Development Of A Flexible Framework For A Network-On-Chip

***Abstract*—We propose a programmable framework for a Network-on-Chip(NoC) based architecture which can be synthesized as a part of SoC. This paper includes the design and implementation of an Integrated Development Environment(IDE) based framework for realization of a NoC subsystem with many interconnected modules communicating with each other through message passing or the establishment of virtual circuits.**

***Index Terms*—Network-on-Chip(NoC), Eclipse based GUI, Routing, Flow Control, Packet Switching, Circuit Switching, Arbitration**

I. Introduction

An Object-based Intelligent Design Assistant Agent (IDAA) with a graphical user interface (GUI) is implemented on a Eclipse based integrated development environment to design and implement a NoC based architecture. It offers the user the flexibility of selecting custom inputs as well as prevent reinvention of well known standard NoC components in the course of the design.

A NoC based architecture [1] is better in terms of bandwidth, scalability and design complexity with an increase in the number of cores in comparison to the traditional bus architecture. A NoC based architecture is conducive to both communication as well as computation. Thus, a near to optimal design of the NoC guarantees crucial attributes such as correctness of communication, efficient and guaranteed routing of packets, freedom from dead-locks and optimal use of silicon resources performing a complex design task by itself that must be repeated for every new silicon design. In terms of regular topologies, NoC based architecture has many types of different dimensions each. In terms of ease of physical mapping and routing patterns, the most common approach is mesh topology with two dimensions using wormhole switching technique [2]. The system modelling is based on several NoC based architecture issues such as *Topology, Flow control, Routing type, Switching Mechanism* to achieve an optimized trade off design based on the users requirement or the specified NoC application. With a flexible implementation of different NoC models, system testing and performance diagnostics are easier and could be made faster.

The rest of the paper is organized as follows. In Section II, we describe the state-of-the-art work in the field of Network-on-Chip based architecture. In Section III, we describe the different levels of implementation in the design of a NoC framework. The top level design is further decomposed into a module level implementation. The following modules are covered in the design - *Flow Control Mechanism, Routing Mechanism, Switching Techniques, Arbitration Protocols*. In Section IV, we have provided a detailed description of the experiments performed with their results. Conclusions are drawn in Section V.

II. Literature Survey

High performance with optimal power consumption to perform various applications required by the user is a major concern of research. Various proposals for a dynamic adaptation of processors, memories and their interconnections are found to provide a more suitable configuration for the system [3]. An interconnection network is a well structured network defined by its topology, routing mechanism, flow control mechanism, type of arbitration and switching technique.Various papers such as fault-tolerant wormhole routing [4], deadlock-free adaptive routing using virtual channels [5], turn around adaptive routing [6] and balanced dimension order routing [7] have been published to design a deadlock free and a minimal adaptive routing technique. With the emerging approach for a NoC based architecture, there is a great enhancement in the performance in terms of power, throughput, latency and speed [8] [9]. A NoC based architecture provides a reliable and a controlled interconnection structure of nodes. The architecture includes various types of topologies like mesh, ring, triangular and honeycomb with regular as well as irregular interconnected networks. It provides a wide range of possible configurations and topologies in the design & automation, for instance, in building SpaceWire based System-on-Chip (SoCWire) communication network [10], performance evaluation [8], encoded signalling [11], wire optimization [12], and programming models [13].

Packets of data passing through the buffers along their routing path in reserved time slots while encountering no contention similar to going through a virtually dedicated circuit is called a Virtual Circuit [14]. Time Division Multiplexing Based Virtual Circuits (TDM VC) techniques developed by Atherial & Nostrum [15] are also used in our paper. It is believed that complex NoC based architectures will allow us to implement complex, heterogeneous sets of applications. Jantsch & Lu [14] propose how Quality of Service(QoS) properties are associated to each subsystem, how the system performance can be statistically analyzed and most importantly, how the impact of the composition on the performance of individual subsystems can be understood and limited. It also mentions a scenario where many applications compete for shared resources and resolves it by a fair allocation policy that gives each application sufficient resources to meet throughput requirements. This idea has been the crux for implementing a user throughput based

slot allocation in multiple VCs. Various approaches have been proposed to present a NoC framework with an improvement in accuracy and speed in an emulation environment platform [13]. The main difference in our approach is the application of wide range of combinations in terms of different topologies with different configurations in a user development environment.

