link and if the number of buffers increased, we will see that the switch throughput will not change significantly. Fig. 11 plotted the average message latency vs. the number of virtual channels. From the graph we can conclude that increasing number of virtual channels increases the message latency, due to switching between virtual channels.

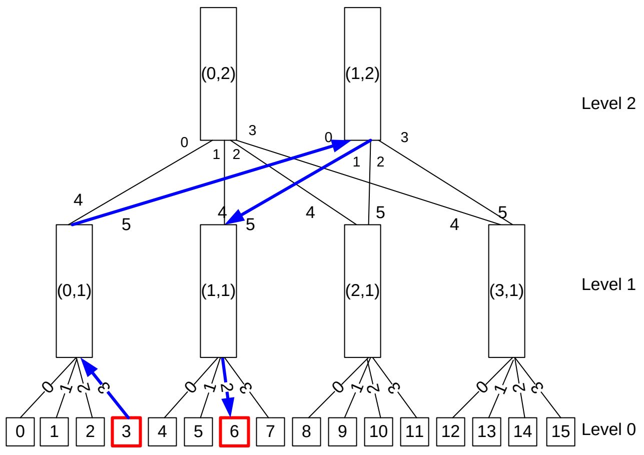


Fig. 9. Example: flit (3,6) is routed and switched from input link (5,0) to output link (2,0)

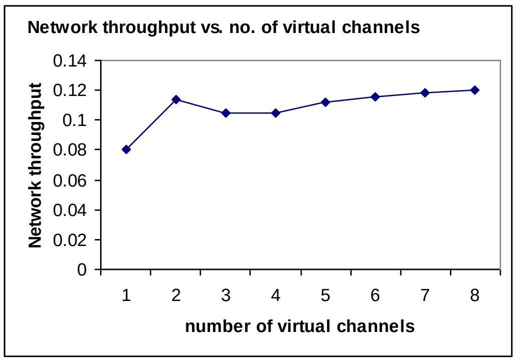


Fig. 10. Throughput vs. number of virtual channels

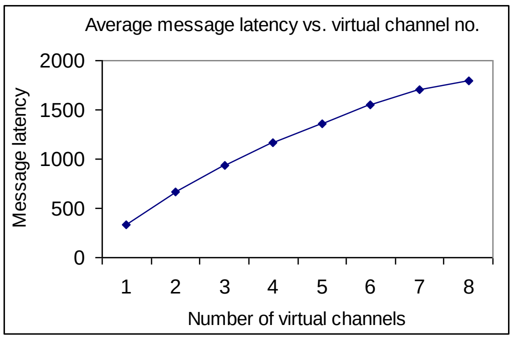


Fig. 11. Average message latency vs. virtual channels

IX. Conclusion

Eventually, we have successfully proven the correctness of all proposed algorithms and procedures of the modular NOC system simulator. The performance analysis of the network throughput, the virtual channel effect and message latency demonstrates the suitability of such system in modeling and simulating NOCs systematically. Also, a complete VHDL code has been written, simulated, and partially prototyped in FPGA technology. Our current work is to develop an extension to our work to support Mesh interconnection networks.

References

[1] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks – An Engineering Approach*, Morgan Kaufmann, 2002.

[2] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers, San Francisco, 2004.

[3] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a Switch for network on chip applications," in *Proceedings of ISCAS*, Bangkok, May 2003 Vol. V, pp. 217-220.

[4] A. Alasar, "Evaluation of system-on-chip interconnect architectures: a case study of fat-tree interconnection networks," MSc thesis, Computer Department, Faculty of Science, Tripoli University, Libya, 2010.

[5] J. Duato, O. Lysne, R. Pang, and T. M. Pinkston, "Part I: A theory for deadlock-free dynamic reconfiguration of interconnection networks," *IEEE Trans. on Parallel and Distributed Systems,* vol. 16, no. 5, pp. 412-427, May 2005.

[6] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet switched interconnections," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition 2000*, pp. 250-256.

[7] C. Leiserson, "Fat-trees: universal networks for hardware - efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892-901, October 1985.

[8] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh, "A scalable communication-centric SoC interconnect architecture," in *Proc. IEEE International Symposium on Quality Electronic Design*, ISQED 2004 San Jose, California, USA, 22-24 March, 2004.

Azeddien M. Sllame earned his B.Sc in Computer Engineering in 1990 from Engineering Academy, Tajoura, Libya. He got his M.Sc in Computer Science and Technology from Brno Technical University, Czech Republic in 1997. In 2003 he granted his PhD in Information Technology from Brno University of Technology. He published more than 20 scientific papers in many international conferences in the area of designing of high-performance digital systems, system-on-chip design techniques and evolvable hardware systems.

Asma Alasar got her MSc in Computer Science from Faculty of Science, Tripoli University, Libya in 2010. Her research interests include simulation and modeling of SOC designs.