



**DESIGN OF A CONFIGURABLE 3D NETWORK ON
CHIP BASED ON THE DIRECT ELEVATOR 3D
ROUTING ALGORITHM**

By

Maha Ramadan Mohamed Beheiry

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications

FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
OCTOBER 2017

DESIGN OF A CONFIGURABLE 3D NETWORK ON CHIP BASED ON THE DIRECT ELEVATOR 3D ROUTING ALGORITHM

By

Maha Ramadan Mohamed Beheiry

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications

Under the Supervision of

Prof. Ahmed M. Soliman

Emeritus Professor

Electronics and Communications Department

Faculty of Engineering , Cairo University

Dr. Hassan Mostafa

Assistant Professor

Electronics and Communications Dept.

Faculty of Engineering , Cairo University

**FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
OCTOBER 2017**

**DESIGN OF A CONFIGURABLE 3D NETWORK ON
CHIP BASED ON THE DIRECT ELEVATOR 3D
ROUTING ALGORITHM**

By

Maha Ramadan Mohamed Beheiry

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications

Approved by the Examining Committee:

Prof. First S. Name, External Examiner

Prof. Second S. Name, Internal Examiner

Prof. Ahmed M. Soliman, Thesis Main Advisor

FACULTY OF ENGINEERING , CAIRO UNIVERSITY
GIZA, EGYPT
OCTOBER 2017

Engineer's Name: Maha Ramadan Mohamed Beheiry
Date of Birth: 25/6/1990
Nationality: Egyptian
E-mail: mahabehiry@gmail.com
Phone: 01064479725
Address:
Registration Date: 20/9/2012
Awarding Date: --/2017
Degree: Master of Science
Department: Electronics and Communications



Supervisors:

Prof. Ahmed M. Soliman
Dr. Hassan Mostafa

Examiners:

Prof. First S. Name	(External examiner)
Prof. Second S. Name	(Internal examiner)
Prof. Ahmed M. Soliman	(Thesis main advisor)

Title of Thesis:

Design of a Configurable 3D Network on Chip Based on the Direct Elevator
3D Routing Algorithm

Key Words:

3D technology; 3D routing algorithm; SoCs; 3D-NoCs tool; 3D-NoCs generator.

Summary:

The main goal of the thesis is to provide the designers with a tool to create different configurations of the Three-Dimensional Network-On-Chips. These different configurational Three Dimensional (3D) Network-On-Chips (NoCs) will be then evaluated to determine which configuration is the best for a specific design or application. The 3D-NoCs are implemented based on a 3D routing algorithm denoted by Direct Elevator algorithm.

Table of Contents

List of Tables	iv
List of Figures	v
Acknowledgements	viii
Dedication	ix
Abstract	x
1 Introduction	1
1.1 Motivation	1
1.2 Contribution	2
1.3 Organization of The Thesis	3
2 Literature Review	4
2.1 3D Routing Algorithms	4
2.1.1 LA-XYZ: A high throughput look-ahead routing algorithm	4
2.1.2 Topology Aware Adaptive Routing for irregular Mesh 3D-NoC systems	10
2.1.3 Layered Routing in Irregular Networks	12
2.1.4 Elevator-First Routing Algorithm	13
2.2 Simulators and Tools to Create Various 3D-NoCs	18
2.2.1 Booksim Simulator	18
2.2.2 Noxim Simulator	21
2.2.3 NIRGAM: A simulator tool for NoC interconnect routing	22
2.2.4 CONNECT tool to create different 2D-NoCs	23
3 An Overview About The Three Dimensional Network-On-Chips	28
3.1 Three Dimensional Technology	28
3.1.1 3D Stacking Technologies	28
3.1.2 Vertical Interconnects Technologies	29
3.2 Different Implementations and Applications with 3D Technology	30
3.2.1 3D Chip Structure with an iA32 Microprocessor	31
3.2.2 CMOS Based Image Sensors Using 3D Technology	31
3.3 Challenges of the 3D Integration Technology	32
3.4 Network-On-Chips	33
3.5 The Architecture of NoCs	34
3.5.1 Router Design	34
3.5.2 Network Design	35
3.6 Partial Reconfiguration Using NoCs	35
3.7 The Three Dimensional Network-On-Chips	36
3.7.1 The Architecture os 3D-NoCs	36

4	Introduction To The Network-On-Chip Methodology in FPGAs	38
4.0.1	A comparison between the soft NoC and hard NoC on FPGA . . .	38
4.0.1.1	Router Architecture in the NoC hard implementations . . .	39
4.0.2	Area Results	40
4.0.3	Power Results	41
4.1	Impact of NoC parameters on the FPGA Network-On-Chips (NoCs) . . .	42
4.2	A NoC design with debug features on Field-Programmable Gate Array (FPGA)	42
5	The Direct Elevator Three Dimensional Routing Algorithm	45
5.1	Introduction	45
5.2	Elevator-First Three Dimensional Routing Algorithm	47
5.3	The Direct-Elevator Three Dimensional Routing Algorithm	48
5.3.1	A packet routing path using Elevator-First mechanism	48
5.3.2	A packet routing path using Direct-Elevator mechanism	49
5.4	Comparative Performance Analysis	50
5.4.1	Network Load	51
5.4.2	Vertical Complexity	52
5.4.3	Tier Complexity	53
5.4.3.1	Regular Distributed 3D-NoC	53
5.4.3.2	Hierarchical Distributed 3D-NoC	55
5.5	Summary	55
6	3D-NOCET: A Tool for Implementing 3D-NoCs based on Direct-Elevator Algorithm	56
6.1	Introduction	56
6.2	The 3D-NOCET Tool User Guide	58
6.2.1	Automation Infrastructure Scripts	58
6.2.2	Register Transfer Level Design files	59
6.2.3	The 3D-NOCET tool implementation and execution flow	60
6.2.4	How to use the 3D-NOCET tool suite	81
7	Discussion on A Comparative and Performance Study for Different Structures of 3D-NoCs	82
7.1	Introduction	82
7.2	Comparative Results Analysis	82
7.2.1	Vertical Network Complexity	82
7.2.1.1	Impact of Vertical Complexity on The Latency	82
7.2.1.2	Impact of Vertical Complexity on The Power	83
7.2.1.3	Impact of Vertical Complexity on The Area	84
7.2.2	Tier Network Complexity	85
7.2.2.1	Impact of Tier Complexity on The Latency	86
7.2.2.2	Impact of Tier Complexity on The Power	86
7.2.2.3	Impact of Tier Complexity on The Area	87
7.3	Work Conclusion	89
7.4	Future Work	89

References	90
Appendix A 3D-NOCET Tool Source Files	93

List of Tables

2.1	Some Parameters of the Booksim Simulator (Available in both versions of Booksim)[1]	20
2.2	Different NoC simulator and synthesizers[2]	26
2.3	A comparison between different simulators[2]	27
2.4	Nomenclature of the comparison table[2]	27
3.1	Comparison between different stacking techniques[3]	29
3.2	Comparison of vertical interconnect technologies[4]	29
4.1	The evaluated hard NoC implementations[5]	38
4.2	Router areas[5]	40

List of Figures

1.1	Moore's law effect[6]	1
2.1	Conventional XYZ routing algorithm[7]	5
2.2	LA-XYZ routing algorithm flow[7]	5
2.3	Step 1: LA-XYZ routing algorithm[7]	6
2.4	Step 2: LA-XYZ routing algorithm[7]	8
2.5	By-pass in LA-XYZ routing algorithm[7]	9
2.6	The architecture of the 3D-NoC's router[7]	9
2.7	The module of input-port[7]	10
2.8	Larger path variety in TAAR routing algorithm[8]	11
2.9	Operation flowchart of a transport layer and a network layer[8]	11
2.10	(a) Packet B is waiting for Packet A, (b) Packet A is prepared to retransmitted, and (c) Packet B start to enter the temporal storage[8]	11
2.11	(a) Operation flow of TAMRA, and (b) An example[8]	12
2.12	Vertically partially connected 3D-NoC[9]	13
2.13	Different routers in Vertically partially connected 3D-NoC systems[9]	14
2.14	X-First 2D and Elevator-First Routing Algorithms[9]	15
2.15	Elevator First Routing in LOCAL input ports[9]	16
2.16	Elevator First Routing in 2D input ports[9]	17
2.17	Different routers in Vertically partially connected 3D-NoC systems[9]	18
2.18	Module hierarchy of the simulator[1]	19
2.19	Top-level block diagram of the simulator[1]	19
2.20	Router module of Booksim tool[1]	20
2.21	Internal architecture of hub nodes with wireless communication[10]	21
2.22	NIRGAM Simulator Flow Diagram[11]	22
2.23	CONNECT Router Architecture[12]	23
2.24	CONNECT Tool[12]	24
2.25	Comparison between FPGA cost of SOTA and CONNECT router[12]	25
3.1	Illustration of vertical interconnect technologies: wire bonded (a); microbump3D package (b) and face-to-face (c); contactlesscapacitive with buried bumps (d) and inductive (e); through viabulk (f) and silicon on insulator (g)[4]	30
3.2	3D Structure[13]	31
3.3	A cross-section of an imager die schematic[14]	32
3.4	SAM image[14]	32
3.5	Cooling used over a 3D structure[15]	33
3.6	Example of a NoC[16]	34
3.7	Two Inputs/Two Outputs router Implementation[17]	35
3.8	A NoC DPR Architecture[18]	36
3.9	3D Mesh based network[19]	37
3.10	Fat Tree Model[20]	37
4.1	Three NoC implementations[5]	40

4.2	Percentage of router area to chip area[21]	41
4.3	Hard NoC and Soft NoC power consumption[21]	41
4.4	SonicsGN NoC Architecture [22]	42
4.5	The location of the performance tool in the SonicsGN NoC[22]	43
4.6	The Architecture of the performance and monitoring tool[22]	44
5.1	TSV interconnect with landing pad[23]	45
5.2	TSVA cell occupying three standard cell rows (KOZ = 1.205 m) and TSVB cell occupying four standard cell rows (KOZ = 2.44 m)[24]	46
5.3	Elevator-First 3D Router[9]	47
5.4	Elevator-First routing mechanism[25]	49
5.5	Direct-Elevator routing mechanism[25]	50
5.6	Throughput Vs. Packets (random test case)[25]	51
5.7	Throughput Vs. Packets (worst path test case)[25]	51
5.8	Throughput Vs. Tiers (random test case)[25]	52
5.9	Throughput Vs. Tiers (worst path test case)[25]	53
5.10	Throughput Vs. Routers/Tier (random test case)[25]	54
5.11	Throughput Vs. Routers/Tier (worst path test case)[25]	54
5.12	Hierarchical Structure of a 3D-NoC example	55
6.1	The Graphical User interface of the 3D-NOCET tool	57
6.2	The 3D-NoC in 2D Mesh Topology	58
6.3	The 3D-NoC in 2D Ring Topology	58
6.4	The 3D-NOCET tool final information message	59
6.5	The flow chart of the 3D-NOCET tool implementation	60
6.6	The main GUI window structure and the different valid topologies	61
6.7	The implementation of the functionality buttons	61
6.8	Selection of the topology type	62
6.9	Evaluating the parameters in the header file	62
6.10	Determining the script flow according to the topology type	63
6.11	Evaluating the number of wire in the 3D-NoC system using mesh topology	64
6.12	Creating the connections between the routers in the mesh topology	65
6.13	The instantiaions of the mesh routers	66
6.14	Adding the mesh routers' instantiaions into the SystemVerilog top design file	66
6.15	Creating address for each router in mesh topology	67
6.16	Creating the routing tables in mesh topology	68
6.17	The router address	69
6.18	Evaluating the number of wire in the 3D-NoC system using ring topology	69
6.19	Creating the connections between the routers in the ring topology	70
6.20	The instantiaions of the ring routers	71
6.21	Adding the ring routers instantiaions into the SystemVerilog top design file	71
6.22	Creating address for each router in ring topology	72
6.23	The router instantiations in the top module design	73
6.24	The data in the routing table	73
6.25	The global parameters of the 3D-NoC system	73
6.26	The testbench for simulations	74
6.27	The parameters of the mesh router module	75

6.28	The implementation of the 2D routing with mesh topology	76
6.29	The implementation of the Direct-Elevator algorithm in the 3D mesh router module	77
6.30	The parameters of the mesh router module	78
6.31	The implementation of the 2D routing with ring topology	79
6.32	The implementation of the Direct-Elevator algorithm in the ring router module	79
6.33	The final information message	80
6.34	Design_files directory	81
7.1	Latency in Clock Cycles Vs. Number of Tiers in NoC	83
7.2	Power in Watts Vs. Number of Tiers in NoC	84
7.3	Number of Luts Vs. Number of Tiers in NoC	85
7.4	Number of FlipFlops Vs. Number of Tiers in NoC	85
7.5	Latency in Clock Cycles Vs. Number of Routers in Tier	86
7.6	Power in Watts Vs. Number of Routers in Tier	87
7.7	Number of Luts Vs. Number of Routers in Tier	88
7.8	Number of FlipFlops Vs. Number of Routers in Tier	88

Acknowledgements

I would like first to thank my thesis supervisor Dr. Hassan Mostafa and Prof. Ahmed Soliman for their continuous support, patience and motivation. Besides my advisors, I would like to express my sincere gratitude to my awesome manager Rasha El Atfy for her generous support and encouragement. I would also like to thank my amazing workmates who helped me alot with their immense knowledge Hanan Moharam, Hager Fathy, Ali Mahgoub, Ahmed Emara and Rami Ahmed. Also, I am so grateful to Mennatallah Amer and Ahmed Maher for reviewing most of my work and give me insigthful comments. Furthermore, My sincere thanks goes to my husband Ahmed Aly for helping me with his knowledge to implement and develop many ideas into my thesis and supporting me to finish this. Finally, I would like to thank my mother for her spiritual support throughout my life.

Dedication

This work is dedicated to my lovely parents, to my wonderful brothers, and my beloved husband who always keeps pushing me out of all my comfort zones.

Abstract

Three Dimensional integration technology offers a huge opportunity to implement different powerful applications. Moreover, the Network-On-Chip methodology solves the new challenges with the long wiring with new communication approach between the different nodes in one chip. Combining the 3D integration technology and the NoCs leads to such an attractive solution for many applications that have been impossible before. The 3D-NoCs come with variety of research challenges in maintaining the performance efficiency, the vertical interconnections design and placement, and manufacturing process steps. The routing challenge is one of the most vital challenges in implementing the 3D-NoCs. Therefore, implementing and developing a flexible and reliable routing algorithm to communicate between the different 3D stacked tiers is essential. As a part of this thesis work, a 3D routing algorithm, Direct-Elevator, based on the Elevator-First 3D routing algorithm is proposed. The Direct-Elevator algorithm is independent on the number of the number of interconnects, placement of interconnects and the planar network topology. The Direct-Elevator is tailored for the 3D-NoC different structures. It offers a lower communication latency approach in comparison to Elevator-First routing algorithm.

A new tool is proposed in this thesis to provide a solution for the implementation of generic Three Dimensional (3D) Network-on-Chips (NoCs) to serve different applications and designs. The proposed tool denoted by 3D-NOCET, is based on the 3D Direct-Elevator routing algorithm. The 3D-NOCET tool allows the user to create different combinations of 3D-NoCs based on the 2D-routing topology, number of tiers and number of routers per each tier through a fully automation infrastructure. That paves the road to perform diverse experimental evaluations for the different 3D-NoC structures. The future experimental evaluations and the performance comparative analyses help to find the optimal network configuration for different applications.

Chapter 1: Introduction

1.1 Motivation

The Three Dimensional (3D) technology becomes very essential in overcoming the new design challenges. As the size of the transistor keeps shrinking, Moore's law continues to yield high transistor density with each new generation[26] as shown in Figure 1.1. The 3D technology enables the designer to stack and fabricate multiple dies in one chip. The 3D technology is such a powerful solution, but it comes with many other research challenges such as the thermal effect, the parastic capacitance between the vertical links and the floor planning techniques.

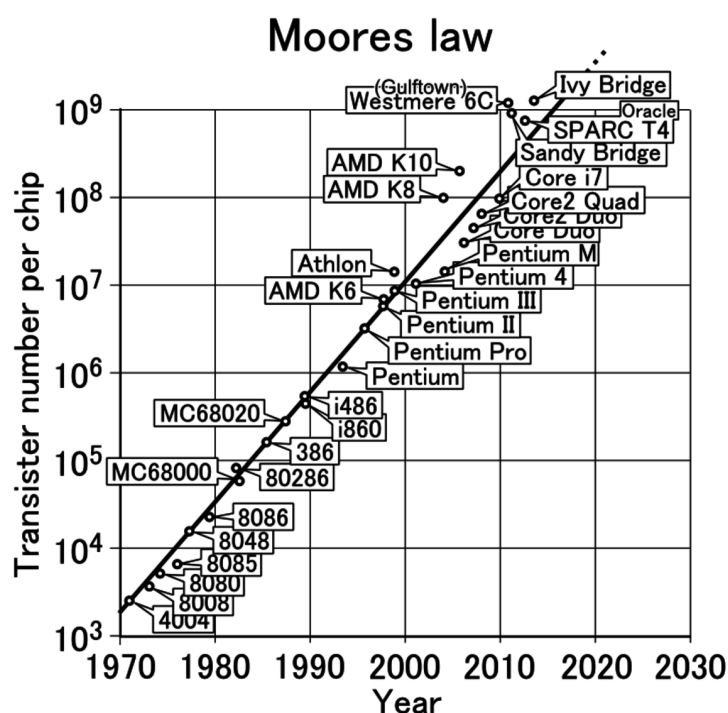


Figure 1.1: Moore's law effect[6]

On the other hand, The Network-On-Chip methodology offers vital solutions to the new wiring challenges in the chip fabrication process. The physical wiring connect the different cores in the System-On-Chip (SoC). The wires become thicker and longer in the top level of the design, which leads at last to very lossy global wiring[27]. The Network-On-Chip (NoC) technique introduces a new a new approach to communicate between the different cores in one chip. The NoC depends on different routers which communicate together to transfer the data from different sources to different destinations on the chip. The NoC design depends on different metrics such as: the router design, the network topology, the switching technique and other factors.

The NoC in the Field-Programmable Gate Array (FPGA) methodology is widely used in many applications. The NoCs in FPGA is able to enhance the system efficiency, increase the design production and reduce the system compilation time. The NoCs on FPGAs open the door to many innovative approaches such as the parial dynamic reconfiguration (PDR), in which the user can reconfigure a part of the FPGA while other parts are operating normally.