III. Design and Implementation

The top view of the block diagram of a NoC based architecture is a 2-D mesh network which is an interconnection of *NxN* number of nodes. It is shown in the Figure 1. Mesh networks are further classified into *In-direct & Direct networks* [16]. In a mesh network, a given router has four ports and a core processing unit. This is shown in Figure 2. The packets are injected to the router from the block named as ***Stub***. Unlike the in-direct mesh network where the 1:1 mapping between the *Stub* and NoC node is not necessary, our 2-D mesh topology follows the 1:1 mapping. The *Stub* block generates a number of random *Flits* (Refer to section III-B for definition) with each containing a source and a destination address. The data width of a *Flit* is parameterized and the addressing, allocated by the stub, differ with each switching technique. Eventually, the *Flit* size is computed depending on its data width and address width. The configuration of mesh network is made through the GUI according to the input given by the user. The information is then extracted by the router module which instantiates all configurations in terms of flow control, routing, arbitration and switching technique. Finally, the NoC's RTL description in the form of a *Verilog*[1] file is generated by the integrated development environment.

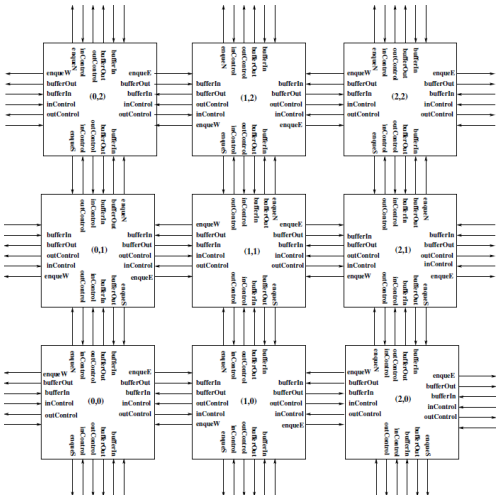

Fig. 1. Mesh Network

A. Graphic User Interface (GUI)

The frontend of the NoC architecture is implemented on the Eclipse IDE framework to develop a Plug-in of type *Multi Page Wizard & Editor*. This allows each user to input

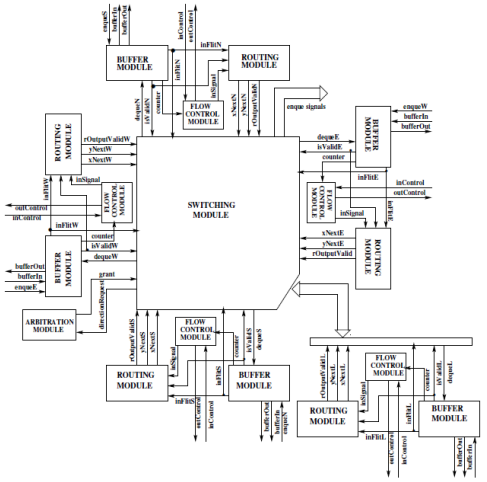

Fig. 2. Router Architecture

settings for the NoC he/she desires. Stage 1 of the plug-in creates a four page wizard to allow the user the flexibility for choosing the required *Switching, Arbitration, Routing and Flow Control Mechanisms*. It also provides the freedom to the user to parameterize elements related to each flit/packet as per the previous settings. Refer to the Figure 3.

Stage 2 of the plug-in makes use of the Open Source Project

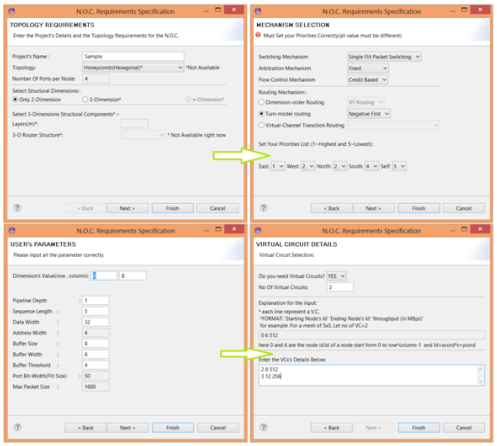

Fig. 3. Stage 1

named *Graphiti* [2] to allow the user to view regular and build custom topologies of the NoC. If regular topology is selected, an auto generated graph for the select NoC is displayed, else the tool provides freedom to the user to make an irregular topology as per his needs. Refer to the Figure 4.

