An Efficient Switch for Fat Tree Network-on-Chip Interconnection Architecture

Azeddien M. Sllame
Computer Department, Faculty of Science
Tripoli University, University Road
Tripoli, Libya
Aziz239@yahoo.com

Asma Alasar
Computer Department, Faculty of Science
Tripoli University, University Road
Tripoli, Libya
Asma44441@yahoo.com

***Abstract*— this paper describes a fat tree based Network-on-Chip (NOC) system. The fat tree includes processing nodes and communication switches. IP node has a message generator unit which randomly generates messages to different destinations with different packet lengths and buffering. Switches use wormhole routing with virtual channel mechanism. The switch consists of the following units: router, input/output link controllers and arbitration units. A simulator has been developed in C++ to analyze the proposed architecture. Moreover, a VHDL model for the employed algorithms has been simulated and prototyped (partially) in FPGA technology.**

Keywords—fat tree; switching, routing, network-on-chip

I. INTRODUCTION

System-on-Chip (SOC) is an integrated circuit that implements most or all of the functions of a complete electronic system. The most fundamental characteristic of a SOC is its structure and connectivity complexity. In practice, most of SOCs are *multiprocessor systems-on-chips* (MPSOCs) because it is too difficult to design a complex SOC without making use of multiple CPUs. However, one of the important key design issues in the MPSOC model is the interconnect topology. In the last decades point-to-point communication links were used. Nowadays, 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".

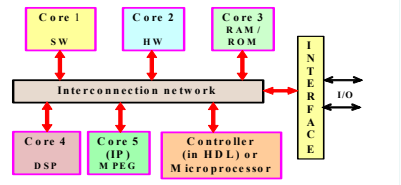

Figure 1. *Generic SOC structure*

This approach has the benefits of being modular, well-structured, flexible, and has efficient performance. Furthermore, interconnection networks are already used in many super-computers 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 in interconnection networks: when one IP block (core) 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 [1,3]. Interconnection networks can be classified according to different characteristics. Their topologies fall into two main classes static (or direct) and dynamic (or indirect). In the static network, point to point links interconnect the network nodes in some fixed topology; a regular topology as mesh or a hypercube is common. The dynamic network allows the interconnection pattern between the network nodes to be varied dynamically: this is accomplished by some forms of switching. Examples of dynamic networks include fat trees and multistage networks. Routing algorithms and switching techniques are the two main factors that control network latency and throughput, and realize the overall network performance [1,2,3]. The most commonly used switching techniques are including: circuit switching, store-and-forward, virtual cut-through, and wormhole routing [1,10]. However, wormhole routing is the most common switching technique used in commercial machines because it allows simple, small, cheap, and fast routers. Wormhole routers use often only input buffering. Blocking resources in case of stalled pipelines are the main drawback of this technique. Though, the problem of degradation of throughput in wormhole switching is solved by the virtual channel concept. Thus, several virtual channels can be 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 instance, four flits of the same packet and mixing flits from different packets is not allowed. Packets can share the physical channel on a flit-by-flit basis. Keeping adding virtual channels to further reduce the blocking; will result in increased network throughput in flits/second. This is due to increased physical channel utilization. On the other hand, increasing channel multiplexing will reduce the data rate of individual messages and this will increase message latency. This increase in latency will overshadow the reduction in latency that was caused by the blocking, which will lead to overall increase in average message latency. In contrast, the network throughput will be increased in general, if the number of virtual channels is reasonable as we will see in our experiments graphs.

978-1-4673-5157-7/13/$31.00 ©2013IEEE

This paper is organized as follows: definitions are presented in section 2. Section 3 summaries related work. Fat tree principles are summarized in section 4. Section 5 explains the proposed switch details. Section 6 outlines the fat tree simulator structure and IP node structure. Finally, results are given in section 7.

II. DEFINITIONS

Core (node): is defined as any reusable design block, i.e. can be used as building block 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: is responsible for forwarding (switching and routing) packets from sender to the intended receiver using suitable techniques to guarantee this function with proper flow control and reasonable quality of services.

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

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: it is the smallest unite of information at link layer and it is of size of one of several words. Flits can be several types and flit exchange protocol typically requires several cycles.

Phit: It is the smallest unite of information at the physical layer, which is transferred across one physical channel in one cycle.

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 for 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.

Latency: it 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:

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

Where P is the total number of messages reaching their destinations and Li is the latency of each message [1,10].