The combination between the 3D and the NoC approaches yields into a very powerful solution for many design challenges as illustrated above. The Three-Dimensional Network-On-Chips (3D-NoCs) have innumerous configurations based on many factors such as the network topology, the number of vertical stacked tiers, the number of routers per each tier and the switching techniques. The ability to create different configuration of the 3D-NoCs paves the road to the designers to explore the optimal configural 3D-NoC design that suits the most their applications. Moreover, creating these different configural 3D-NoCs using automated solution saves a lot of time and certainly gives a wide flexibility for performing many experiments. That is the main motivation behind this thesis work which is providing the designers with a powerful tool to create synthesizable 3D-NoCs.

1.2 Contribution

This work includes the following contribution:

- Implementing a 3D routing algorithm denoted by Direct-Elevator based on Elevator-First routing algorithm. The routing algorithm Direct-Elevator is used to route the packets vertically between the source and destination nodes. It is proved that it offers a lower communication latency than the Elevator-First algorithm.
- The two routing algorithms Elevator-First and Direct-Elevator are implemented using C programming language and a comparison study had been conduced to compare between the two algorithms.
- Creating a new tool denoted by 3D-NOCET to create different configural synthesizable 3D-NoCs based on the number of tiers in the system, the number of routers in each tier and the network planar topology.
- An analysis study has been conducted to evaluate different configurations for the 3D-NoCs and study the effect while varying the system aspects which are the number of tiers, the number of routers per each tier and the planar topology.
- A simple Graphical User Interface has been implemented for the tool to ease the usage and also to elaborate more on how the tool works.

1.3 Organization of The Thesis

This thesis is organized as follows. Chapter 2 contains a survey of the most significant previous work. The survey includes details different implemented 3D routing mechanisms and variety of distinct developed simulator and tools which are used to create 3D-NoCs and 2D-NoCs. Chapter 3 provides a detailed overview about the 3D-NoCs. In which, an overview about the 3D technology is presented followed by a brief about the NoC methodology. Then, a deep elaboration on the powerful combination between the 3D technology and NoC methodology is detailed.

Chapter 4 includes a specific introduction to the NoC on FPGAs. The introduction highlights the benefits using the NoC methodology on the FPGA and also the new challenges in designing the NoC for FPGAs. While in Chapter 5, the Direct-Elevator 3D routing mechanism is illustrated. The illustration spots the main differences between the two routing algorithms Elevator-First and Direct-Elevator. Moreover, a detailed comparison study is presented to evaluate the two routing algorithm with respect to the system total latency while changing the number of transmitted packets, the number of routers per each tier and the number of tiers in the system.

Chapter 6 presents the 3D-NOCET tool through a detailed user's guide that illustrates how the tool works exactly. Eventually, Chapter 7 contains a comparative and performance study on different configurations for the 3D-NoCs. The study illustrates the effect of the vertical network and tier network complexity on the 3D-NoC design. Moreover, it concludes the whole thesis work in a conclusion section and also the future work that can be done.

Chapter 2: Literature Review

2.1 3D Routing Algorithms

Many research work has been conducted to solve the challenges appear while using the 3D-NoCs. One of these challenges is developing and implementing a 3D routing algorithm to route the packets through the 3D stacked chips. The main target of the developed routing mechanisms is to keep the flexibility and utilization provided by the 3D-NoCs.

2.1.1 LA-XYZ: A high throughput look-ahead routing algorithm

This routing algorithm is developed to minimize the system communication latency and power consumption, also to improve the system overall throughput. The 3D-NoC systems mainly depend on the Dimension Order Routing (DOR) XYZ algorithm. The routing algorithm routes the first packet flit to the X dimension, then to the Y and eventually to the Z dimension to reach its target. The output port is determined in this algorithm by comparing the processing node with the address of the final destination node [7]. Despite the fact that the routing algorithm is simple to be implemented and also free of livelock and deadlock, but it has problems with inefficiency in the pipelining.

Figure 2.1 shows a router pipeline based on the XYZ routing in which the virtual channels are not considered to enhance the performance of the routing algorithm. The router pipeline in Figure 2.1 is consisted of four pipelining as follows:

- Buffer Writing (BW).
- Routing Calculation (RC).
- Switch Arbitration (SA),
- Crossbar Traversal stage (CT).

The router introduces a latency overhead and power consumption as the packet needs to pass by all these four stages to reach its final destination. That will cause a very large decay in the performance of the whole system, also this problem is related proportionally to the network size.

To solve this problem, the authors in this work proposed a Look-Ahead-XYZ routing algorithm which is denoted by LA-XYZ [7]. In this routing algorithm, a precomputation to the next port direction of the router is done then adds this information to the packet.

When the packet arrives at the downstream node, the switch arbiter will use the next port identifier added to the packet to route the packet to the neighbour node. At the same time the routing calculation works in parallel to determine the next output port that the packet will take. This parallel procedure optimizes the pipelining time in the LA-XYZ routing algorithm as shown in Figure 2.2.

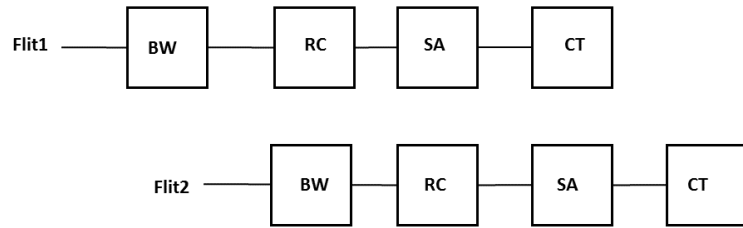


Figure 2.1: Conventional XYZ routing algorithm[7]

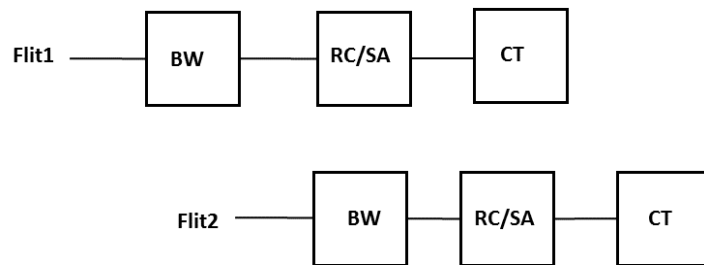


Figure 2.2: LA-XYZ routing algorithm flow[7]

The LA-XYZ routing algorithm mainly takes two steps to route the packets as follows:

1. Extracting the information of the next output port which is embedded in the packet. According to the direction of the output port, the address of the next node is calculated to be used in the next step. This step is illustrated in Figure 2.3.

Algorithm 1: LA-XYZ 1st phase: Assign next address

```
// Current node address
Input:  $X_{cur}, Y_{cur}, Z_{cur}$ 
// Next port identifier
Input: Next-port
// Next node address
Output:  $X_{next}, Y_{next}, Z_{next}$ 

// Evaluate Next node x_address
if (Next-port is EAST) then
|  $X_{next} \leftarrow X_{cur} + 1;$ 
else
| if (Next-port is WEST) then
| |  $X_{next} \leftarrow X_{cur} - 1;$ 
| else  $X_{next} \leftarrow X_{cur};$ 
end

// Evaluate Next node y_address
if (Next-port is NORTH) then
|  $Y_{next} \leftarrow Y_{cur} + 1;$ 
else
| if (Next-port is SOUTH) then
| |  $Y_{next} \leftarrow Y_{cur} - 1;$ 
| else  $Y_{next} \leftarrow Y_{cur};$ 
end

// Evaluate Next node z_address
if (Next-port is UP) then
|  $Z_{next} \leftarrow Z_{cur} + 1;$ 
else
| if (Next-port is DOWN) then
| |  $Z_{next} \leftarrow Z_{cur} - 1;$ 
| else  $Z_{next} \leftarrow Z_{cur};$ 
end
```

Figure 2.3: Step 1: LA-XYZ routing algorithm[7]

2. A comparison takes place between the current processing node and the destination address to know if the packet reaches its target or not. This step is illustrated in Figure 2.4.

Algorithm 2: LA-XYZ 2nd phase: Define new Next-port

```
// Destination address
Input:  $X_{dest}, Y_{dest}, Z_{dest}$ 
// Next node address
Input:  $X_{next}, Y_{next}, Z_{next}$ 
// New next port for next node
Output: New-next-port

if ( $X_{next}$  is equal to  $X_{dest}$ ) then
  if ( $Y_{next}$  is equal to  $Y_{dest}$ ) then
    if ( $Z_{next}$  is equal to  $Z_{dest}$ ) then
      |  $New\_next\_port \leftarrow$  LOCAL;
    else
      | if ( $Z_{next}$  is smaller than  $Z_{dest}$ ) then
      | |  $New\_next\_port \leftarrow$  UP;
      | else  $New\_next\_port \leftarrow$  DOWN;
    end
  end
  else
    | if ( $Y_{next}$  is smaller than  $Y_{dest}$ ) then
    | |  $New\_next\_port \leftarrow$  NORTH;
    | else  $New\_next\_port \leftarrow$  SOUTH;
  end
end
else
  | if ( $X_{next}$  is smaller than  $X_{dest}$ ) then
  | |  $New\_next\_port \leftarrow$  EAST;
  | else  $New\_next\_port \leftarrow$  WEST;
end
```

Figure 2.4: Step 2: LA-XYZ routing algorithm[7]

As an optimization approach in the LA-XYZ routing algorithm, a no-load bypass approach is taken as shown in Figure 2.5. The number of the pipelining stages can be optimized by overlapping the buffer writing stage. In this approach, if the input FIFO buffer is empty, the packet does not need to be stored and it will continue its path straight to the routing calculation and switch arbitration stages which are still done in parallel.

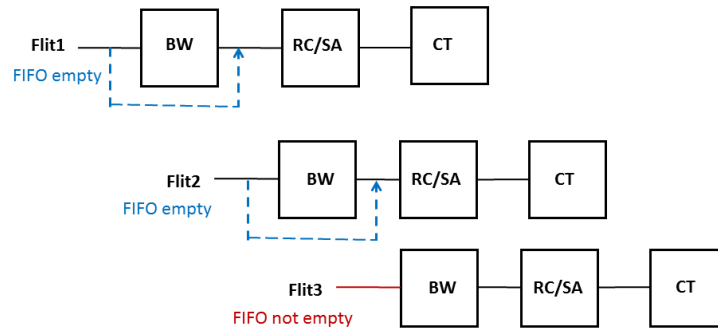


Figure 2.5: By-pass in LA-XYZ routing algorithm[7]

Figure 2.6, a full 3D-NoC's router system is shown. The architecture contains seven input modules for every direction in addition to the switch allocator and crossbar modules. While the input-port module is shown in Figure 2.7, each module contains two components: input buffer and router module[7].

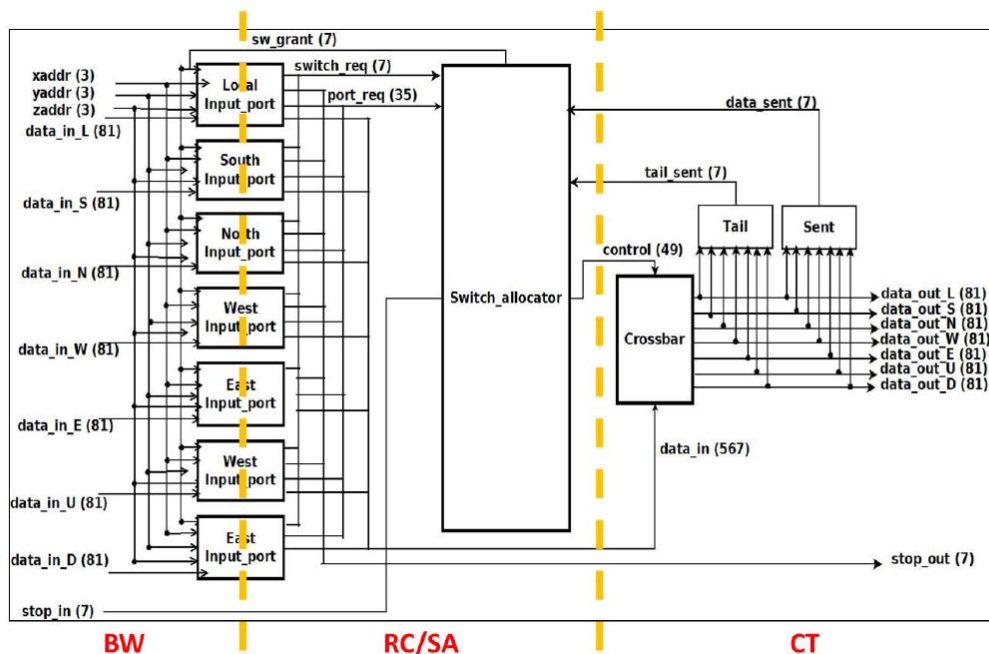


Figure 2.6: The architecture of the 3D-NoC's router[7]

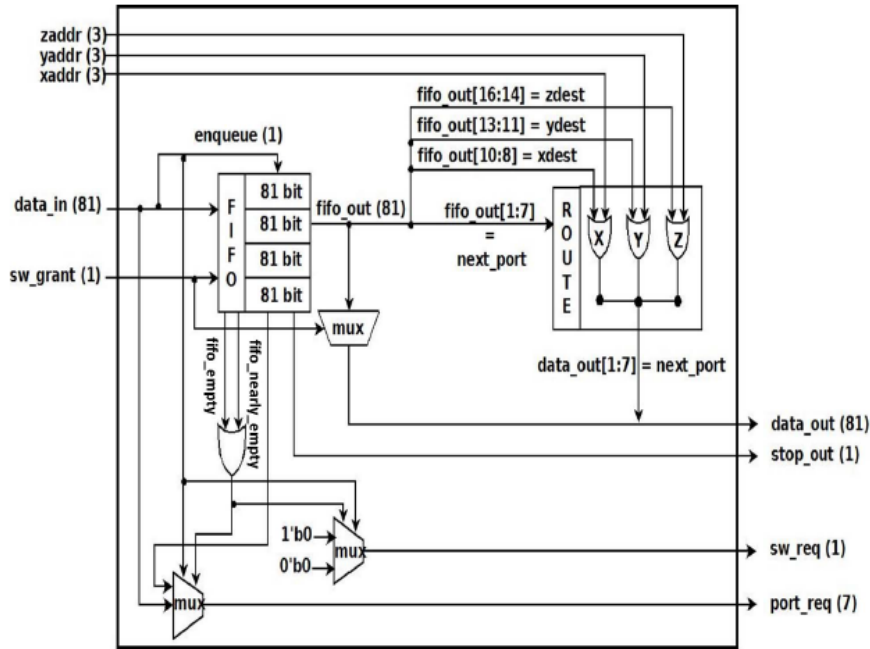


Figure 2.7: The module of input-port[7]

The 3D-NoC system which is using the LA-XYZ routing algorithm is evaluated with respect to the area utilization, power consumption and clock frequency. The authors used two famous benchmarks to evaluate their work which are as follows:

1. JPEG encoder application. The application is suitable to be used in evaluating the 3D-NoCs[28]. Additionally, the JPEG is a small application that can be mapped into eight nodes only. That can show the benefits of the 3D-NoC and the LA-XYZ routing algorithm clearly.
2. Matrix multiplication application. This application has the potential in reaching the best performance in parallel architectures and also creating enough network traffic to evaluate the system performance.

2.1.2 Topology Aware Adaptive Routing for irregular Mesh 3D-NoC systems

The thermal issue is such a critical challenge in the 3D-NoC systems. Therefore, the topology in these systems becomes irregular mesh topology. The routing algorithm here is proposed to balance the traffic load in the irregular Mesh 3D-NoCs[8]. The routing algorithm is denoted by Topology Aware Adaptive Routing (TAAR). The traffic becomes more in balance in the irregular mesh systems as shown in Figure 2.8 because of the larger path variety in the proposed routing algorithm.

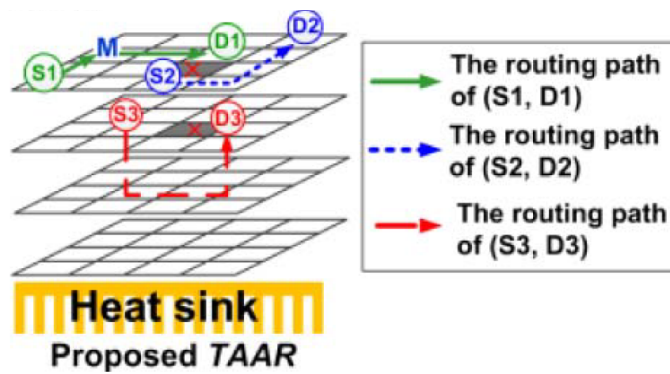


Figure 2.8: Larger path variety in TAAR routing algorithm[8]

For the diversity of the packet routing path, an adjustment is proposed, denoted by Topology-Aware Multiple Routing Adjustment (TAMRA)[8]. In another adopted routing algorithms, the virtual channels technique is used to solve the deadlock issue. This technique comes with a very large area cost. In TAAR routing algorithm after considering the area issue caused by the virtual channels, a Store-and-Forward (SAF) policy is used to avoid the deadlock problem in the routing algorithm by cutting the violated turn as shown in Figure 2.9. The TAMRA has two stages of serial routing as shown in Figure ??[8], the routed packets are temporarily saved in a network interface first.

The switching takes place, if there are two packets request the same output port direction. For the rerouted packets, an overhead latency happens due to the switching process. The worse case is when each router transmits one packet per cycle, the excess overhead of the latency will equal to (the largest packet size) x 3 clock cycles. While comparing this worst case overhead latency to the overhead of the SAF policy which is at least 10 to the power seven cycles, the re-routed packets can be easily neglected.

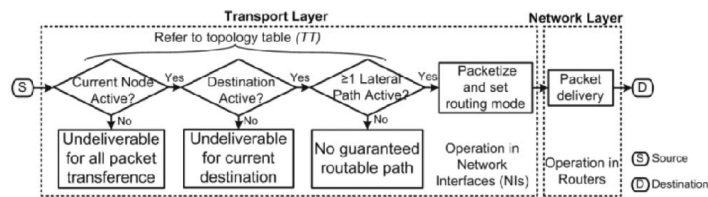


Figure 2.9: Operation flowchart of a transport layer and a network layer[8]

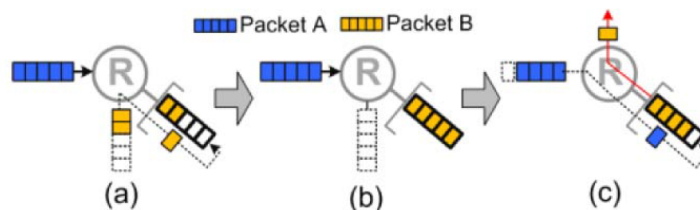


Figure 2.10: (a) Packet B is waiting for Packet A, (b) Packet A is prepared to retransmitted, and (c) Packet B start to enter the temporal storage[8]

The flow charts of the TAMRA is shown in Figure 2.11. The first routing stage, the adaptive routing mechanism is used to deliver the routed packet from the source node S to the intermediate node M. The routing mode is then determined according to the topology, then the packet will be routed from the M node to the final destination [8].