Stage 3 of the plug-in provides a comprehensive amalgamation of the configuration settings and graph based topology of the network. To complete the generation of the required RTL synthesizable files, the 3rd page provides a **Generate NoC** button which is linked to the NoC back-end

[1] http://iverilog.icarus.com/

[2] http://www.eclipse.org/graphiti/

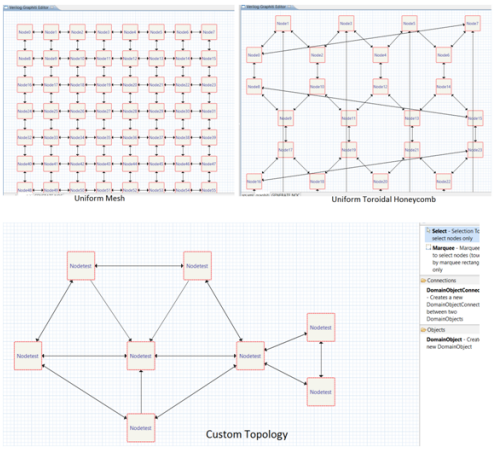

Fig. 4. Stage 2

generator scripts written in *Python*[3] to dynamically generate the required files. Refer to the Figure 5.

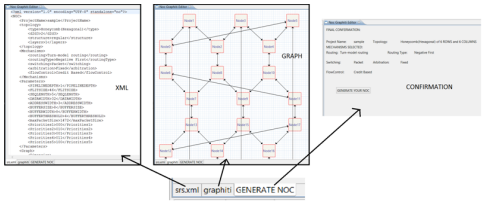

Fig. 5. Stage 3

B. Flow Control

A flow control mechanism determines the allocation of resources to messages as they traverse the network, including buffers and links. There is a significant impact on the *Throughput* and *Latency* of a network through flow control. For 2D tori and meshes, each router has five ports. In each of the ports, one or two Virtual Channels [5] are used. In the first configuration, the buffer size can be optimized according to one's requirement and hence the transfer of packets occurring from one router to another results in the reduction of traffic congestion and eliminates the loss of packets. There is a consistent check on the buffer overflow by the use of a counter. Message sources injected in the form of packets which consist of transfer units known as *Flits*. The *Flits* in the buffer can be accessed by a read and a write pointer for *Dequeue* and *Enqueue* operation respectively, similar to a FIFO mechanism. This can be achieved by two major algorithms namely ,**Go & Stop** [17] and **Credit Based Flow Control** [18].
The second configuration deals with two virtual channels being used in case of Virtual Channel Transition Routing (Refer to section III-C). The transfer of a *Flit* through a channel is selected by the routing mechanism. The output *Flit* departing from a port of the router is selected through the arbitration mechanism (Refer to section III-D).
In order to achieve the specified goals, we introduced parameterized *Verilog* models of flow control in NoC routers. The depth of input buffers, located at the input ports, and the *Flit* size (channel width), are parameterized by the means of *Verilog* generics [12].

C. Routing

Routing mechanism is targeted to be a *k-ary n-cube topology* categorized into deterministic and adaptive algorithms [14]. Deterministic routing algorithms always use the same fixed path for communication between a source-destination pair. Adaptive routing algorithms, on the other hand, can take various conditions into account while determining the path to the destination. Unlike deterministic, adaptability is constrained to area, cost and power consumption. **XY Routing** and **YX Routing** are classified under *Deterministic Routing* approach whereas **Turn Around Routing** and **Virtual Channel Transition Routing** comes under *Adaptive Routing* [5] [6]. **West-first, North-last and Negative-First** are among the few examples of *Turn Around Routing* which prohibits the turns that can cause a deadlock. The cyclic dependencies can be eliminated through the use of virtual channel mechanism.
In each of our implementation of routing algorithm, there exist a declaration of an offset parameter, which is defined as the arithmetic difference between the destination address of the flit and address of the router where the flit is currently present. *xOffset* and *yOffset* are the offset for x, y coordinates respectively. In XY routing, packets must travel along x-direction first to make *xOffset* zero and then starts travelling in y-direction to the point *yOffset* becomes equal to zero. Similar algorithm exist for YX routing, but with order reversed. In case of the *West-First* routing, the flit moves in the west direction if *xOffset* is less than zero at first place and then it can move in any other direction except west. North-Last routing prohibits the north turn until the condition arrives where *xOffset* becomes zero and *yOffset* becomes greater than zero. Negative-First Turn-model Routing restricts two direction turns, *North to West and East to South*.Virtual channels are introduced to improve performance and avoid deadlocks. We have used two virtual channels per physical channel creating two virtual networks under an algorithm named **Balanced Dimension-Order Routing (BDOR)**, which is a simple extension of the dimension-order routing [7].