III. PREVIOUS WORK

In 1985, Leiserson had proved formally that fat tree is the most cost-efficient for VLSI realizations [9]. Since then, fat tree was got great attention and appeared in some super-computer architectures. Dally in [2] presented the usefulness of using on-chip interconnection networks in place of ad-hoc global wiring structures in SOC designs. In [6], authors evaluated the cost and the performance of a proposed interconnection network through joint functional modeling and physical implementation of some parts of the architecture. However, they used old communication protocols which are based on split transaction model for modeling the communications and they call it as *Scalable, Programmable, Integrated Network* (SPIN) for packet switched SOC systems [6]. In [5,7,8] authors have been investigated that 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 MPSOC interconnection architecture. Their work illustrated that this type of interconnection networks can offer an attractive alternative solution for SOC systems 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 [7] 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 [7] in the arbitration method, routing algorithm and in the modular design of our switch. Our design also, includes separate set of algorithms for internal switching functionality such as input/output link controller and virtual channel management procedures. Moreover, our design differs also in the way of instantiating the fat tree overall structure.

IV. FAT TREE PRINCIPLE

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 blocks are at the leaves of the tree, and the interior nodes are communication 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. This in turn, will increase the bandwidth at higher levels. Messages are used to communicate between sending and receiving nodes that means construction of paths consisting of some intermediate switches (for routing purposes) from the sources to the destinations. "Fig. 2", shows a butterfly fat tree with 64 IP blocks interconnected by suitable number of switches in intermediate levels. The IP cores 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.

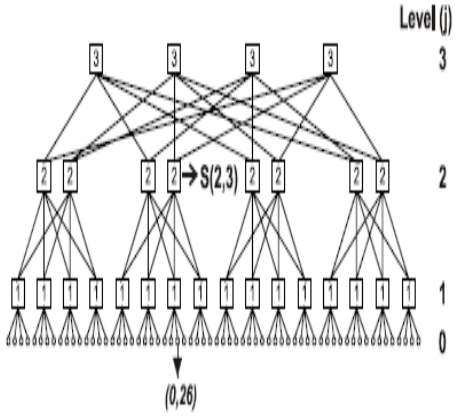

Figure 2. Butterfly fat tree structure with 64 IP cores

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 of $\frac{1}{2^{(l-1)}} \times \frac{N}{4}$ 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 core at the coordinate *(i, 0)* has the parent at coordinate (p,*1*) and $p=i/4$. For example if we have the IP core (62, 0), it has the parent switch (15, 1). Each switch has two parent (p) coordinates ($p1$,$l+1$) and ($p2$,$l+1$)

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

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

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).

V. 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; every port is associated with a pair of opposite unidirectional channels, one for inputs and one for outputs. Fig. 4 illustrates the main components of the switch. Each switch is constructed from the following entities:

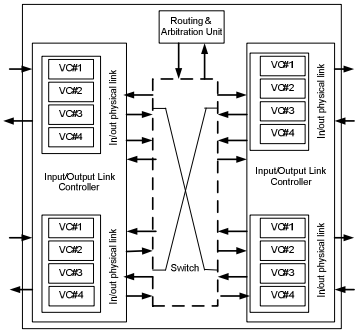

Figure 3. The top-level of the proposed switch structure

A. *Arbitration unit*

The switch includes an arbitration unit that is used to give permission to one of the competing input messages to grant access to the output port. The proposed arbitration unit used in the designed switch employs round-robin strategy with the help of a *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 are 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 in the switch 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 x
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

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. Furthermore, it controls the input buffer which composed of a FIFO memory that are used for storing one or more flits; that are required for storing transferred data until the next channel is available. Hence, virtual channel technique is implemented to avoid deadlock in wormhole switching. Therefore, input buffer unit holds as many buffer objects as many virtual channels are requiring. However, if we speak in more details this unit is responsible for:-

- It checks the availability of free input virtual channel, if exists then it returns free virtual channel number;
- It manages sending out the flit that is available in input virtual channel buffer;
- It helps do routing function by setting the outgoing physical link number that the flit occupying in the virtual channel must follow to reach the destination;
- It keeps track of the outgoing physical link number that is used by flit occupying the virtual channel now;
- It does path setting up using the outgoing virtual channel number for the flit occupying the virtual channel ;
- It sends flit by passing the front flit in the specified input virtual channel on the corresponding input physical link, and.
- It makes buffer management.