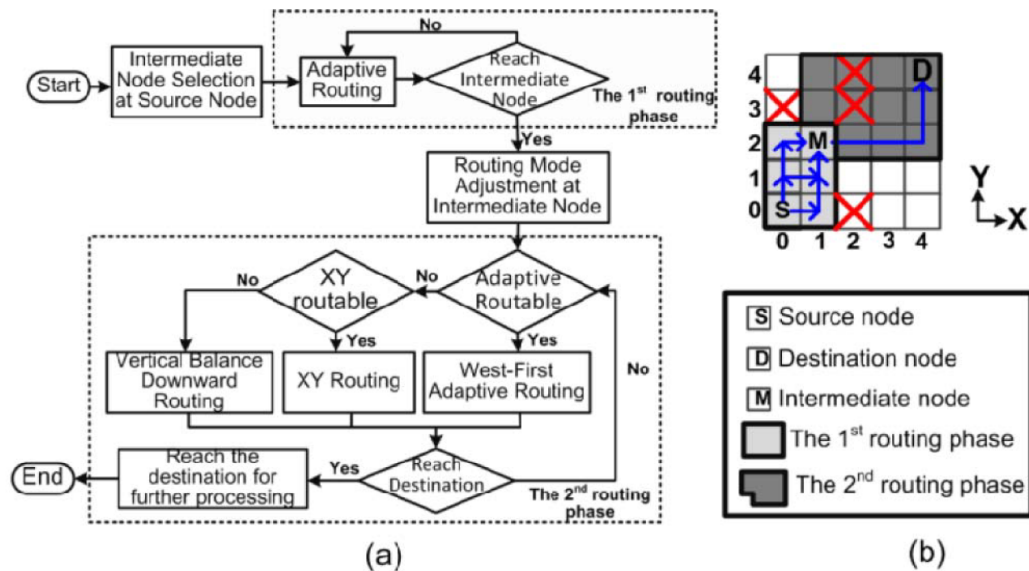


Figure 2.11: (a) Operation flow of TAMRA, and (b) An example [8]

2.1.3 Layered Routing in Irregular Networks

In this routing algorithm, the authors proposed a routing algorithm with groups of virtual channels in each network layer. For each virtual channels group, some source and destination address pairs are assigned [29]. That distribution in the traffic leads to an increase in the routing efficiency in the whole system.

The routing algorithm is easy to be implemented with no extra switch features than the already exist virtual channels.

The routing algorithm divides the the physical network into number of layers. Each layer represent a virtual network with the same connection to the original physical network. The packets will remain in each layer where they are first injected. Moreover, a routing function will include all the necessary information on the routing in a specific layer. As a conclusion, the routing algorithm provides no restrictions on the paths that the packet may take through the network. Meanwhile, the deadlock issue is avoided by forcing the packets to use specific layers. That leads to a shortest path routing with also a load balancing routing approach.

The routing algorithm can be used in many applications. The simulation results show that the used techniques lead to a significant performance gains in comparison with the standard UpDown routing algorithm.

2.1.4 Elevator-First Routing Algorithm

The Elevator-First routing algorithm is the most flexible and generic routing algorithm proposed in all the previous mentioned routing algorithms. It is designed for vertically partially connected in regular 2D dies as shown in Figure 2.12, the different regular 2D dies can be in various shapes and sizes such as mesh, ring, star and torus.

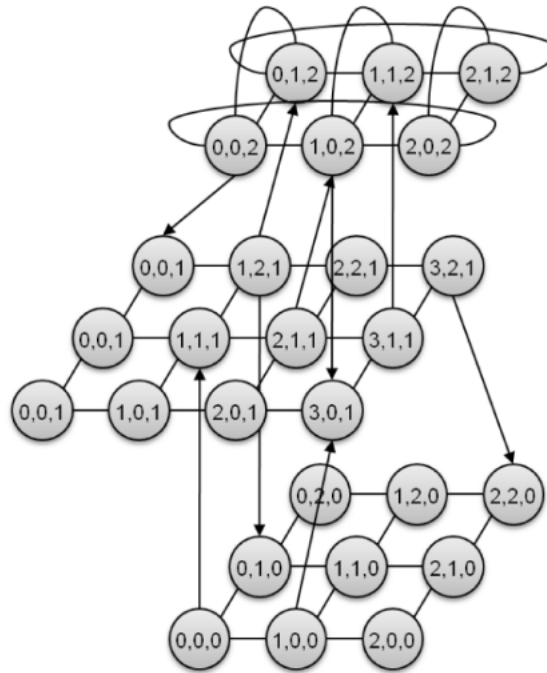


Figure 2.12: Vertically partially connected 3D-NoC[9]

The authors proved that the routing algorithm is fully independent of the shapes and dimensions of the 2D topologies, and also the number and location of the TSVs in the 3D-NoC system. A Vertically-Partially-Connected 3DNoC which is supported by this routing mechanism can come with different number, placement and location of the vertical interconnects, as well as the topology in each stacked die. Figure 2.13, shows an example of such architecture in which the routers are divided into four types as follows:

1. 2D routers which have five conventional ports.
2. 3D routers which have up links to connect the stacked dies together.
3. 3D routers which have down links to connect the stacked dies together.
4. 3D routers which have both up and down links

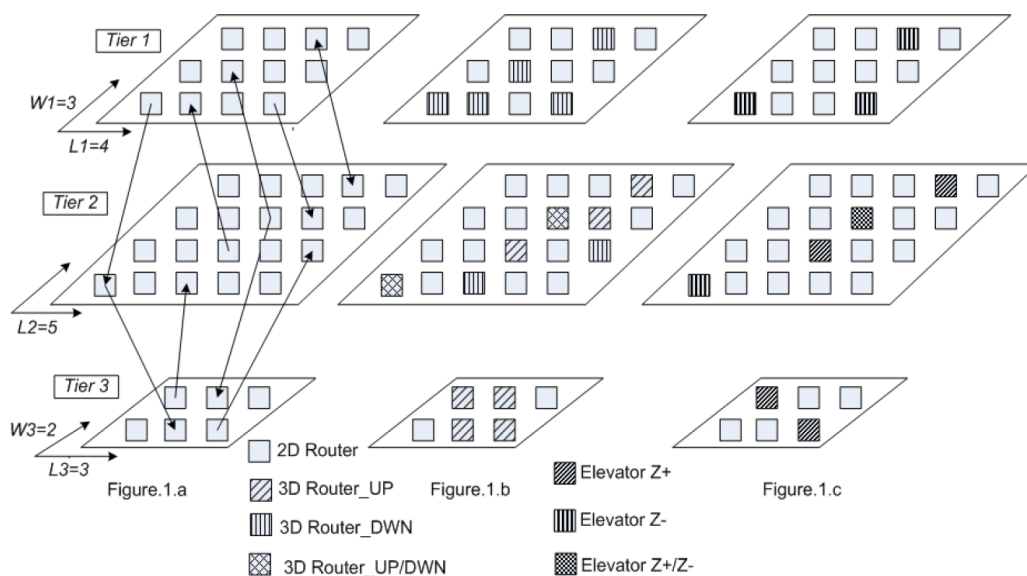


Figure 2.13: Different routers in Vertically partially connected 3D-NoC systems[9]

When issuing a packet through the 3D-NoC, if the source and destination nodes are in the same tier. Then, the packet will be routed directly according to the 2D topology in this tier. Otherwise, if the source and destination nodes are not in the same tier, an algorithmic dimension routing technique such as Z-first algorithm[9]. Figure 2.14 shows the Elevator-First algorithm flow, In tier two, a node S1 transmits a packet to a destination node D1 in an upper tier from the S1 node's tier. S1 has no direct vertical interconnect to the destination node D1. So, S1 will route the packet first to the Elevator Z+ in the same tier. It adds a temporary header to the packet that includes: Elevator address and two other flags. T and U flags are set to one, which means that the Elevator Z+ is not the final destination and that the packet is going up in the 3D-NoC. When the Elevator gets the packet and by checking the flag T. It realized that the header is temporary and will route the packet up as U flag is set to one.

The packet will be forwarded to the upper link and gets into the intermediate stage through a mediatory node M. The check will then occur again to see if the M and D nodes are on the same tier, if not same sequence of logic will take place[9]. The flow charts of the Elevator-First algorithm are shown in Figure 2.15, Figure 2.16 and Figure 2.17[9].

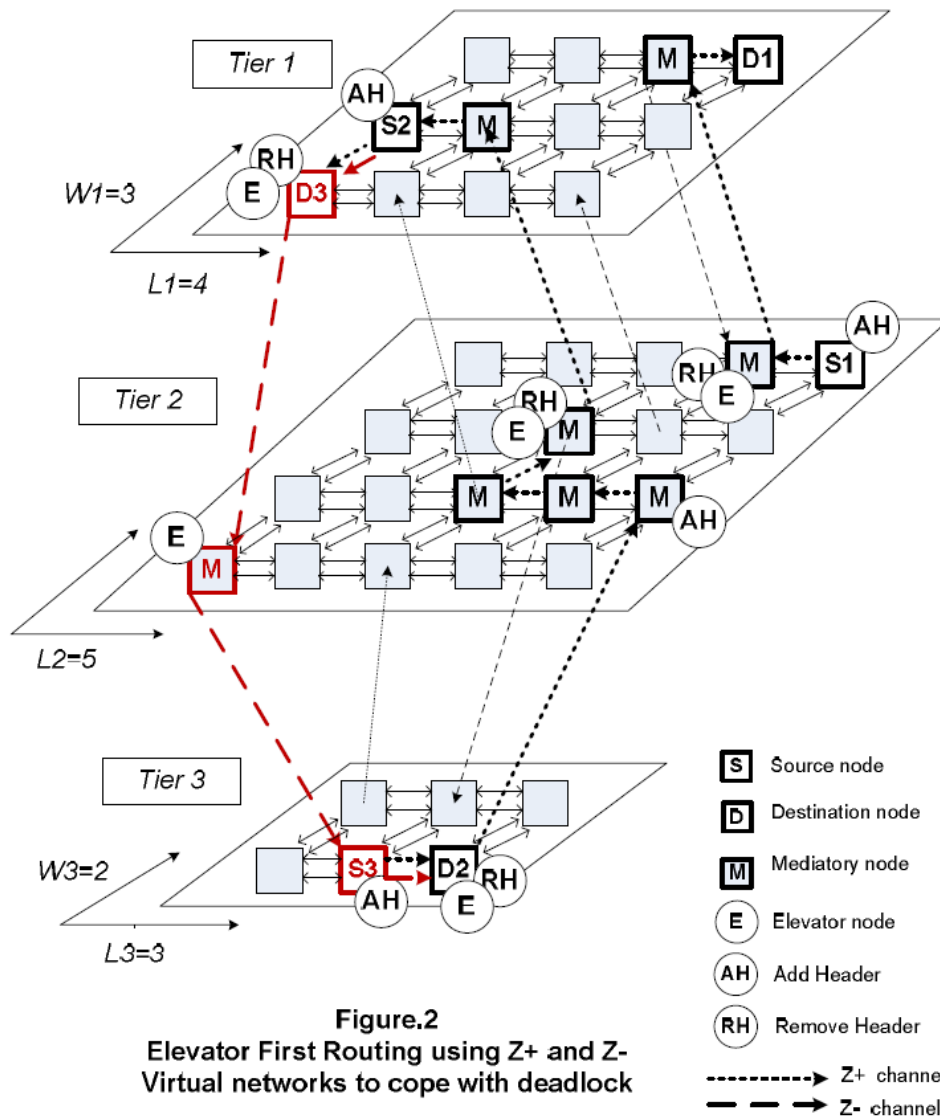


Figure.2
Elevator First Routing using Z+ and Z- Virtual networks to cope with deadlock

Figure 2.14: X-First 2D and Elevator-First Routing Algorithms[9]

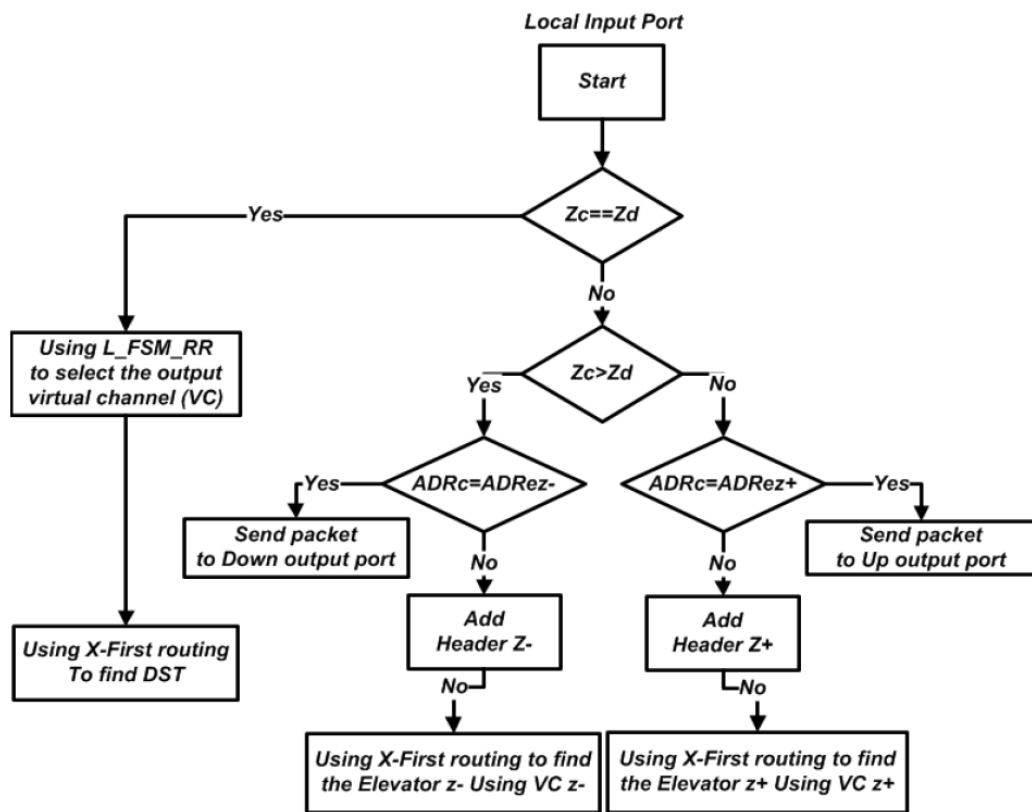


Figure 2.15: Elevator First Routing in LOCAL input ports[9]

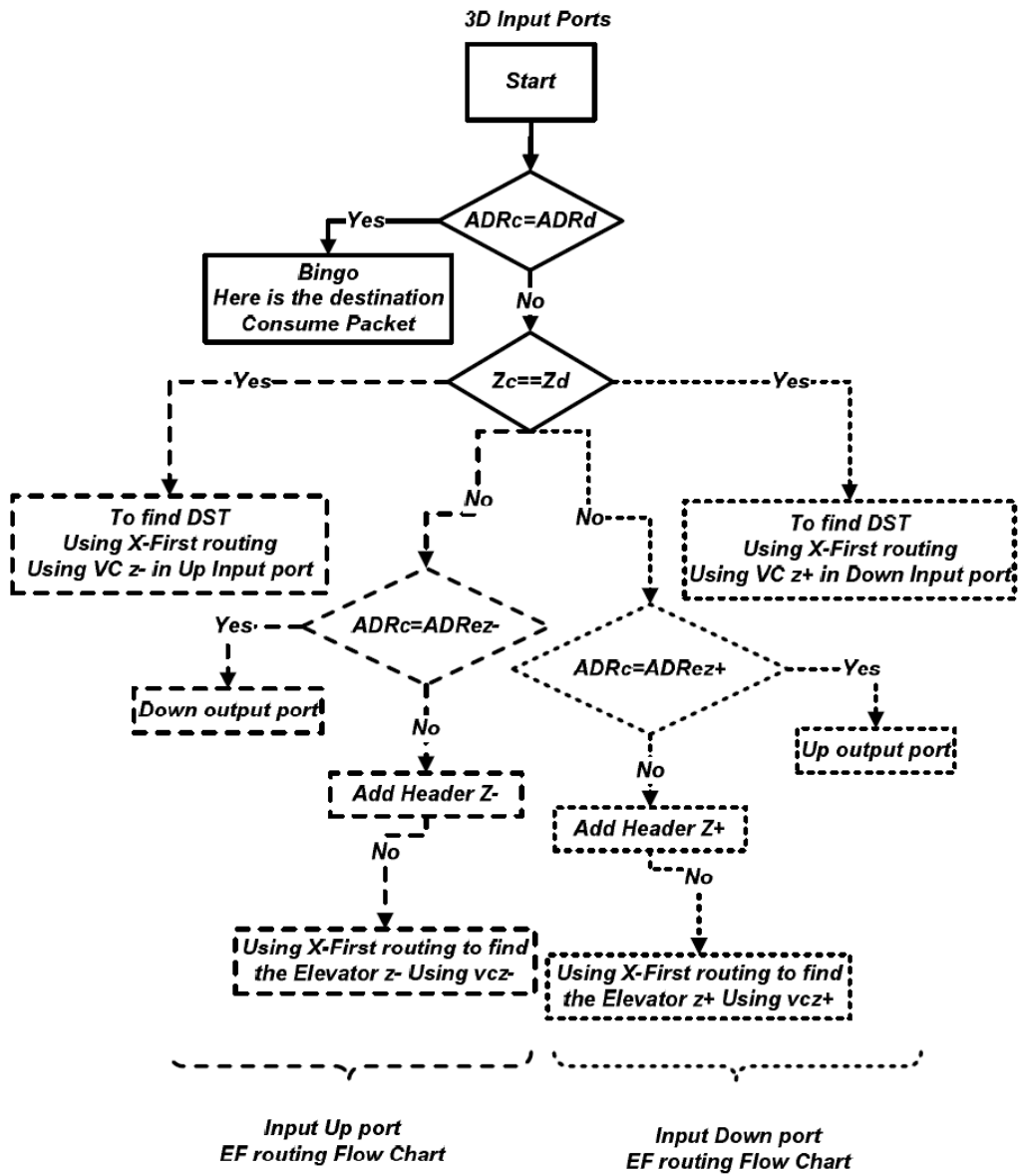


Figure 2.16: Elevator First Routing in 2D input ports[9]

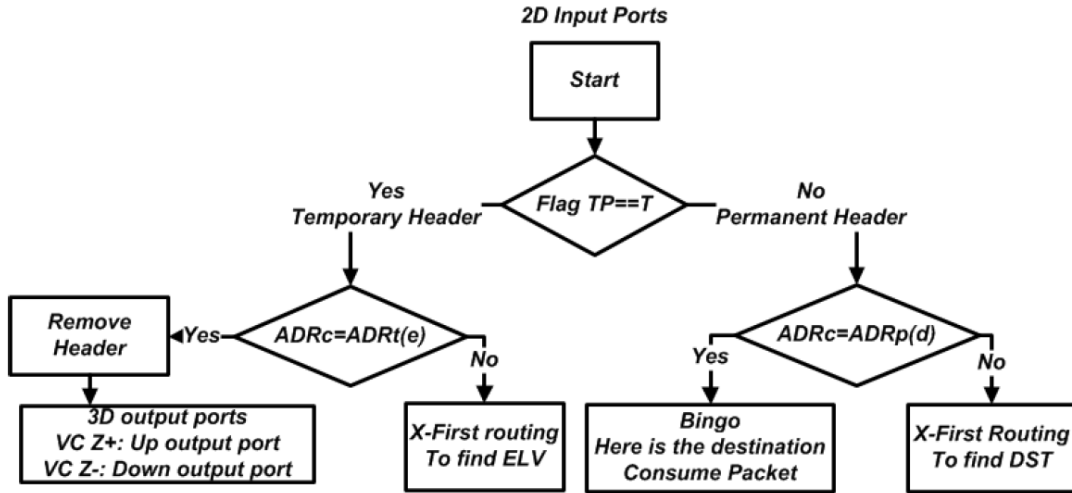


Figure 2.17: Different routers in Vertically partially connected 3D-NoC systems[9]

As a further work to implementing the Elevator-First algorithm, a 3D router is implemented to support the Elevator-First algorithm functionalities. The Elevator-First routing algorithm is the base of the Direct-Elevator algorithm which is proposed in this work which is illustrated in Chapter 4.