D. Arbitration

The block diagram architecture of the Arbitration module is shown in Figure 6.In Arbitration, three modules namely *Fixed, Least Recently Used (LRU), Round Robin* have been implemented as a hybrid priority encoder, with the priority list getting updated after each clock cycle by a feedback mechanism. The basic architecture is shown in Figure 6.
Each *Request Array* is a multi bit array with the convention shown in Table I. Each bit, when enabled denotes that the corresponding port has polled a request and its service would

[3]http://www.python.org/

be attended to as per the priority list logic, based on the arbitration protocol being used. Each combinational logic block is dependent on the protocol being serviced to. Refer to Table II for allocation of the grant per algorithm.

L	S	N	W	E
1	**1**	**0**	**0**	**0**
[4]	**[3]**	**[2]**	**[1]**	**[0]**

TABLE I
REQUEST ARRAY CONVENTION

Current priority	Next priority for Fixed	Next priority for LRU	Next priority for Classical RRB	Next priority for Advanced RRB	Grant allocation
West	West	West	East	South	0
East	East	East	Local	North	0
Local	Local	South	South	West	0
South	South	North	North	East	0
North	North	Local	West	Local	1

TABLE II
COMPARISON OF MULTIPLE PROTOCOLS FOR ALLOCATION OF THE GRANT

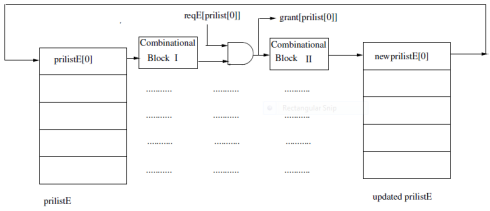

Fig. 6. Block Diagram Architecture for the Arbitration Module

The **Classical Round Robin(RRB)** version [19] of static allocation has been upgraded to prevent static allocate time slots to a specific port to ensure inhibition of wastage of clock cycles; instead it allocates the slot if and only if it polls a request, thus manages traffic unlike the conventional RRB.

The **Weighted Round Robin(WRR)** version aids to the circuit switching module by statically allocating time slots to the ports that share the same buffer present in the path of multiple virtual circuits. The WRR module reads input from files created by a *Python* script which is designed to translate QoS requirements per virtual circuit to a cycle based allocation of the slots on each port of a router. The nomenclature of the files (2 per port per router) allows easy access to the WRR module; One carries the information of the ports requesting slot allocation for a particular destination while the other contains the number of slots to be are allocated for each of the respective ports in the previous file.

E. Switching

The switching mechanism is implemented using a multiplexed crossbar network.The block diagram architecture is shown in the Figure 7. The inputs to the each port of the router comprises of an input flit and a flit validation signal from the flow control mechanism followed by a validation signal corresponding to the coordinates computed by the routing mechanism and a grant signal generated on the basis of the direction request logic from the arbitration mechanism. Each port works in parallel to achieve a crossbar implementation of communication between the native and the immediate neighbouring routers. Based on the grant signals and the input validation signals from flow control & routing modules, the selection logic to a multiplexed network is enabled to generate the respective *Enqueue* and *Dequeue* signals for each port of the router. These are accepted as control signals by the buffers present at each port of the immediate neighbouring routers.

1) Single Flit Packet Switching: For implementing single flit behaviour, each flit carries a unique destination address which is utilized by routing mechanism to compute its immediate next coordinates and are made available as inputs to the switching module. Since 1 packet is equivalent to 1 flit, **Wormhole Switching** [14] is implemented in our design.