The following algorithm is used for moving flits from switch input buffer to switch output buffer and we call it as switching inside algorithm.

```
Input: Selected virtual channel (vc) ,
incoming flit (flit)
Input link port (input port)
Mapping Table Array (mappingtable)
Route Info Array (routeinfo)
Path Info Array (pathinfo)
Output: moving flit from switch input buffer
to switch output buffer
Procedure:
  If vc has flit to send then
    If flit type = HEADER then
     If( routeinfo[vc]>=0) /***Route is
already exist & to find free vc in the output
port**/
outputport=routeinfo[vc]
       outvc=getFreeVC()
/**get free vc in output port***/
      If outvc=>=0 then
         pathinfo[vc]=outvc
         Add flit to output port
         Remove flit from input port
      End if
     Else /*if header flit is not exist in the
route info array and needs to be routed*/
       outputport=find route (destination,
switch index, switch level)
       routeinfo[vc]=outputport
mappingtable[inputport×noOfVirtualLinks+vc]=ou
tputport
      outvc=getFreeVC()
/**get free vc in output port***/
      If outvc=>=0 then
         pathinfo[vc]=outvc
         Add flit to output port
         Remove flit from input port
      End if
   End if
   Else /**if the flit is not header flit**/
outputport=mappingtable[inputport×noOfVirtualL
inks+vc]
      outvc=pathinfo[vc]
      Add flit to output port
      Remove flit from input port
    End if End procedure
```

C. 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 core. *Least common ancestor algorithm* is used in our design as a routing algorithm, as seen below. 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.