2.2 Simulators and Tools to Create Various 3D-NoCs

2.2.1 Booksim Simulator

Booksim simulator tool is proposed to create different NoC architectures. A simulator is implemented to be a cycle-accurate simulator for the NoCs, it can also model the interconnects between the network nodes. The author released two versions of the tool. First, Booksim1 was a simulator that produces a generic networks that did not target the NoCs environmental setup. consequently, the simulator has been used in many network contexts. Also, it has been used to study many network aspects such as network topology, routing technique, flow control, router architecture and the quality of the networks. From the perspective of the system, Booksim offers a wide flexibility, that comes from parametrizing all the network components within the tool.

The simulator is designed in modules to ease the modification process that may take place in any module of the system. However, the benefits provided by Booksim1, yet it did not support some important features and topologies which are implemented in Booksim2. Booksim2 was implemented and developed to include many other modifications as it reflects in a better way the state-of-art in the NoC research. Specifically, its features are more detailed in modulating the router architecture, the communication channel delay and support for the additional traffic load models.

As a simulator overview, the booksim is divided into hierarchical modules that implement different functionalities supported by the simulator as shown in Figure 2.18. Each module of these simulator modules has an interface which eases the replacement and modification of the module implementations without affecting other parts in the hierarchy.

Moreover, a top level block diagram is shown in Figure 2.19. The top level modules of Booksim are the traffic manager and the network. The traffic manager is a wrapper around the network that model the source and destination endpoints in the network. The wrapper injects packets into the network according to the user selected configuration which depends on the traffic pattern, packet size, packet injection rate and other aspects. The traffic manager is also responsible for ejecting the packets out of the network from the destination endpoints and terminating the simulation. The network top level includes a group of routers and channels, which are connected according to the selected topology. The communication between all nodes across the channels.

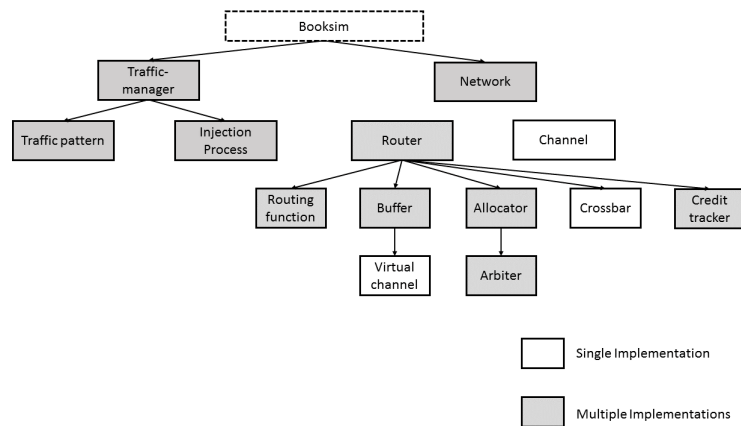


Figure 2.18: Module hierarchy of the simulator[1]

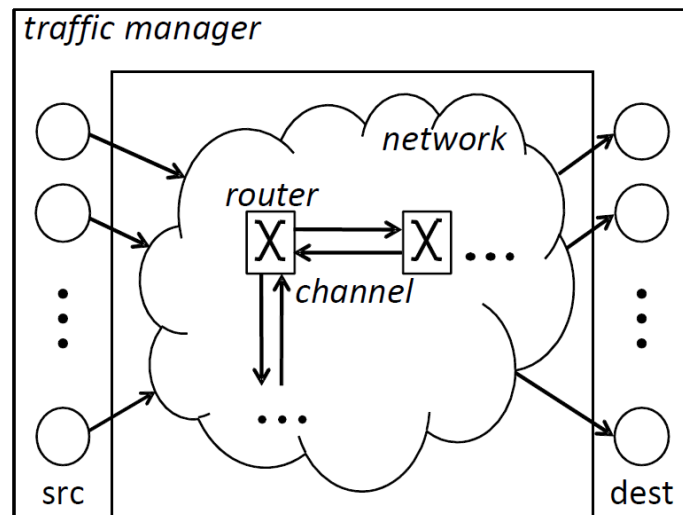


Figure 2.19: Top-level block diagram of the simulator[1]

A router has been implemented to support the Booksim tool functionalities. The components of the router are divided into four stage pipelined design as shown in Figure 2.20.

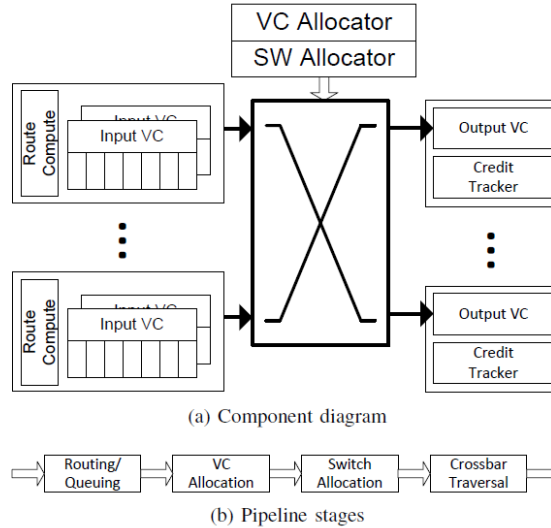


Figure 2.20: Router module of Booksim tool[1]

The tool comes with a lot of parameters as shown in Table 2.1, that provides the user with a large range of possible networks. The tool is available as an open source tool in which the user can also access its source files[1].

Table 2.1: Some Parameters of the Booksim Simulator (Available in both versions of Booksim)[1]

BookSim parameters	Description
Topology	
topology	Specifies the network connectivity based on well known topologies (e.g mesh, butterfly).
K,n	Specify the network size and configuration, for the selected topology.
Routing	
routing_function	Determines the routing algorithm for the selected topology (e.g., dimension order).
Flow Control	
num_vcs	Number of virtual channels per physical channel.
vc_buf_size	Input buffer size of each virtual channel.
wait_for_tail_credit	Enable atomic VC allocation.
Router Microarchitecture	
vc_allocator, sw_allocator	The type of allocator used for switch and virtual channel allocation.
credit_delay, routing_delay, vc_alloc_delay, etc.	Latency parameters for the router pipeline.
input_speedup, output_speedup, internal_speedup	Speedup in the crossbar and router pipeline compared to network channels.
Traffic	
traffic_pattern	Synthetic traffic pattern used in the simulation (e.g., uniform, transpose, etc.).
injection_rate	Average injection rate for each node.

2.2.2 Noxim Simulator

Noxim simulator is developed based on SystemC, which is a system description library written in C++[10]. The main motivation behind implementing Noxim tool is allowing a scalable performances to the different generated NoCs. The architecture of the NoCs depends on two components which are as follows:

- The network nodes: which represent the main items of the Noxim architecture. However the network nodes are always related to the input and storage elements, but they are also required to support different functionalities related to the data distribution.
- Processing elements: which are considered as the main components that actually performing storage and computation functionalities. The processing elements are also responsible for generating and consuming the data network packets.
- Routers: which are responsible for routing the data packets, communication techniques in the network and also implementing the flow control through the whole network system.
- Hubs: These components are playing a vital role in Noxim tool architecture as they allow the communications between any two non-adjacent nodes. The hub acts as an intermediate stage between the two nodes. The Noxim also provides an interesting possible architecture which is Hub-to-Hub wireless connection as shown in Figure 2.21.

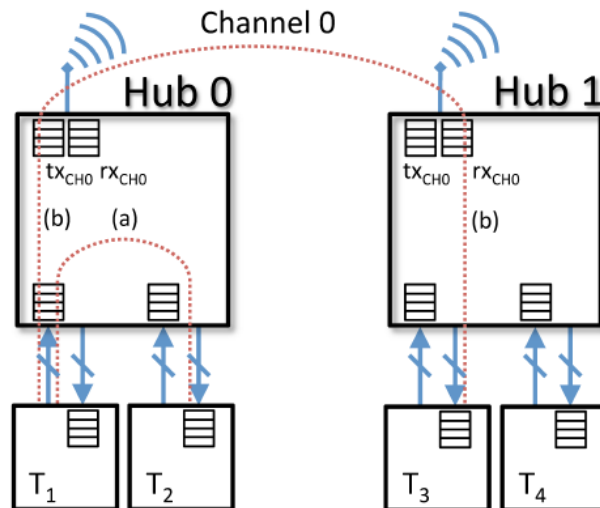


Figure 2.21: Internal architecture of hub nodes with wireless communication[10]

- Channels: in which the packet is transmitted using a known wireless frequency. The frequency of the transmission is taken by only one hub at a time.

Noxim offers a wide range of the NoCs that can be evaluated with respect to the power and performance. It is also available on the web as an open source tool for users.

2.2.3 NIRGAM: A simulator tool for NoC interconnect routing

NIRGAM tool is implemented based on SystemC, which allow the user to experiment and evaluate different application and routing mechanisms. It enables the user to analyze the performance with respect to the latency and the throughput[11]. Currently, the NIRGAM simulator supports mesh and torus topologies with wormhole switching technique. The users can choose of the available embedded routing algorithms such as deterministic XY, deadlock free odd-even, source and Q-routing[11]. The Quality of Service is also taken into consideration as the user can reserve a certain amount of the available band width for Guaranteed Throughput (GT) load traffic.

Figure 2.22 shows the flow diagram of the NIRGAM tool and its different modules.

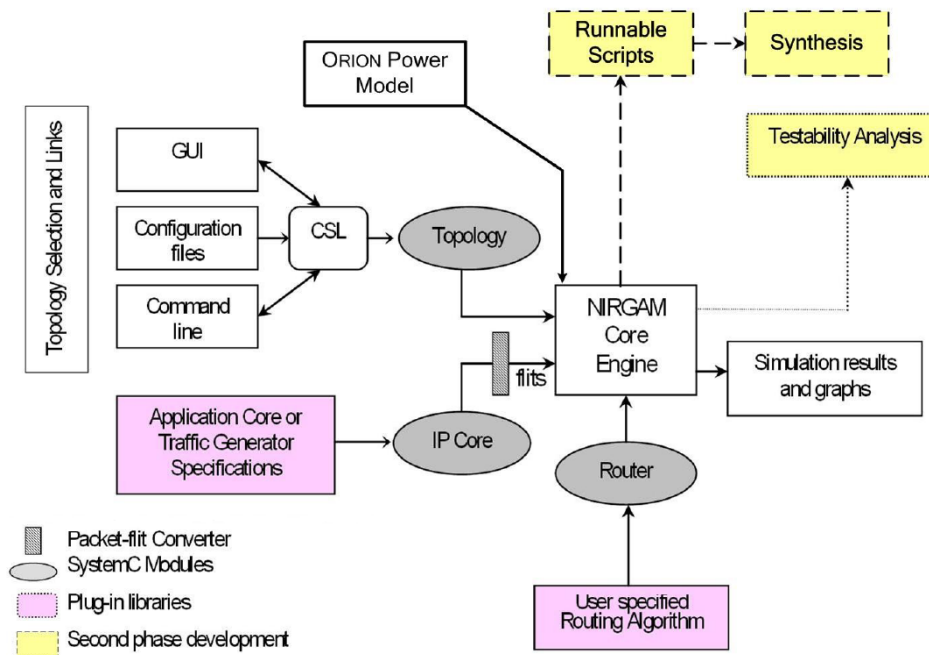


Figure 2.22: NIRGAM Simulator Flow Diagram[11]

Currently the following parameters are supported by the NIRGAM simulator as follows:

- Topology: 2D Mesh and 2D Torus.
- Switching mechanism: Wormhole.
- Routing Algorithms: Source routing, deterministic XY and adaptive Odd-Even (OE).
- Applications: sender, receiver and traffic generator.

- Plug in to support network routers.
- Plug in to support the applications.

NIRGAM tool is also available on the web for downloading and usage

2.2.4 CONNECT tool to create different 2D-NoCs

CONNECT is a NoC generator that provide the user with synthesizable RTL designs. The NoCs produced from CONNECT are evaluated against high-quality RTL NoC based on ASICs. CONNECT NoC is meant to be part of an FPGA system[12]. Therefore, the CONNECT NoCs have to balance between two main tradeoffs:

1. Providing efficient performance to satisfy the application requirements.
2. Minimizing the usage of the FPGA resources to utilize them for the rest of the system components.

CONNECT takes both tradeoffs into consideration by making the NoC implementation as efficient as possible.

A router has been implemented for the communications over the NoCs created by CONNECT. The routers are configurable due to many parameters, the router architecture is shown inFigure 2.23.

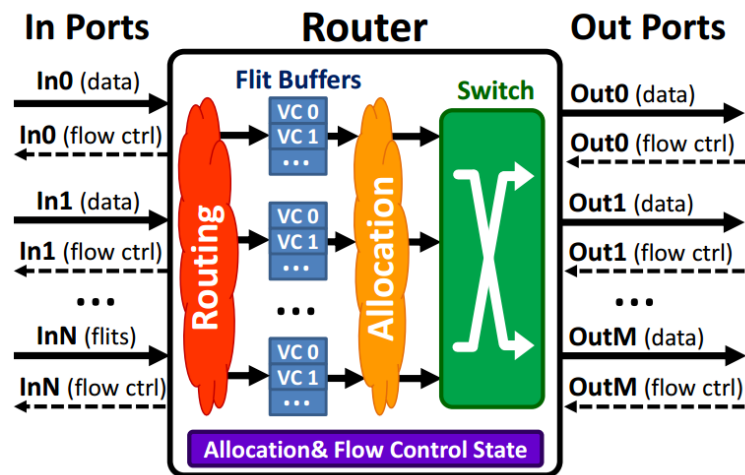


Figure 2.23: CONNECT Router Architecture[12]

The CONNECT tool comes with a web-based Graphical User Interface (GUI) as shown in Figure 2.24 that allows the user to enter all the 2D-NoC specifications regarding the following components:

- Network topology.
- Number of Endpoints.
- Router type.
- Number of virtual channels.
- Flow control type.
- Flit data width.

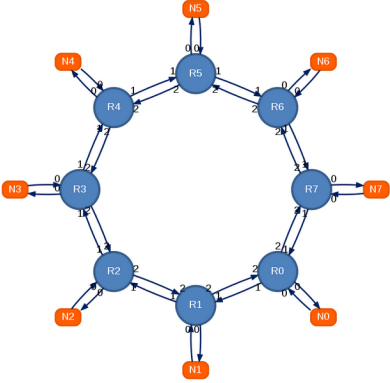
Parameter	Value	Preview (<input type="checkbox"/> hide endpoints)
Network Topology		
Topology <small>(i)</small>	Double Ring	 <p>Click on preview image to enlarge.</p>
Number of Endpoints	8	
Network and Router Options		
Router Type <small>(i)</small>	Virtual Channel (VC)	
Number of VCs <small>(i)</small>	2	
Flow Control Type <small>(i)</small>	Credit-Based Flow Control	
Flit Data Width <small>(i)</small>	64	
▶ Advanced Options (click to expand)		
Contact and Delivery Info		
Name	First Last	
Affiliation		
Email <small>(i)</small>	Valid email required	
<input type="checkbox"/> I have read, understood, and I agree to the license terms		
Generate Network	← click here to generate network	

Figure 2.24: CONNECT Tool[12]

As mentioned before CONNECT is tailored to be used on FPGAs. The FPGA has a reconfigurable nature which represents the main difference between them and ASICs. Implementing NoCs using FPGA creates very unique opportunities as well as challenges. The CONNECT tool provides NoCs which uses the FPGA reconfigurability in many of its applications.

The architecture of CONNECT NoC has two main routing design decisions which are as follows:

- Single pipeline stage: CONNECT uses a single stage router pipeline instead of the common three or five stages router. That leads to a low hardware cost, low latency and also a more efficient buffer usage due to the reduction in the round trip between the network routers.
- Tightly coupled routers: CONNECT tool maximizes the wire usage by using wider interfaces that leads to thinner coupling routers.

The most important advanced features of the CONNECT tool are as follows:

- **Topology-agnostic:** The tool provides a flexibility related to the routing algorithm which gives the ability to use different topologies.
- **Virtual Channels:** CONNECT tool uses the virtual channels to avoid the deadlock problem that may occur in routing the packet through the NoC system.
- **Virtual Links:** These links facilitate the implementation of the NoC's endpoints. This feature guarantees transmitting and delivery of multi-flit packets.

The synthesis results of CONNECT tool are evaluated using Xilinx XST technology. Additionally, all results are compared against a high quality virtual channel based router called SOTA to consider the hardware cost and network efficiency of CONNECT as shown in Figure 2.25.