2) Multi Flit Packet Switching: For implementing multi flit behaviour, a sequence of 3 types of flits primarily known as *Header*, *Body* and *Tail* are utilized. In the above set sequence, the destination coordinates reside in the *Header* flit. As the *Header* hops across each router of the mesh architecture, it disables a set of signals named *allowHeader* for each port along its path to which it has hopped from. This prevents the entry of another *Header* flit along the same path and makes sure that only *Body* flits and *Tail* flits can traverse along the path used by its corresponding *Header*.

3) Circuit Switching with VC Support: For implementing *Virtual Circuit* [14] behaviour, a set of configuration files are generated per port per router by a *Python* script. These configurations contain a sequence of cycle based allocation for each port of a router to emulate **Time Division Multiplexing** (TDM) [14] at each node of the mesh. The allocation takes place in synchronization with the *Weighted Round Robin* arbitration. Refer section III-D for the algorithm .

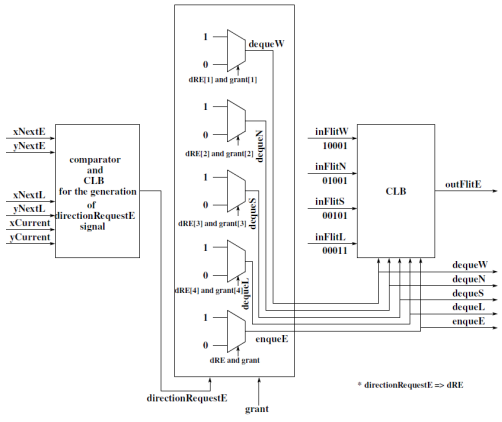

Fig. 7. Switching Architecture

IV. Experiments

Each of the following tests have been done exhaustively i.e individual module testing, randomized module testing, and the system level testing. In randomized module testing, flits with randomly generated destination addresses are injected into the NoC. Correct routing of the each of the flits is verified using *Python* scripts. These scripts also record the average latency and the total number of flits routed within a given simulation time.The system level testing is done as described below -

A. Configurations Implemented

- Configuration 1 - Packet switching technique with go and stop flow(GNS) control mechanism and different routing types including XY routing or YX routing(DR), west-first(WF), north-last, negative first turn routing supported by Round Robin arbitration
- Configuration 2 - Packet switching technique with credit based flow(CRE) control mechanism and different routing types including XY routing or YX routing(DR), west-first(WF), north-last, negative first turn routing supported by LRU and Round Robin arbitration
- Configuration 3 - Circuit switching technique with go and stop flow(GNS) control mechanism and different routing types including XY routing or YX routing(DR), west-first(WF), north-last, negative first turn routing supported by Round Robin arbitration

B. Procedure

- For the first configuration, a flit is routed independent of the path of the flits following it. The flit size is computed depending upon the data width and address width of the flit .The *Stub* considers every flit as a single transfer unit creating a file of the flits sent. A routing algorithm defines the path for each flit keeping the corresponding arbitration protocol and the flow control mechanism into consideration. As the flits are routed to their respective destination, a file is created for the flits received. There is a verification script to verify the received flits which are optimally routed through a distributed minimal routing path with the flits sent from their respective sources.
- For the second configuration, unlike the *Go and Stop* flow control mechanism , a *Credit Based* flow control is used. It uses a multi bit control signal which is utilized for making certain routing decisions. The arbitration performs its respective internal computations to generate a desired priority list in accordance to the testing inputs fed in and thus the flits are routed to their destination successfully.
- For the third configuration, the packet behaves as multi flit data with a *Header*, multiple *Body* and a *Tail* flit, reserving the path as each header hops across the nodes of the network. For verification, the entire set of flits must be received altogether at the destination before being compared with the flits transmitted. The *Stub* assigns 2 bit pointers at the beginning of the flits to distinguish between them correctly.

Note that, **the graph shown below contain data gained from a subset of our experiments with the three configurations, namely different instances of configuration 1 & 2**.

C. Observations & Results

Following the above procedure, parameters such as **Latency/Packet** and **NoC Throughput** are calculated and are compared to analyze the performance of the system in the above configurations. The total time of simulation is fixed to **70 clock cycles**. The traffic analysis for the above configurations is divided into *Non-Uniform Traffic Pattern* and *Uniform Traffic Pattern*.