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
```

D. 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 IP 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. The detailed actions that are taken by this unit are declared in "Fig. 4".

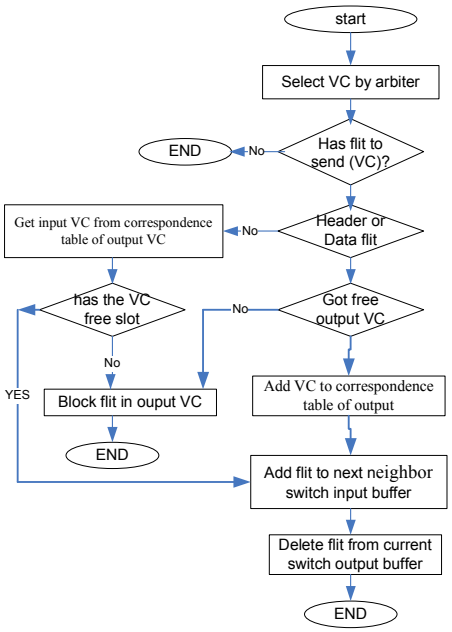

Figure 4. Move flit from switch output buffer to switch input buffer

VI. BUILDING THE FAT TREE MODEL STRUCTURE

The network is the main entity in which all nodes and switches are interconnected to perform the functionality of generating, transferring, routing, switching and receiving of messages between IP blocks of the network (in the form of flits), see "Fig. 6". 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. However, each node has its address, its generated message list to hold the generated messages, and received message list to hold the received messages from different nodes. After generating the messages, an output virtual channel buffer number is assigned to the flits, and then the flits start moving from generated message list to the corresponding output virtual channel in order to be transferred later from the nodes to their parent switch.

B. Packet Generator

The packet generator unit is in charge of generating packets in random lengths and it works in IP node level to generate the random data to pass through the fat tree NOC model. Flit types and their corresponding structures are depicted in "Fig. 5". 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.

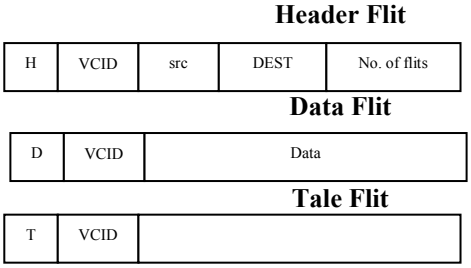

Figure 5. Flit types

VII. RESULTS

A simulator has been developed in C++ to analyze the proposed fat tree NOC architecture. The packet communication flow in the simulator is shown in "Fig. 6". "Fig. 7", 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 every simulation cycle the simulator performs the following:

- move flits from every node to adjacent switches;
- move flits from input buffer to output buffer of the switches;
- move flits from output buffer of the switch to the input buffer of the adjacent switches and/or nodes;
- move flits from the input buffer of the resource node to the received message list of that node;
- 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 that are used in the simulated fat tree structure, traversed switches, and 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 core and switch. In traffic movement we can see many details about certain flits for a particular packet from a specified message such as type of flit, current position of the

flit i.e. all details of switching and routing (switch no., level of the switch, virtual channel no., physical channel no., status of received flit, moving of the flit inside the switch from its input to switch output ports, etc). "Fig. 8", 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 showed 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. 9", 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, because of switching between virtual channels.

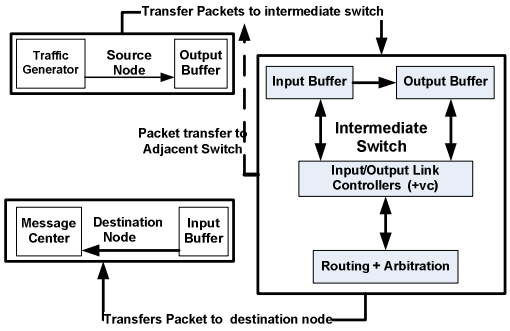

Figure 6. Packet communication flow for the proposed fat tree based NOC system

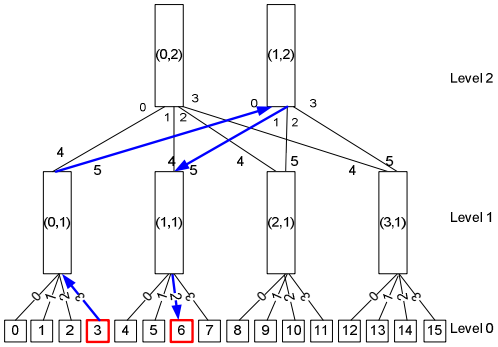

Figure 7. Example: flit (3,6) is routed and switched from input link (5,0) to output link (2,0)

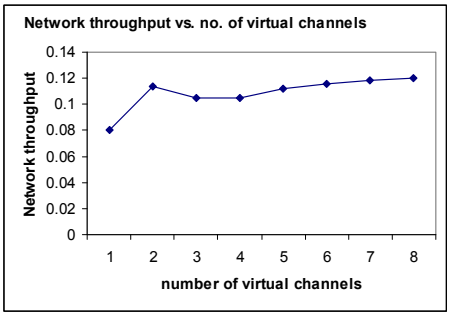

Figure 8. Throughput vs. number of virtual channels

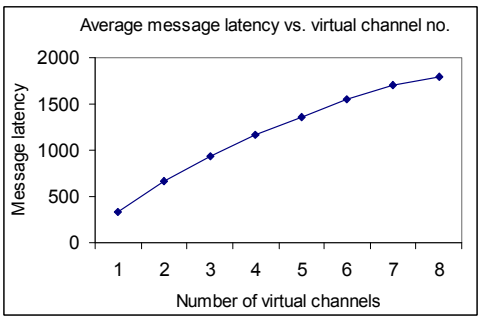

Figure 9. Average message latency vs. virtual channels

VIII. CONCLUSION

The performance of using the developed switch in obtaining efficient NOC interconnection network based on fat tree structure have been demonstrated by simulating the network throughput and the virtual channel effect, buffer utilization and message latency. Also, a complete VHDL code for the proposed algorithms has been written, simulated and partially prototyped in FPGA technology. The simulator could be easily used for education purposes in different computer science courses. Finally, this research is continuing by extending the developed simulator to handle MESH structure (under final testing and comparisons).

REFERENCES

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

[2] Dally, W. J., and B. Towles, "Route packets, not wires: On-chip interconnection networks," *Proc. of the DAC'38 Conference*, Las Vegas, June 2001.

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

[4] P. Magarshack, P.G. Paulin, "System-on-Chip Beyond the Nanometer Wall", *Proceedings of DAC'03,* June 2-6, 2003, Anaheim, USA.

[5] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, "*Design of a Switch for Network on Chip Applications*", Proceedings *of ISCAS*, Bangkok, May 2003 Vol. V, pp. 217-220.

[6] P. Guerrier, A. Greiner**,"**A generic architecture for on-chip packet switched interconnections", Proceedings of DATE"00 Conference, pp. 250 –256.

[7] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, "High-Throughput Switch-Based Interconnect for Future SoCs", Proceedings of 3rd IEEE IWSOC'3, Calgary, Canada.

[8] Cristian Grecu, Partha Pratim Pande, Andre Ivanov, Res Saleh, "*A Scalable Communication-Centric SoC Interconnect Architecture*," IEEE International Symposium on Quality Electronic Design, ISQED 2004, California, USA, 22-24 March, 2004.

[9] C. Leiserson, "Fat trees: Universal Networks for Hardware-Efficient Supercomputing", *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892-901, October 1985.

[10] Asma 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.