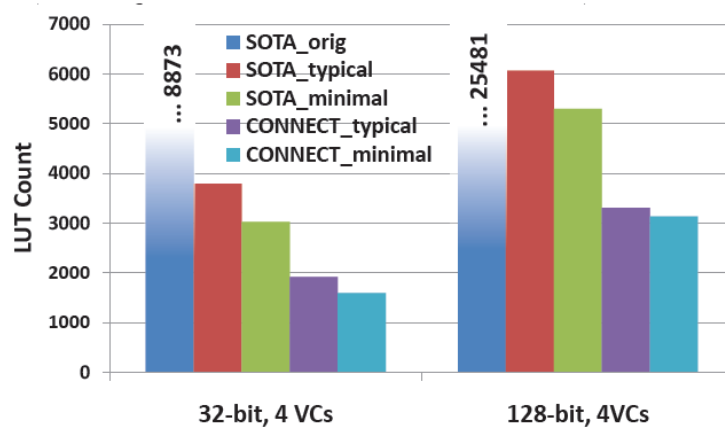


Figure 2.25: Comparison between FPGA cost of SOTA and CONNECT router[12]

In this thesis, a tool is proposed to create synthesizable 3D-NoCs structures as the 2D-NoC created by the CONNECT tool.

The simulators in general are very useful for the following purposes:

- Evaluating different hardware implementationd without the cost of fabricating the hardware physically.
- Debugging the problems through a fully controlled environment.
- Obtaining a detailed performance analysis with respect to most of the important metrics.

A valuable survey is conducted on the different NoCs tools[2]. The survey compares between many tools whether they are developed from research groups or from the industry. The NoC tools are classified into two main categories: synthesizers and simulators. For the synthesizers, the quality of generated NoCs with respect to area and power consumption is usually discussed. Also, synthesizers come in industrial commercial versions such as FlexNoC[30] and INOC[31].

On the other side, the simulators are mainly focused on the network performance with respect to the output throughput, latency and the overall system reliability.

The authors show a precise comparison between different simulators and synthesizers in this survey[2]. Many NoCs simulators and synthesizers are shown in this survey in Table 2.2.

Table 2.2: Different NoC simulator and synthesizers[2]

#	Tool	Year	Team	References
1	NS-2	1995	DARPA and later Contributors	[22]
2	Noxim	2010	Catagne University	[26]
3	DARSIM	2009	MIT	[27]
4	SunFloor – 3D	2006 - 09	EPFL (switzerland)	[28-30]
5	ORION 1 et 2	2003 - 09	Princeton University	[32-34]
6	INSEE	2005	Basque University (Spain)	[37]
7	ATLAS	2005	Federal University of Brazil	[38]
8	NOCIC	2004	Massachusetts University	[39]
9	Pestanna Environment	2004	Phillips Research Laboratories	[40]
10	PIRATE	2004	Polytechnique School of Milan	[41]
11	SUNMAP	2004	Stanford University	[42]
12	xpipesCompiler	2004	Bologne University – Stanford University	[43]
13	μ Spider	2004	Bretagne Sud University	[44]
14	OCCN	2004	ST microelectronics	[45]
15	NoCGEN	2004	University of New South Wales	[46]
16	FlexNoC	-	ARTERIS	[17]
17	iNoC	-	-	[18]
18	The CHAIN works tool suite	-	Silistix	[19]

Table 2.3 shows a general comparison between different other simulators, while the NOMENCLATURE of the comparison can be found in Table 2.4.

Table 2.3: A comparison between different simulators[2]

#	Tool	Specification												
		Modeling							Simulation				Hardware Synthesis	Availability
		NS	BS	PD	RA	SS	PIR	TD	AC	PC	T	L		
1	NS-2	+	+	+	+	+	+	+	+	+	+	+	-	+
2	Noxim	+	+	+	+	+	+	+	-	+	+	+	-	+
3	DARSIM	+	+	+	+	+	+	+	-	-	+	+	-	-
4	SunFloor – 3D	+	-	-	-	-	-	-	-	+	+	+	+	-
5	ORION 2.0	-	+	-	-	-	-	-	+	+	-	-	-	-
6	ATLAS	+	-	+	+	-	+	-	-	-	+	+	+	+
7	PIRATE	+	+	-	-	-	-	-	+	+	-	-	-	-
8	SUNMAP	+	+	-	-	-	-	-	-	+	+	+	+	-
9	μSpider	+	+	-	-	-	-	-	-	-	+	+	+	-
10	NoCGEN	-	+	-	-	-	-	-	-	-	+	+	+	-
11	FlexNoC	+	+	+	+	+	+	+	+	+	+	+	+	commercial
12	iNoC	+	+	+	+	+	+	+	+	+	+	+	+	commercial
13	The CHAIN works tool suite	+	+	+	+	+	+	+	+	+	+	+	+	commercial

Table 2.4: Nomenclature of the comparison table[2]

Modeling	Simulation
NS: Network Size	AC: Area Consumption
BS: Buffers Size	PC: Power Consumption
PD: Packets Distribution	T: throughput
RA: Routing Algorithm	L: Latency
PIR: Packets Injection Ratio	
SS: Selection Strategy	
TD: Traffic Distribution	

Chapter 3: An Overview About The Three Dimensional Network-On-Chips

3.1 Three Dimensional Technology

Three Dimensional (3D) technology is a new trend in which a stack of many dies are fabricated in one chip. As the transistor's size keeps shrinking, Moore's law continues to yield higher transistor density with each new process transistor generation. That leads to multi-core Chip Multi Processors (CMPs) with hundreds of interconnects in one chip[26]. The 3D technology becomes more vital to solve the Moore's law challenges. Moreover, The 3D intergration technology combines the diverse ability of System in a Package (SiP) while expanding the intergration capabilities of the System-On-Chips (SoCs)[32].

3.1.1 3D Stacking Technologies

The 3D integration technology has four types of stacking as follows[3]:

1. Package stacking:
In this technology, the different dies are packaged individually then stacked together in one 3D chip. The number of 3D interconnects in this case is in the range of hundreds of micrometers. So, that puts a constraint on the number of 3D interconnects. The advantages of this technology are the moderate design time and also fabrication cost.
2. Die Stacking:
In this technique, the connection of the dies is using Through Silicon Vias (TSVs) and micro bumps. The difference of the die stacking than the other stacking technologies, is that die stacking stack dies individually in the substrate. However, the density of the TSV interconnects is still in the range of micrometers.
3. Wafer Stacking:
This stacking technique stacks first the wafers then cut the wafers into dies. It requires shorter time when it is compared to the die stacking. The main issue with this technique is the low production yield.
4. Device Stacking:
This stacking technology is done by fabricating the active layers at low temperature VLSI steps[33]. It offers a more communication channels between the different layers. Therefore, It gives the best solution for the high density storage and parallel processing applications.

A comparison between the different types of stacking technology is shown in Table 3.1.

Table 3.1: Comparison between different stacking techniques[3]

	Package	Die	Wafer	Device
Pitch of 3D Interconnects	Large	Moderate	Moderate	Small
# of 3D Interconnects	Less	Moderate	Moderate	More
Parasitic of 3D Interconnects	L/C	R/C	R/C	R/C
Device Degradation	No	Minor	Minor	Yes
Thermal Impact	Less	More	More	More
Heterogeneous Integration	Okay	Okay	Okay	Maybe
Design Time	Less	Moderate	Moderate	More

3.1.2 Vertical Interconnects Technologies

The 3D technology comes with many types of vertical interconnections. Table 3.2 shows a brief about the several 3D interconnections. A comparison in terms of assembly level, maximum number of tiers in 3D structure, interconnect pitch, and the connected routing resources. While in Figure 3.1, each interconnecting technique is illustrated[4].

Table 3.2: Comparison of vertical interconnect technologies[4]

Characteristic	Wire bonded	Microbump 3D package	Microbump Face-to-face	Contactless Capacitive	Contactless Inductive	Through via Bulk	Through Via SOI
Assembly level	Die	Die	Die	Die	Die	Wafer	Wafer
Tier limit	Assembly process	Heat	Assembly process	Assembly process	Heat	Heat/yield	Heat/yield
Vertical pitch (mm)	35 to 100	25 to 50	10 to 100	50 to 200	50 to 150	50	5
Metal Layers	All	Top 1 to 2	Top 1 to 2	Top	Top 1 to 2	All, top	All, top

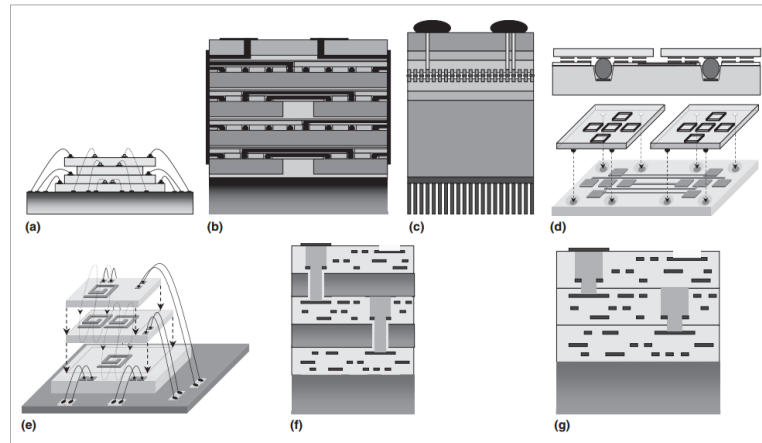


Figure 3.1: Illustration of vertical interconnect technologies: wire bonded (a); microbump 3D package (b) and face-to-face (c); contactless capacitive with buried bumps (d) and inductive (e); through via bulk (f) and silicon on insulator (g)[4]

The vertical interconnects as shown in Figure 3.1 can be in various types as illustrated below:

- Wire bonded: Most famous technique, in which the wires connect each die in the stack of the 3D structure.
- Microbump: Microbump approach uses gold or solder bumps on the surface of each die to make the required connections.
- Contactless: This technique uses capacitive or inductive coupling to communicate between different dies in the vertical dimension.
- Through via bulk: The TSV approach provides the maximum possible interconnect density. In which the interconnection can be done whether face-to-face or face-to-back.

3.2 Different Implementations and Applications with 3D Technology

The 3D technology offers a large scale of new implementations and applications. In this section, two interesting applications and implementations using the 3D structure are illustrated.

3.2.1 3D Chip Structure with an iA32 Microprocessor

In this implementation, the 3D structure is evaluated while it is applied to a iA32 microprocessor. A design database for the microprocessor is evaluated while floor planning it for 3D structure[13]. In Figure 3.2, the 3D structure is shown. Two dies are connected together in a face-to-face shape with a vertical interconnect.

The vertical interconnects are located on top of the metal stack for each die. The heat is used here to bond the two stacked dies together. The backside vias are used to connect the input/output to the two stack dies. The power then is delivered across these backside interconnects.

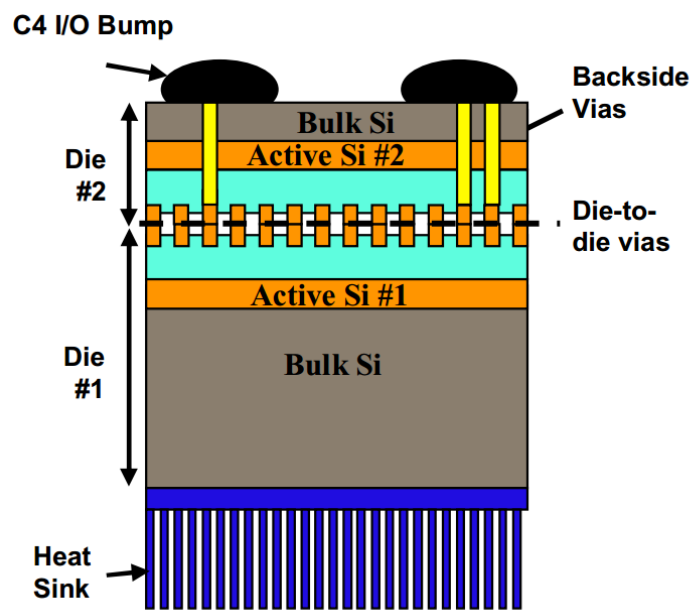


Figure 3.2: 3D Structure[13]

3.2.2 CMOS Based Image Sensors Using 3D Technology

The demand of thinner and lighter devices such as tablets, laptops and mobiles is increasing, so developing smaller and better resolution CMOS image sensors becomes vital for such industries. In this application, the 3D technology is used for the imager applications with a low temperature oxide to oxide and TSV interconnection. The process enables electrically the stacking and connection of two chips- illuminated imager chip above an image processing chip.

The proposed approach is based on low temperature oxide-oxide bonding and TSV process as shown in Figure 3.3 which represents a schematic cross-section of the design[14]. The bonding quality in the 3D structure of the application is studied by a SAM (scanning acoustic microscope) approach. This can spot any defect at the oxide-oxide bonding interface. Figure 3.4 is a SAM image of a 200mm wafer pair after the bonding is done, the black color shows the good quality of the bonding interface[14].

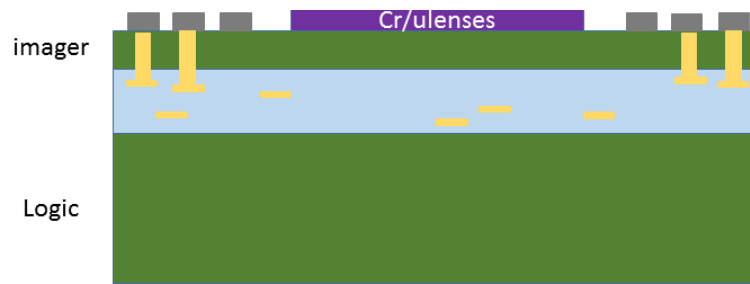


Figure 3.3: A cross-section of an imager die schematic[14]

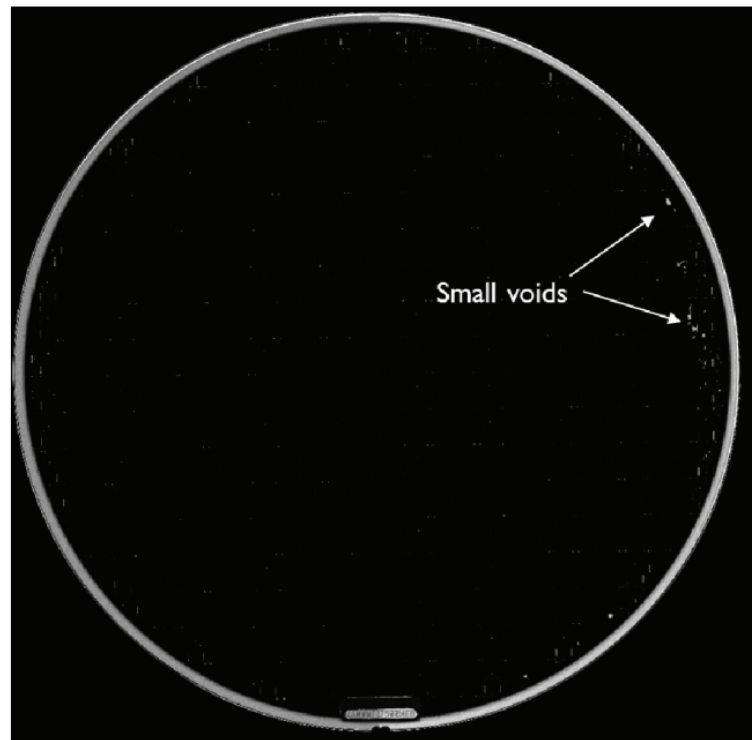


Figure 3.4: SAM image[14]

3.3 Challenges of the 3D Integration Technology

Nevertheless, The new technology comes with several benefits, it comes with very challenging research problems. In this section, the most vital challenges are going to be discussed.

- Cooling is required according to the high power and circuit density. Moreover, for large applications with high performance effective cooling techniques to control the temperature while keeping the high performance due to the short vertical connections used to connect the 3D stacked chips as shown in Figure 3.5.

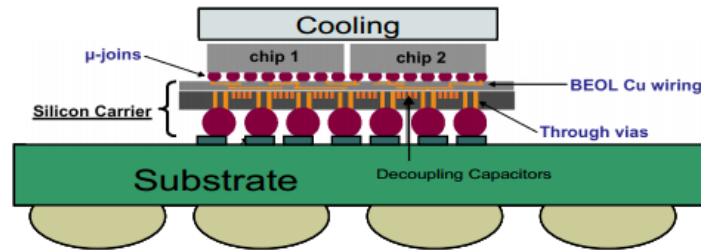


Figure 3.5: Cooling used over a 3D structure[15]

- The usage of TSVs introduces new challenges in the fabrication process such as: etching technique, metallization, and shape of the TSV[15].
- The wafer bonding introduces a very important challenge which is the alignment accuracy between the stacked dies in the 3D structure. The bonding misalignment can be measured using Vernier-type structure[34].

3.4 Network-On-Chips

The networking techniques are applied to the SoC designs to solve the wiring challenges in the fabrication process. Physical wires are the communication channels between the different cores in the SoC design. Wiring width increases in higher wiring level, that leads to thicker and wider wires in the top levels than the low levels.

The increase in the wiring width decreases the wire resistance, also the spaces between the different wires keep the capacitance effect from growing.

Additionally, at the same time the inductance effect increases relatively to the resistance and capacitance effects do.

Therefore, global wires will become very lossy[27]. Network On Chip (NoC) approach introduces a solution for this global wiring growth issue, by providing interconnection networks between the different chips. Figure 3.6 shows a NoC which has different cores connected together via routers nodes.

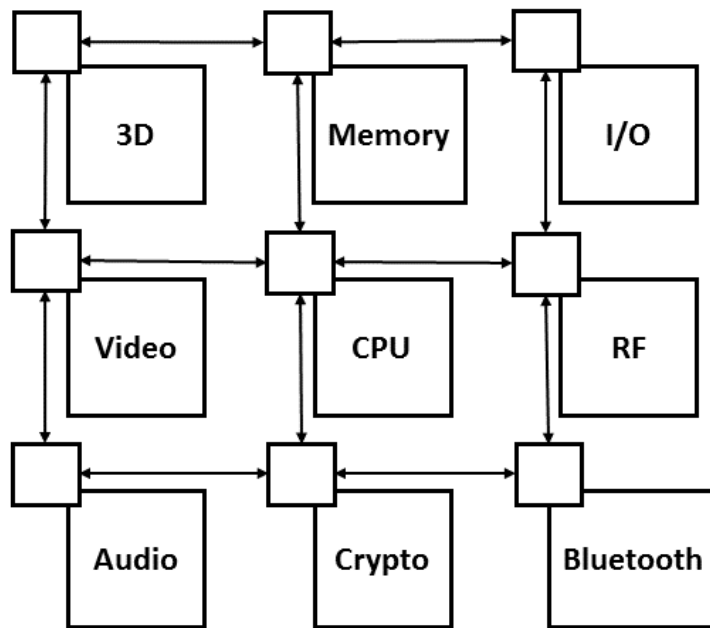


Figure 3.6: Example of a NoC[16]

3.5 The Architecture of NoCs

Building a NoC that is most convenient for a specific application requires decisions about the router and network designs[17].