- The testing for the fixed traffic pattern i.e
  $xDest$[4] = (($xSource$[5]+2)% N[6])
  $yDest$[7] = (($ySource$[8]+2)% N)
  is done such that the packets being injected to the maximum possible extent from all *Stubs* which depicts a high congestion scenario. The graphs in Figure 8 & 9 clearly show that, for a non-uniform (often realistic) traffic pattern, an adaptive algorithm (in this case *West First*) when provided with more information flow through the use of credit-based flow control, provides the *Lowest Latency/Packet* and *Highest NoC Throughput*.
- The testing for the uniform traffic pattern i.e *xDest,yDest,xSource,ySource* take values depending on the mesh size, is done similarly. The graphs in Figure 10 & 11 clearly show that, for a uniform traffic pattern, deterministic routing outperforms any adaptive routing. This is in concordant with the work of Hu & Marculescu [20].

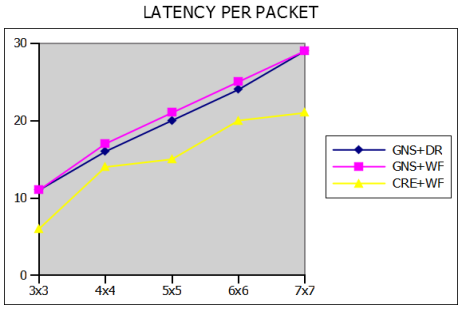

Fig. 8. Latency/Packet versus Mesh Size for Non-Uniform Traffic Pattern

V. Conclusion and Future Works

The **IDAA** offers the user the flexibility to customize the 4 major elements of any NoC design i.e *Flow Control Mechanisms, Routing Mechanisms, Switching and Arbitration Mechanisms* simplistically. Its user friendly GUI coupled with the *Graphiti* based support provides a complete development framework for getting a wide variety of mesh-based NoCs at

[4]X Destination Coordinate
[5]X Source Coordinate
[6]Number of rows/column
[7]Y Destination Coordinate
[8]Y Source Coordinate

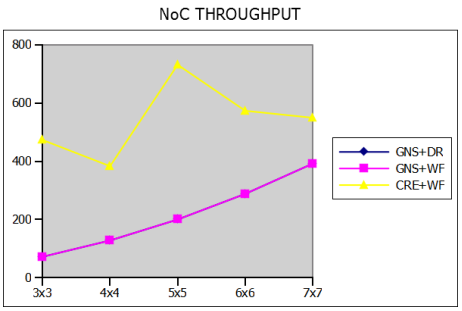

Fig. 9. NoC Throughput versus Mesh Size for Non-Uniform Traffic Pattern

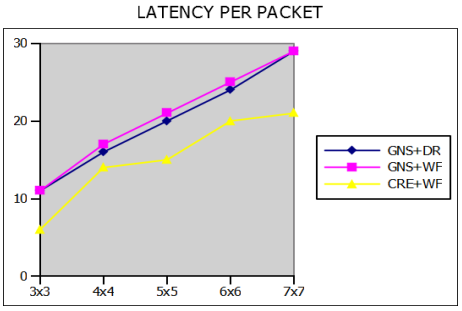

Fig. 10. Latency/Packet versus Mesh Size for Uniform Traffic Pattern

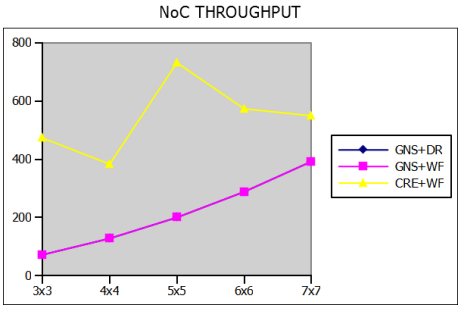

Fig. 11. NoC Throughput versus Mesh Size for Uniform Traffic Pattern

the touch of a button. Also, the support of virtual circuits each supported with QoS is one of the highlights of our framework. However, a deeper understanding of its proper functionality & exhaustive testing is a part of our future work. Following the testing of the three configurations listed above, it is clear that our implementation emulates the theortical understanding of the subject well. There is definitely a room for improvement towards analysis dealing with new parameters and larger mesh size.