3.5.1 Router Design

The router design depends on mainly the number IPs connected to it, as it reflects directly on the number of input and output ports and whether they are different or equal[17]. There are some aspects related to the router design as follows:

- Switching technique such as the virtual cut through switching technique in which the packets are routed directly once they arrive if there is an available output channel.
- Routing algorithm.
- Router Implementation. The router contains input and output blocks as shown in Figure 3.7 to support the routing algorithm functionalities and communicate between different IPs[17].

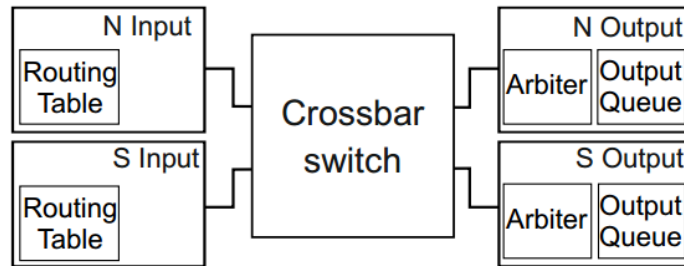


Figure 3.7: Two Inputs/Two Outputs router Implementation[17]

3.5.2 Network Design

Network design depends on many aspects such as the number of cores, the number of routers and the network topology. Network topology is considered a very important aspect in network designing as it gives the opportunity to create different structures of NoCs such as mesh, star and torus topologies.

3.6 Partial Reconfiguration Using NoCs

The Dynamic Partial Reconfiguration is considered one of the most vital applications of using the NoCs. It can be used to reach high hardware flexibility, reliability and utilization[18]. In DPR, part of the circuit can be dynamically reconfigured while other parts of the circuits are operational at the same time.

The DPR is implemented on Field Programmable Gate Array (FPGA) and based on the NoC methodology of communication between the different cores. Figure ?? shows the architecture of a NoC DPR, it is composed of NoC which includes router and Resource Interface[18].

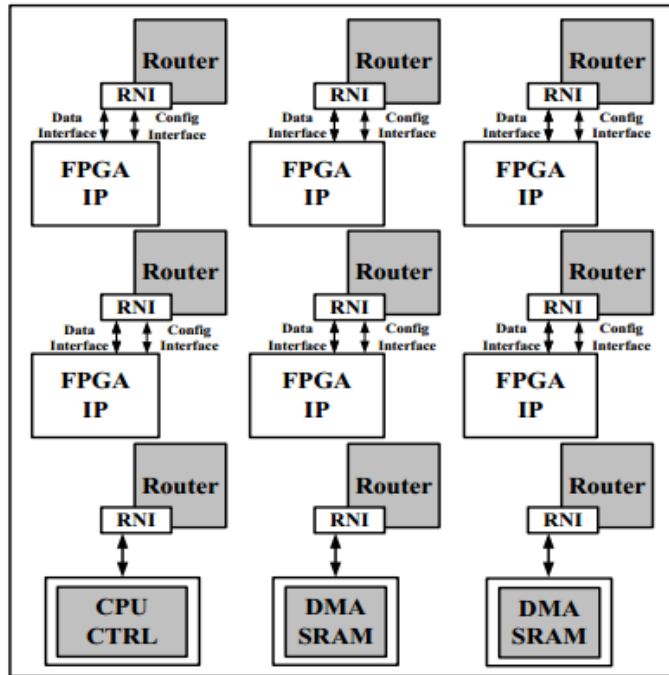


Figure 3.8: A NoC DPR Architecture[18]

3.7 The Three Dimensional Network-On-Chips

As Illustrated earlier in this chapter, the 3D intergration technology and the NoCs have numerous benefits individually. The Three Dimensional Network-On-Chips (3D-NoCs) combine the benefits from the two technologies and create opportunities for new applications.

As mentioned before, the NoC methodology solved the perfomance constraints arising with the long interconnects across the chips. Also, it allows the intergration of many IPs in one single SoC chip. On the other hand, the floorplanning of the 2D chips puts restrictions on the enhancements that can be provided by the NoC[35].

Therefore, the 3D technology enables new possible architectures that were very hard to be implemented before a long with the NoCs.

3.7.1 The Architecture os 3D-NoCs

The new vertical dimension allows a large freedom in selecting the topology for each stacked NoCs. There are different 3D-NoCs structures as follows:

- Mesh based network: It is the most common 2D-NoC structures. A 3D mesh network is shown in Figure 3.9.

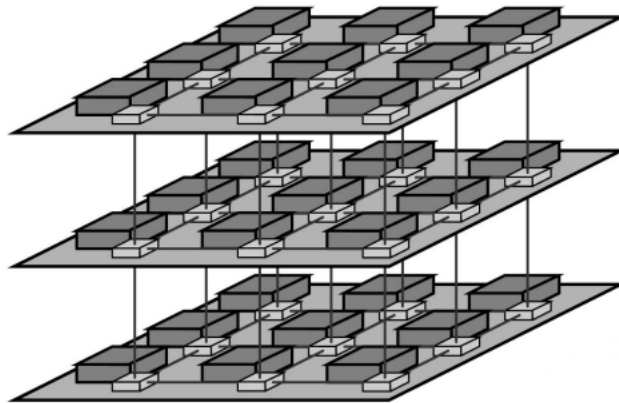


Figure 3.9: 3D Mesh based network[19]

- Tree based network: there are two tree based network architectures such as butterfly fat tree which is shown in Figure 3.10.

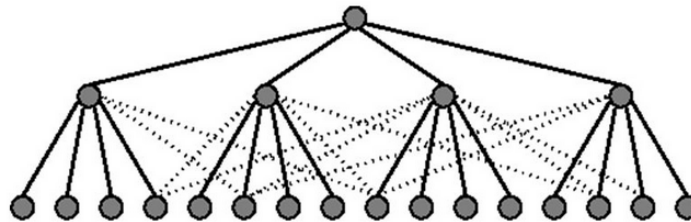


Figure 3.10: Fat Tree Model[20]

The different architectures of the 3D-NoCs introduce different problems regarding the power consumption of the whole system, the placement of the routers, and the placement and number of the vertical interconnects.

Many studies have been conducted to evaluate the different 3D-NoCs architecture with different 3D routing algorithms.

Chapter 4: Introduction To The Network-On-Chip Methodology in FPGAs

As mentioned in the previous chapter the vital role of the NoC methodology in general design chips becomes very important. The introduction of the NoC concept to the FPGAs is one of the main motivations behind this thesis work.

In this work, a tool is proposed to create synthesizable 3D-NoCs that can be evaluated across the latency, area and power aspects. The 3D-NoCs created are also tailored to be part of an FPGA. Therefore, highlighting the importance of the NoC approach in the FPGAs has to be part of this thesis background.

In the previous chapter the architecture of NoCs and the partial configuration concept are presented shortly as a part of the introduction to the 3D-NoCs in general. In this chapter, a detailed study about the NoC in FPGAs is going to be conducted.

Adding the NoCs within the FPGAs has the ability not only to enhance the system efficiency of the interconnections between the different nodes in the system, but also to increase the design production and reduce the compilation time using a higher level of abstraction in the communication through the system. The interconnections in the FPGA faces lately many challenges which we can summarize as follows:

- Poor interconnection scaling due to the increasing in wire resistance while decreasing in the transistor performance.
- The time consumption is really high due to the high interconnect delay in the system which leads to a difficulty in determining the critical path in the functional design description.
- Large interconnection amounts are consumed due to the new high speed input/output interfaces.
- The low level of the abstraction for the interconnects leads to a large compilation time which affects the whole design efficiency.

According to these challenges, the NoC methodology is introduced within the FPGA as a solution. Many studies have been performed to evaluate the methodology and its all implemented components.

4.0.1 A comparison between the soft NoC and hard NoC on FPGA

A study has been conducted to compare between the tradeoffs of the soft and hard implementations for NoC[5]. Authors present three different FPGA architectures and implementations of a hard NoC implementation. Then, these three implementations are compared to a soft NoC implementation to determine the advantages from each one.

All implementations have hard routers and two of them have additional hardware to support the Time-Division Multiplexed (TDM) wiring feature. The third implementation is a standard FPGA based on hard routers. All implementations are illustrated in Table 4.1.

Table 4.1: The evaluated hard NoC implementations[5]

Router	Port width	Freq	Wiring
Soft StratixII	32 bit	200 MHz	Static
Soft Virtex4	32 bit	200 MHz	Static
Hard	32 bit	200 MHz	Static
Hard	8 bit	800 MHz	TDM
Hard	8 bit	800 MHz	TDM (shared)

The TDM feature enables the time multiplexing between the hard blocks which include the network routers. That certainly leads to better area utilization. Hard routers for FPGA have been proven to be more area and power efficient, and also give a better performance[21]. Otherwise the soft routers consume large area, yet they give more flexibility in the implementation process in silicon.

4.0.1.1 Router Architecture in the NoC hard implementations

A router with switched circuit is implemented due to its input buffer resources. Despite that these routers have poor ability in sharing resources, yet with wiring flexibility on the FPGA and known traffic patterns, this makes the designer able to solve this problem. To give flexibility in the used topology, any router port can be connected to any part on the FPGA through a configurable wiring.

The router is mapped to StratixII and and Virtex4 boards. In which, a 90 nm technology is used which enables to translate the configurable resources on the FPGA into area. The interconnections are classified into three types: short, medium and long. The design depends mainly on the medium length interconnections, that gives a good comparison basis.

The three evaluated implementations are shown in Figure 4.1.

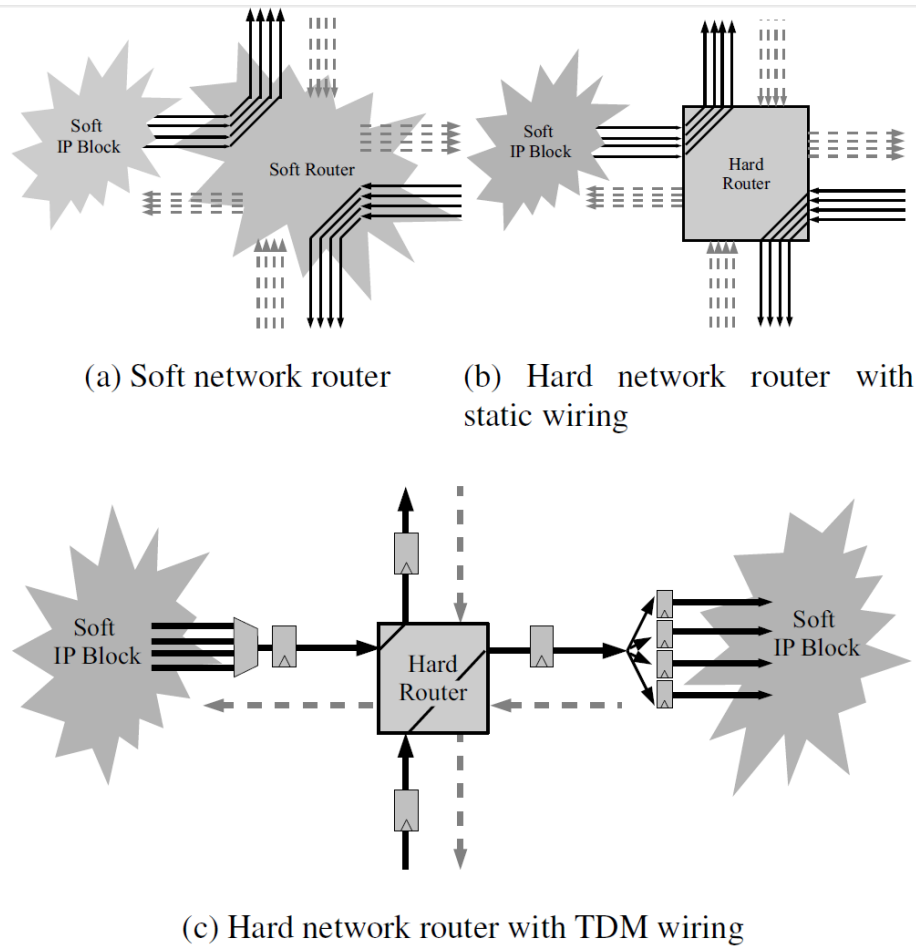


Figure 4.1: Three NoC implementations[5]

4.0.2 Area Results

The area is calculated to include the Look-Up table (LUT), router area and wiring area. The silicon area of each router is shown in Table 4.2.

Table 4.2: Router areas[5]

Router	Area (μm^2)
Soft 32 bit NoC StratixII	125 077
Soft 32 bit NoC Virtex4	126 000
Hard 32 bit NoC Static	9 549
Hard 8 bit NoC TDM	5 757

The hard routers area will vary more while using different topologies as they consume a smaller proportion of the overall network area than the soft routers. The percentage of network area which is consumed from the whole chip area for different routers are shown in Figure 4.2.

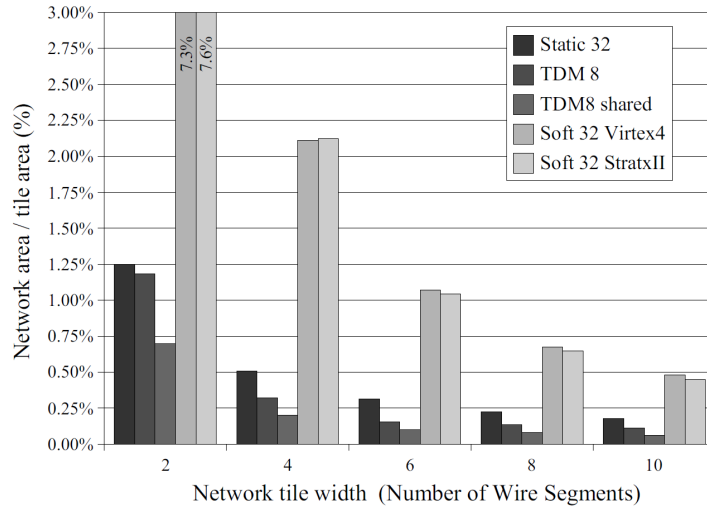


Figure 4.2: Percentage of router area to chip area[21]

The hard and soft router consume 1% and 10% of their network areas respectively.

4.0.3 Power Results

The power is evaluated by adding two data circuits across the router and transmitting router data into the network. Figure 4.3 shows the power consumption which is consisted of two parts: router power consumption and wiring power consumption. The soft NoC consumed power which is greater than the hard NoC with TDM wiring by four times.

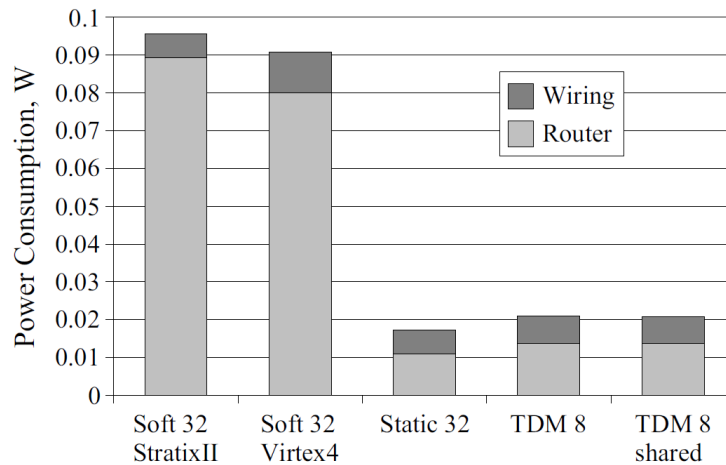


Figure 4.3: Hard NoC and Soft NoC power consumption[21]

As a conclusion to the study above, the soft NoC uses less than three percent of the whole chip area. While the hard NoC does not exceed one percent. The cost of the un-used hardware has been minimized by using the TDM wiring approach.

Regarding the power consumption, the soft NoC shows very bad performance while comparing it with hard NoC. Most of FPGAs use the hard NoC implementation for its better performance regarding the area and power aspects.

4.1 Impact of NoC parameters on the FPGA Network-On-Chips (NoCs)

Developing a method for evaluating the effect of the fundamental NoC parameters in the FPGA based NoC, gives the designer a more clear picture. The effects of buffer flit depth, data flit width and the virtual channels parameters are evaluated[36].

The study shows that the data flit width and buffer flit depth have the greatest effect on the FPGA area and clock frequency. Meanwhile, keeping the virtual channels equal to four provides the best performance with good efficiency for the mesh and torus topologies.

4.2 A NoC design with debug features on Field-Programmable Gate Array (FPGA)

The NoC design on the FPGA comes with many tradeoffs, the monitoring of all variable NoC parameters facilitate finding the optimal intersection point between all aspects that suits each application or design.

SonicsGN is a NoC that enables high speed networks and it is used in vital applications including smart phones, battery and home devices as shown in Figure 4.4.

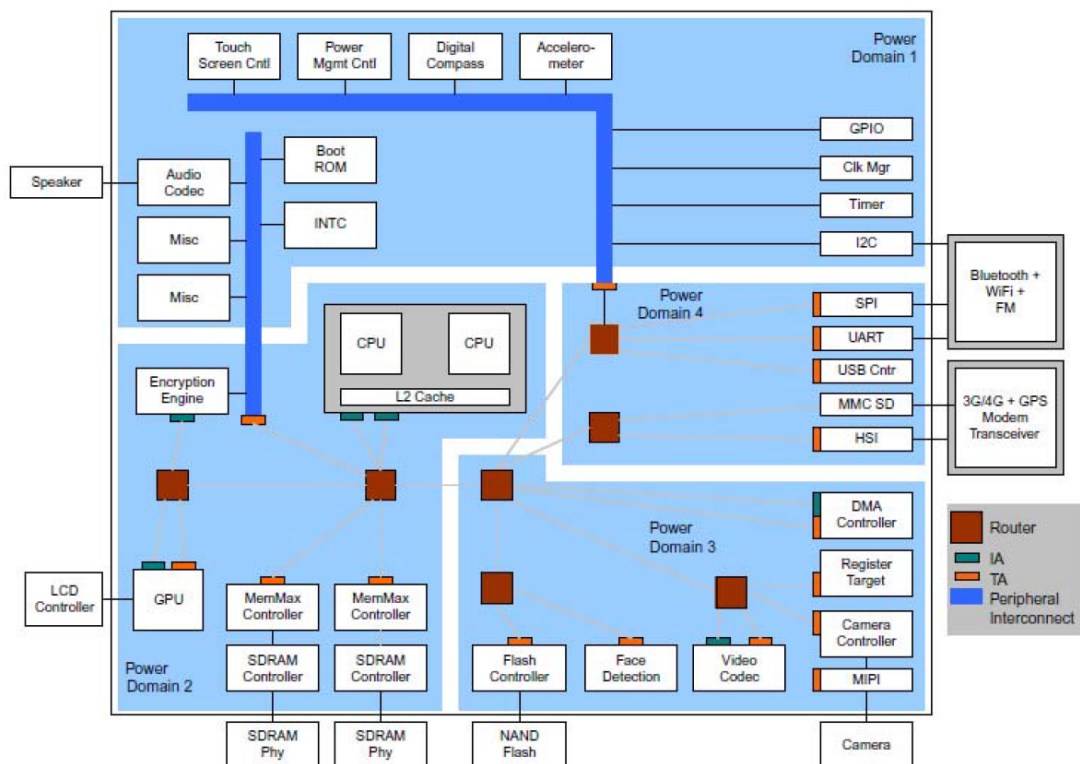


Figure 4.4: SonicsGN NoC Architecture [22]

Additionally to the NoC architecture, a performance monitor and hardware trace module is implemented[22]. The performance monitor tools provides a system visibility and debugging environment across the network interconnections. The tool visualizes all the interactions which occue on the chip. Moreover, the system utilization and the latency can be measure during the runtime.

The performance tools can be located in the SonicsGN NoC as shown in Figure 4.5.

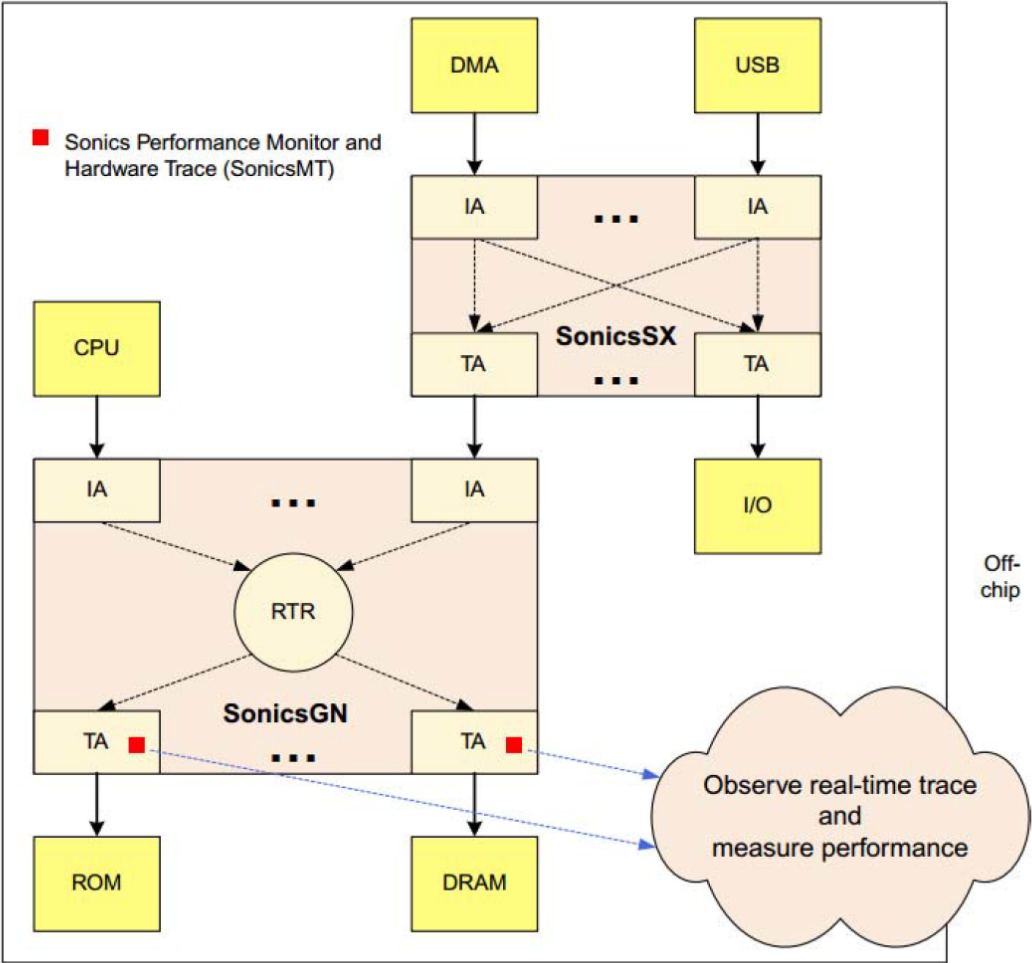


Figure 4.5: The location of the performance tool in the SonicsGN NoC[22]

While the performance monitoring tool architecture can be shown in Figure 4.6.

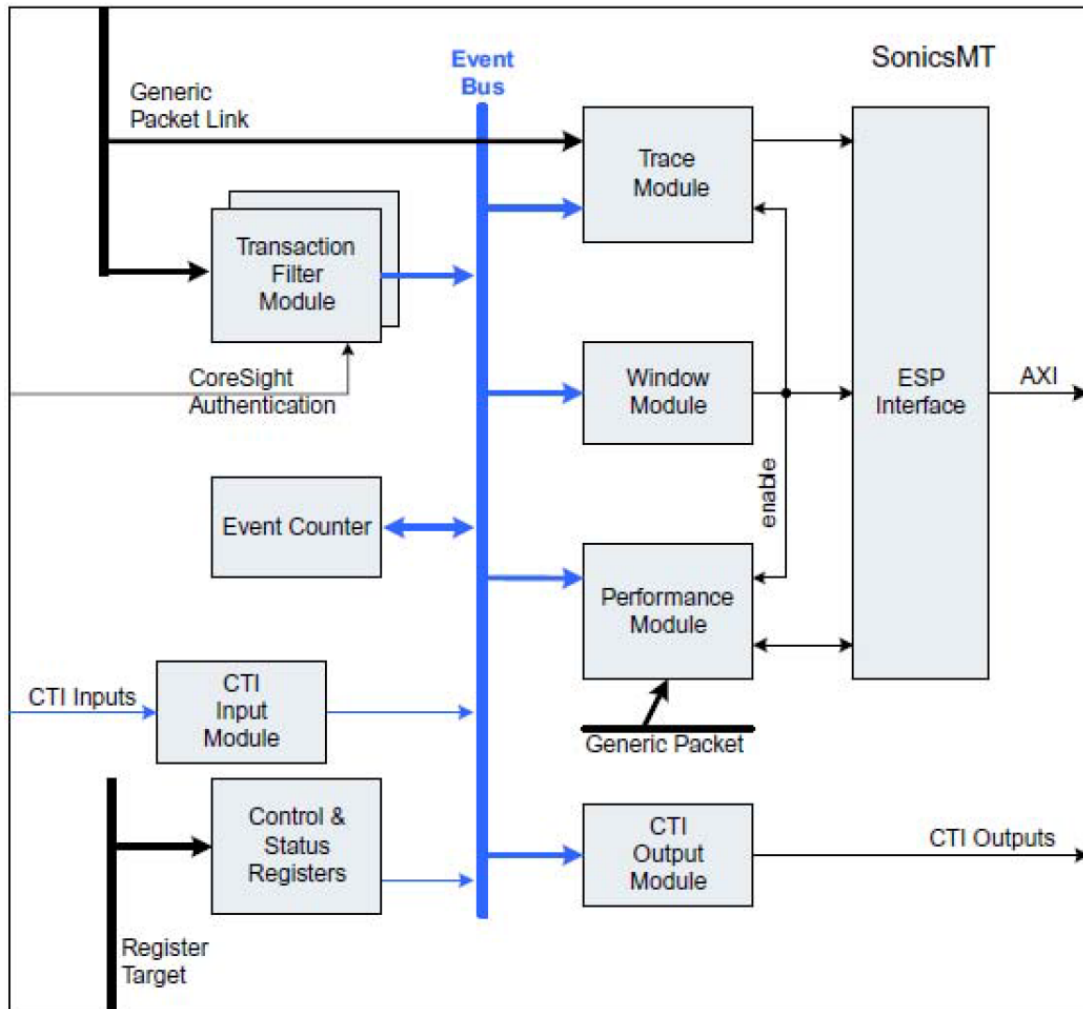


Figure 4.6: The Architecture of the performance and monitoring tool[22]

The performance and monitoring tool can work in several working modes as follows:

- Enable phase.
- Program phase.
- Run phase.
- Post process phase.

As illustrated in this chapter, the FPGA based on NoC methodology solves many design challenges, but it comes with another challenges and tradeoffs. Therefore, many router architectures are evaluated and also many studies have been conducted on the different network metrics and their effects on the FPGA performance regarding the area and power utilizations.

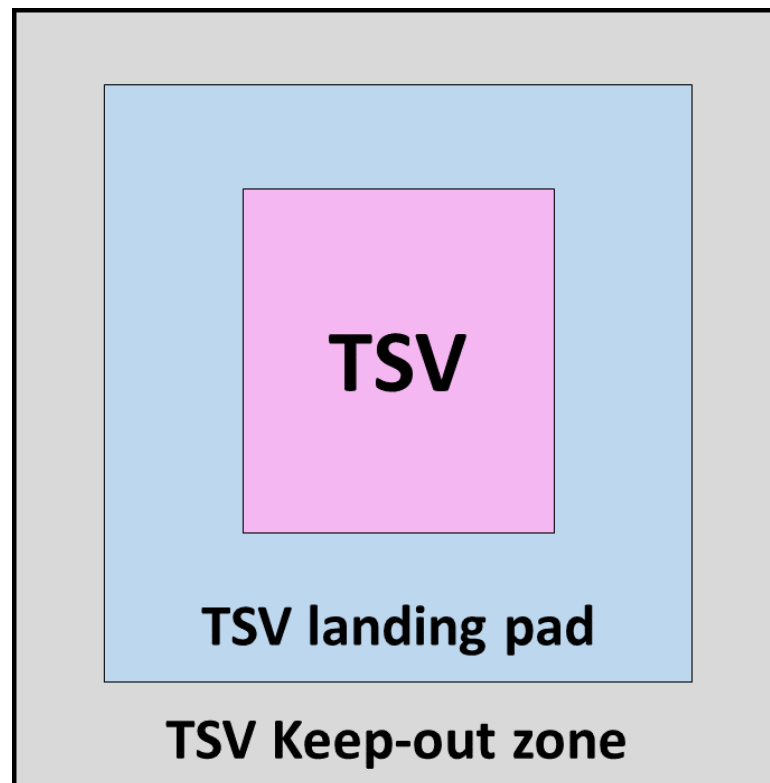


Figure 5.2: TSV A cell occupying three standard cell rows (KOZ = 1.205 m) and TSV B cell occupying four standard cell rows (KOZ = 2.44 m)[24]

Secondly, The cost of manufacturing a 3D integrated circuit using TSV technology is relatively high. Also, each extra fabrication step introduces a high risk for design defects, that results in yield reduction in the fabrication process. The yield is an exponential function of the number of TSVs and the frequency of defects. Thus, the yield exponentially decreases when the number of TSVs increases beyond a certain limit.

The cost efficiency of the manufacturing process introduces an essential tradeoff between the short and fast TSV interconnects, and a constraint on the number of possible fabricated TSVs in the 3D integrated circuit. As mentioned before in the first chapter, 3D Integration technology enables the integration of different fabricated integrated circuits with several technologies. Maintaining the same topology for all the vertical tiers of such dies and using a regular 3D network topology is absolutely hard. To support the network topology diversity of the design tiers and reduce the number of TSVs needed by the network to keep the yield efficiency at its acceptable limit.

A 3D dead-lock routing algorithm, denoted by "Elevator-First Algorithm" has been implemented to connect a 2D chips vertically and partially[37]. The 2D chips are connected in the planar level with usual topologies such as mesh topology, star topology, ..etc. The goal of the design of such a 3D routing algorithm is not to determine a fixed number of vertical connections TSVs or even their locations in the design.

This flexibility gives the designer the ability to determine the most convenient design and topology of each tier, also the number and location of TSV interconnects. In addition to the routing algorithm, a 3D router has been implemented to support the functionality of the 3D routing algorithm Elevator-First.

5.2 Elevator-First Three Dimensional Routing Algorithm

As mentioned before the Elevator-First algorithm is independent on the number and locations of the vertical interconnects (TSVs), also the planar topology used in each tier. That allows inhomogeneous network tiers to be combined in one 3D intergrated chip. Additionally, the routing decision in the Elevator-First routing algorithm is taken based on the final destination which is available in the header of the transmitted packet and the current node. There are two possibilities of the routing in the system. First, if the current node and the destination node are in the same planar tier, in this case a usual 2D routing algorithm such as mesh will be used to route the packet. Second, the current node and the destination nodes are not located in the same tier, in this case the usual 2D routing algorithm is not applicable anymore.

The routing mechanism in this case works as follows: if a router issues a packet which has a destination node located in a different tier, the router adds a new header to the packet that includes the address of the elevator 3D router and a flag to indicate that the elevator router is not the final destination node of this packet. The location of the elevator router is read directly from a register. The elevator flag is tested when the router is the destination. If it is set, the current router is only an intermediate node between the source and destination nodes and not the final packet target.

Then the extra header will be deleted and the packet will be transmitted through a vertical link with its original header. Eventually, if the elevator flag is not set, the current router will be the packet target and will consume it. The Elevator-First routing algorithm is implemented with two virtual channels per one link to provide a deadlock freedom in the design. Besides the implementation and design of the Elevator-First, a 3D router has been implemented to provide the essential hardware, as shown in Figure 5.3, that needed to support the routing algorithm. The router comes with a low overhead in comparison of a router in a fully mesh connected 3D-NoC[9].

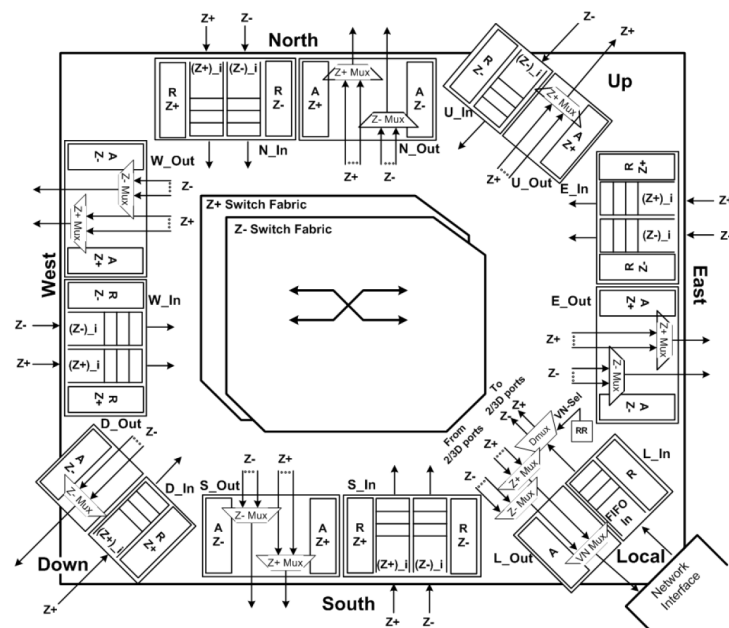


Figure 5.3: Elevator-First 3D Router[9]

In the thesis work, a 3D routing algorithm, Direct-Elevator, which is based on the Elevator-First routing algorithm is proposed[25]. The Direct-Elevator is also independent of the network planar topology, number of the vertical interconnects and placement of them. The Direct-Elevator has all the benefits of the Elevator-First algorithm, but with a low latency approach.

5.3 The Direct-Elevator Three Dimensional Routing Algorithm

The Direct-Elevator algorithm also operates on partially vertically connected 2D chips. The routing mechanism of the Direct-Elevator is slightly different from the routing mechanism of the Elevator-First algorithm which leads to an optimization in the communication latency between the tiers in the 3D system.

The Elevator-First algorithm relies on elevator nodes to connect the vertical tiers. In order to transmit a packet from a node to another node which is located in a different tier, the packet needs to be transmitted to the elevator node first. To illustrate and elaborate more on the difference between the two routing mechanisms, a full routing path will be followed using both routing algorithms.

5.3.1 A packet routing path using Elevator-First mechanism

A packet path is from the source node S1 to the destination node D3 as shown in Figure 5.4. When a source issues a packets, it firstly checks if the destination is located with it in the same planar tier or not. Accordingly, if the destination and source nodes are in the same tier, the source routes the packet directly using the 2D topology which is a fully connected mesh in this example. In this case, S1 and D3 are located in two different tiers. Hence, S1 routes the packet to the elevator router E1. Additional flit will be added to the packet including the address of E1 router. A T flag will be set to indicate whether the elevator is the packet final target or not. Also, a U flag will be set indicating whether the packet is ascending or descending in the 3D system.

Then, the packet reaches the intermediate adjacent tier through an intermediate node M. Moreover, M acts as a new source and the previous steps will be repeated again. In this routing scheme, a packet cannot be routed directly from the source node S1 to the destination node D3. It has to go through multiple nodes in between the routing path.

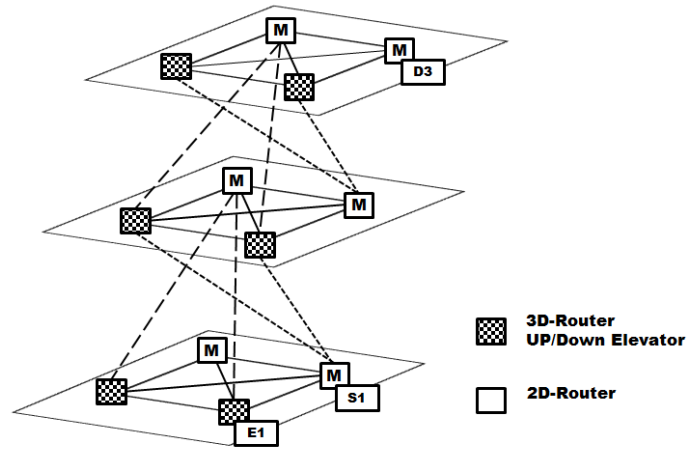


Figure 5.4: Elevator-First routing mechanism[25]

5.3.2 A packet routing path using Direct-Elevator mechanism

The Direct-Elevator algorithm is based on a smarter approach that has an effective impact on the system latency consumed in the packets routing. In case of routing a packet from S1 to D3 as shown in Figure 5.5. Once S1 issues a packet, it decides whether the destination is on the same tier or not. If it is not as in this case, it routes the packet to the elevator router. The elevator router then directly routes the packet to the tier in which the destination node is located.

The elevator nodes in the Direct-Elevator algorithm are directly connected to form a real elevator. Therefore, the Direct-Elevator algorithm saves the time of adding and removing the temporary elevator header flit to the original packet. Regardless of the location of the destination tier or how far it is from the source tier, this procedure is done once during the routing path.