REFERENCES

[1] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johny berg, Kari Tiensyrj and Ahmed Hemani, "A Network on Chip Architecture and Design Methodology", *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*,2002

[2] K. M. Al-Tawil, M. Abd-El-Barr, and F. Ashraf, "A survey and comparison of wormhole routing techniques in mesh networks",*IEEE Network*, vol. 11, pp. 3845, 1997.

[3] Paulo C. Santos, Gabriel L. Nazar, Fakhar Anjam, Stephan Wong, Dbora Matos, Luigi Carro"A Fully Dynamic Reconfigurable NoC-based MP-SoC: The Advantages of Total Reconfiguration",Springer-Verlag Berlin Heidelberg 2011

[4] Zhen Zhang,Alain Greiner,Sami Taktak,"A Reconfigurable Routing Algorithm for a Fault-Tolerant 2D-Mesh Network-on-Chip",*45th ACM/IEEE Design Automation Conference*, May 2008

[5] William J. Dally and Hiromichi Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels",*IEEE transactions on Parallel and Distributed*, VOL. 4, NO. 4, APRIL 1993

[6] Christopher J. Glass and Lionel M. Ni, "The Turn Model for Adaptive Routing", *Proceedings of the 19th annual international symposium on Computer architecture*, Volume 20 Issue 2, May 1992

[7] Jose Miguel Montaana, Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano,"Balanced Dimension-Order Routing for k-ary n-cubes",*International Conference on Parallel Processing Workshops*, Sept. 2009

[8] Partha Pratim Pande,Cristian Grecu, Michael Jones, Andre Ivanov,Resve Saleh"Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect", *IEEE transaction on Computers*, vol. 54, No. 8, August 2005.

[9] Mridula Agarwal,Nitin Jain,Rupesh Dubey,"Performance Enhancement w.r.t. Time in Hetrogeneous Network on Chip Architecture", *International Journal of Latest Trends in Engineering and Technology*,Vol. 2 Issue 3 May 2013

[10] Sami HACHED, Mohamed GRAJA, Slim BEN SAOUD"Design and Implementation of NoC architectures based on the SpaceWire protocol", *International Journal of Advanced Computer Science and Applications*,Vol. 2, No.2, February 2011

[11] D'Alessandro C.S., Delong Shang, Bystrov A., Yakovlev A.V.,"Phase-Encoding for On-Chip Signalling",*IEEE Transactions on Circuits and Systems I: Regular Papers*,Volume:55,Issue: 2,March 2008

[12] Ying-Cherng Lan, Michael C. Chen, Alan P. Su, Yu-Hen Hu, Sao-Jie Chen"Flow Maximization for NoC Routing Algorithms",*Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI*

[13] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, F. Catthoor, "A Complete Network-On-Chip Emulation Framework"*Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, IEEE Computer Society*, 2005

[14] Fayez Gebali, Haytham Elmiligi and Mohamed Watheq El-Kharashi,"Networks on chips - Theory and Practice",CRC Press, Inc.,2009, ISBN:1420079786 9781420079784

[15] Goossens K., Dielissen J., Radulescu A.,"thereal network on chip: Concepts, architectures and implementations", *IEEE Design and Test of Computers*, vol 22, 2005.

[16] , Srinivasan Murali, Giovanni De Micheli,"SUNMAP: a tool for automatic topology selection and generation for NoCs",*Proceedings of the 41st annual Design Automation Conference, 2004*, pp 914-919, ISBN:1-58113-828-8, DOI: 10.1145/996566.996809

[17] S. Jamaloddin GoIestani , "A Stop-and-Go Queueing Framework for Congestion Management", *ACM SIGCOMM Computer Communication Review*, 1990

[18] Martijn Coenen, Srinivasan Murali, Giovanni De Micheli, Andrei Radulescu & Kees Goossen*Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*

[19] Preeti Bajaj and Dinesh Padole (2011),"Arbitration Schemes for Multiprocessor Shared Bus, New Trends and Developments in Automotive System Engineering", Prof. Marcello Chiaberge (Ed.), ISBN: 978-953-307-517-4, InTech

[20] Jingcao Hu and Radu Marculescu,"DyAD - Smart Routing for Networks-on-Chip",*In ACM/IEEE Design Automation Conference*, 2004, pp 260–263