Accordingly, the Direct-Elevator routing algorithm can be simply considered as a special case tailored from the Elevator-First routing algorithm.

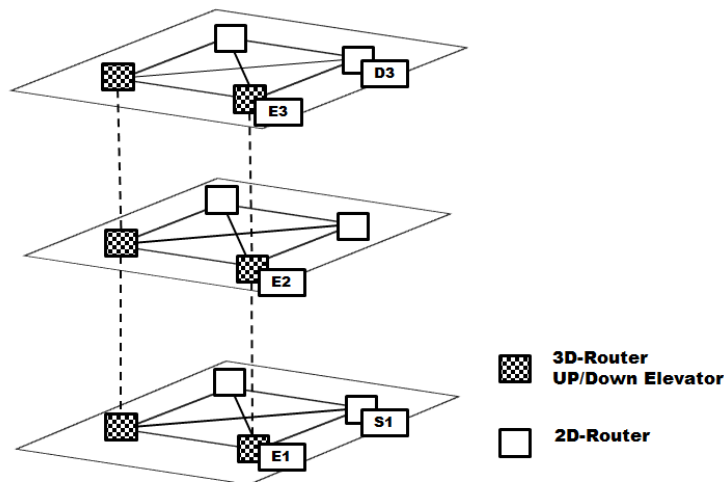


Figure 5.5: Direct-Elevator routing mechanism[25]

5.4 Comparative Performance Analysis

To compare the performance between the Elevator-First and Direct-Elevator algorithms, comparative experiments were conducted. For a reliable performance evaluation and fair efficient comparison, the environment has been unified represented in the following toolset:

- C programming language
- GNU Compiler Collection (GCC) under Cygwin

Each experiment measures the network throughput in the following two cases:

1. Random packet transmission.
2. Packet transmission over the worst network path.

The network throughput is defined as the successful received packets per second. While the network worst path is defined as the longest path between two nodes in the 3D-NoC system. For each conducted experiment the impact of network load, the vertical complexity and the tier complexity on the network throughput by varying three aspects respectively which are:

- The number of transmitted packets across the 3D-NoC.
- The number of tiers in the 3D-NoC.
- The number of routers per each tier in the 3D-NoC.

5.4.1 Network Load

The network load is evaluated by testing the 3D-NoC while increasing the number of transmitted packets across the system. The packets are transmitted in the network at random basis and over the network's worst path. In these simulations, the number of routers per each tier and number of tiers is fixed to four.

As the network load is only tested in this case, the other metrics need to be fixed. Figure 5.6 shows the throughput of the two algorithms over a random range of transmitted packets which varies from 50,000 to 200,000 packets. Additionally, Figure 5.7[25] shows the throughput of the two algorithms over the same range of the packets over the network's worst path.

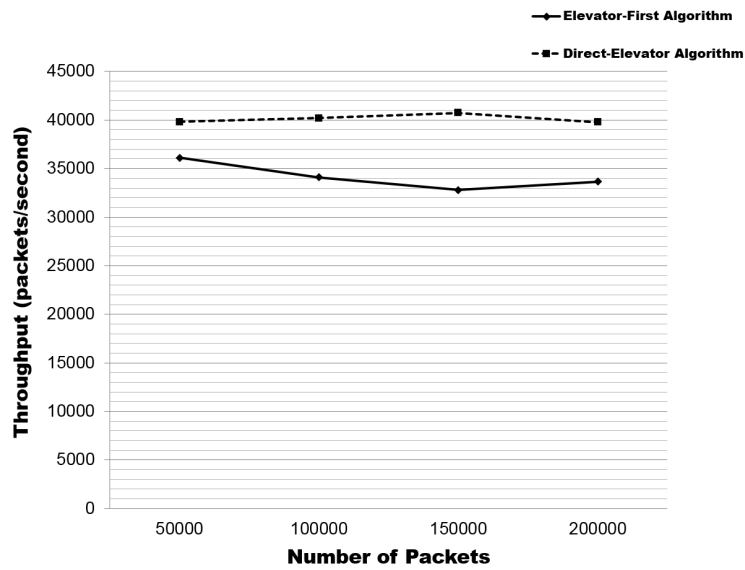


Figure 5.6: Throughput Vs. Packets (random test case)[25]

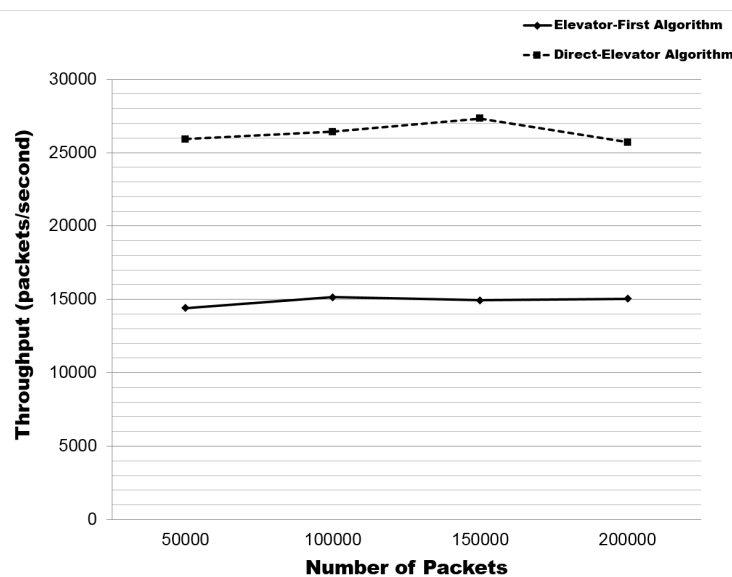


Figure 5.7: Throughput Vs. Packets (worst path test case)[25]

In Figure 5.6 and Figure 5.7[25], the network throughput of the two algorithms changes slightly over the range of the transmitted packets. Hence, this behaviour reflects the stability offered by the two algorithms.

5.4.2 Vertical Complexity

The vertical complexity measures the impact of increasing the number of tiers in the 3D-NoCs on the network throughput. Figure 5.8 shows the throughput of the two algorithms while changing the number of tiers from three to six.

In these simulations, the random network transmitted packets and number of routers per each tier are fixed to 100,000 and four, respectively. On the other hand, Figure 5.9 shows the throughputs of the two algorithms over a range of tiers varies from three to six. The number of random transmitted packets over the worst path is 100,00, while the number of routers per each tier in the 3D-NoC is four.

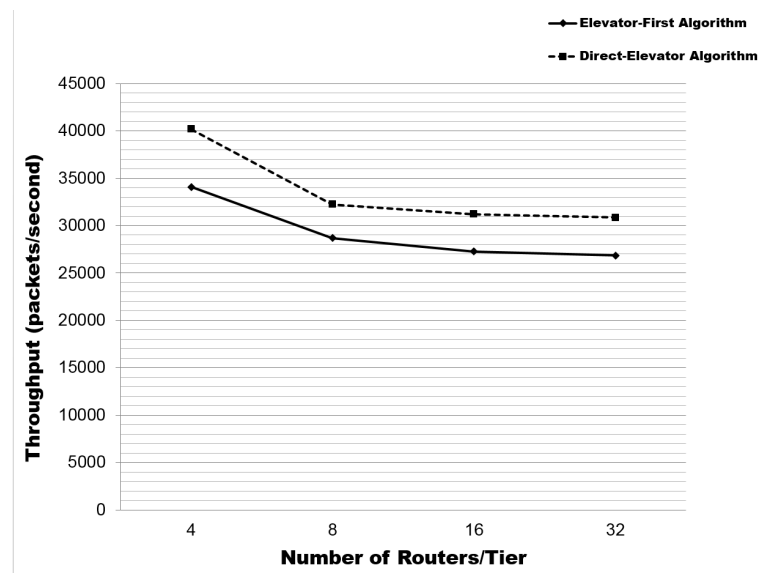


Figure 5.8: Throughput Vs. Tiers (random test case)[25]

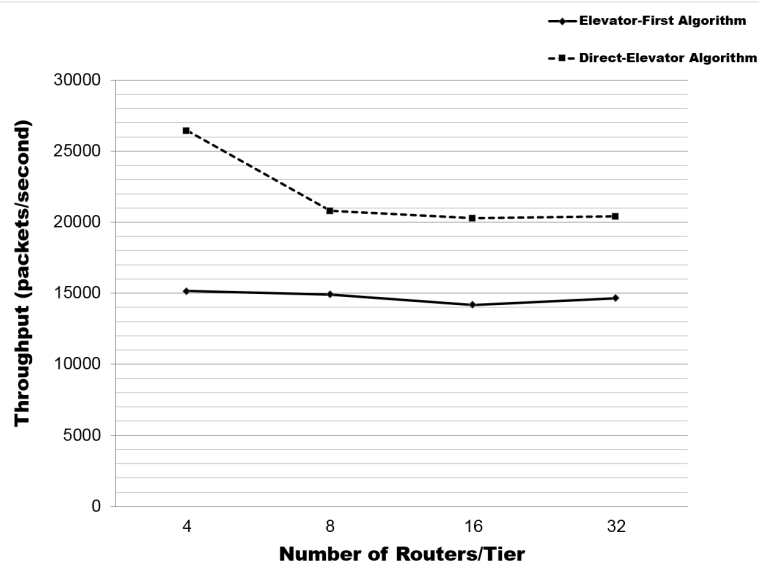


Figure 5.9: Throughput Vs. Tiers (worst path test case)[25]

In Figure 5.8, the throughput decreases while increasing the number of tiers in the 3D-NoC. As while increasing the number of tiers in the system, the vertical interconnects become longer which causes this significant drop in the network throughput. Moreover in Figure 5.9[25], the throughput decays more as all packets are routed across the worst path.

5.4.3 Tier Complexity

The tier complexity measures the impact of varying the number of routers per each tier on the throughput. In the 3D-NoCs, the routers per tiers can be distributed regularly or hierarchically.

5.4.3.1 Regular Distributed 3D-NoC

A regular distributed 3D-NoC has the same number of routers per each tier in the system. Figure 5.10 shows the throughputs of the two algorithms over a range of routers per each tier from four to 32. Here in these simulations, the random transmitted packets are fixed to 100,000 and the number of tiers in the 3D-NoC is equal to four. Additionally, Figure 5.11[25] shows the two throughputs over a varying range of routers per tier from four to 32. Here, the number of transmitted packets over the worst path are 100,000, while the number of system tiers are fixed to four.

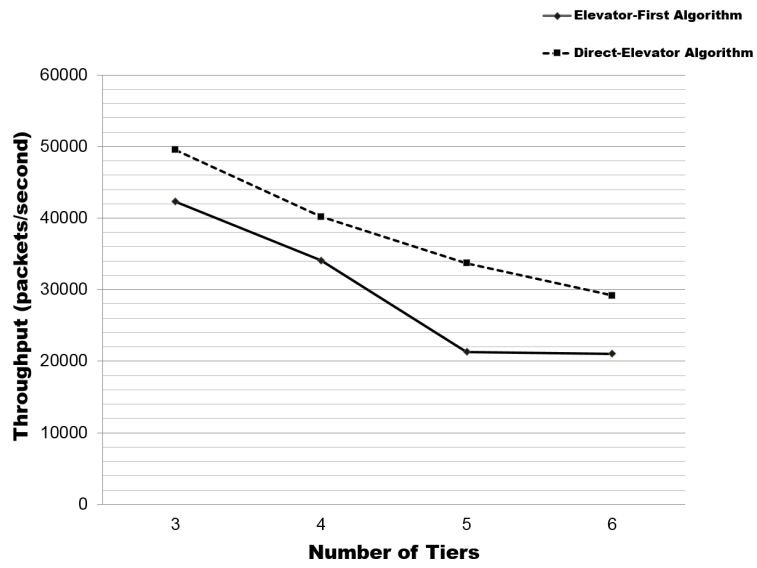


Figure 5.10: Throughput Vs. Routers/Tier (random test case)[25]

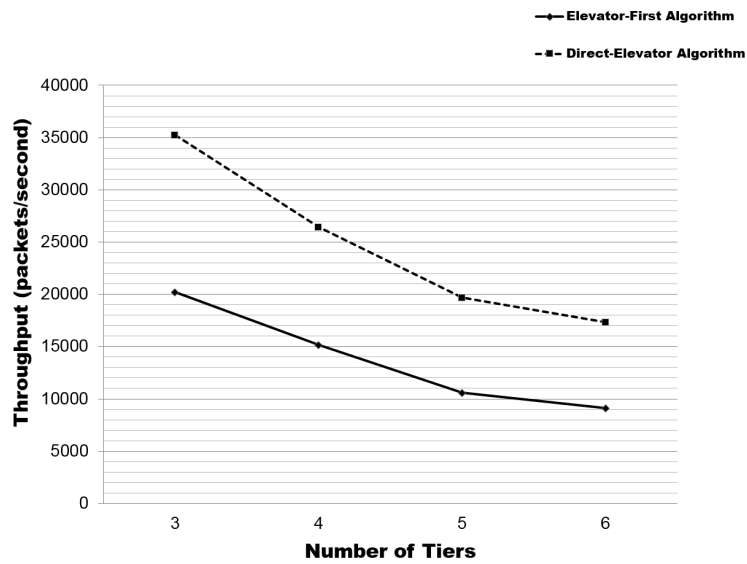


Figure 5.11: Throughput Vs. Routers/Tier (worst path test case)[25]

As shown in Figure 5.10 and Figure 5.11, the throughput changes slightly with increasing the number routers per each tier.

5.4.3.2 Hierarchical Distributed 3D-NoC

In a hierarchical structure of a 3D-NoC contains four tiers, the routers are distributed as four routers in the first tier, eight routers in the second tier, 16 routers in the third tier and 32 routers in the fourth tier as shown in Figure 5.12. In this case, the transmitted random packets are 100,000. The throughput of the Direct-Elevator is measured to be higher than the throughput of the Elevator-First algorithm at the same conditions by 5.3 percent. On the other hand, while transmitting the same number of packets over the worst path, the throughput of the Direct-Elevator algorithm is higher than the Elevator-First algorithm with 43 percent.

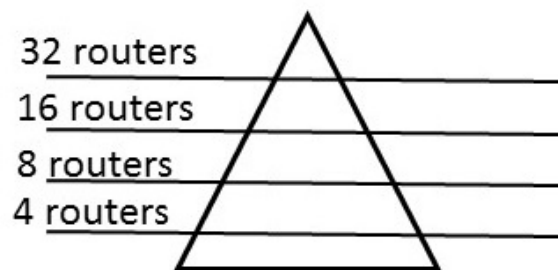


Figure 5.12: Hierarchical Structure of a 3D-NoC example

5.5 Summary

The two algorithms are implemented using the same software environment. The Direct-Elevator has been proved to have a better performance in the latency aspect than the Elevator-First algorithms. The Direct-Elevator saves the processing time which is taken by the system to add and remove extra temporary header, to differ between the intermediate elevator node and the destination node that leads to a significant optimization in the network latency.

The number of the elevators in the Direct-Elevator algorithm can be definitely adjusted according to the requirements of each application[25].

Chapter 6: 3D-NOCET: A Tool for Implementing 3D-NoCs based on Direct-Elevator Algorithm

6.1 Introduction

Implementing the flexible 3D routing algorithm, Direct-Elevator, paves the route to implement a tool which provides different configurational 3D-NoCs structures. The proposed tool denoted by 3D-NOCET which is based on the Direct-Elevator, enables the user to create different structures of the 3D-NoCs based on the number of routers per tier, number of tiers and the 2D network topology. Experimental evaluations and performance analyses then will be conducted to determine which is the most convenient 3D-NoC configuration for the application.

The 3D-NOCET tool offers diverse synthesizable configurational 3D-NoCs through a fully automated processes. The user is able to create different 3D-NoCs structures by choosing the number of routers per each tier, the number of tiers and the 2D network topology.

Many tools were implemented to provide the users with different 3D-NoCs, but most of them were providing a SystemC designs like Noxim and NIRGAM. 3D-NOCET tool provides Register-transfer level (RTL) designs for its 3D-NoCs. Therefore it allows the user to evaluate the 3D-NoC with respect to the area, power and latency which gives the complete picture of the most important design tradeoffs.

The tool works through an easy-to-use Graphical User interface (GUI). The tool GUI allows the designer to select the number of routers per each tier separately, the number of tiers and the 2D topology whether it is mesh topology or ring topology. The tool then creates the RTL design files for the 3D-NoC system according to the application specifications through an automation implemented infrastructure. Figure 6.1 shows the GUI of the 3D-NOCET tool, the user in this case selects a 3D-NoC structure with the following specifications:

- Four routers per each tier.
- Four tiers in the 3D-NoC structure.
- Mesh topology as the 2D routing topology.



Figure 6.1: The Graphical User interface of the 3D-NOCET tool

The tool has been implemented and developed with hardware limitations as follows:

- Maximum number of routers per tier equals to 255.
- Maximum number of tiers in the 3D-NoC equals to 16.
- Minimum number of tiers in the 3D-NoC equals to two.
- Two 2D topologies, mesh and ring.

The tool has been implemented using the following toolset and environmental settings:

1. TK/TCL.
2. Shell scripting.
3. Red-Hat 5 Linux machine- 64bits.

The tool implemented all the 3D-NoCs with two vertical interconnections that connect all the elevators together across the system's tiers. The vertical interconnects route the packets directly through the system in the 3D dimension. Moreover, the 2D routing mechanism varies according to the supported network topology in this case whether it is mesh topology or ring. The tool provides two main categories of the 3D-NoCs as shown in Figure 6.2 and in Figure 6.3.

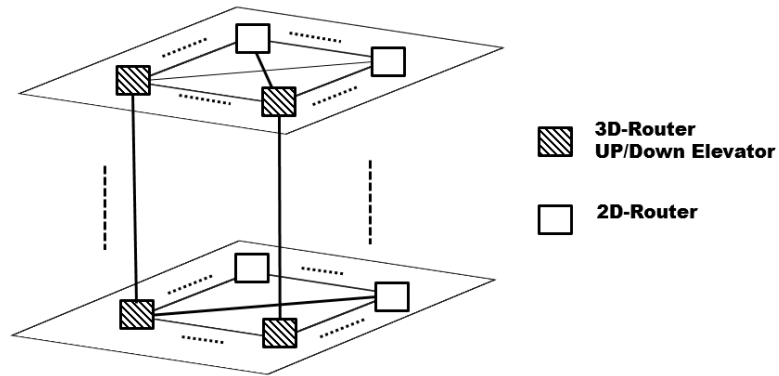


Figure 6.2: The 3D-NoC in 2D Mesh Topology

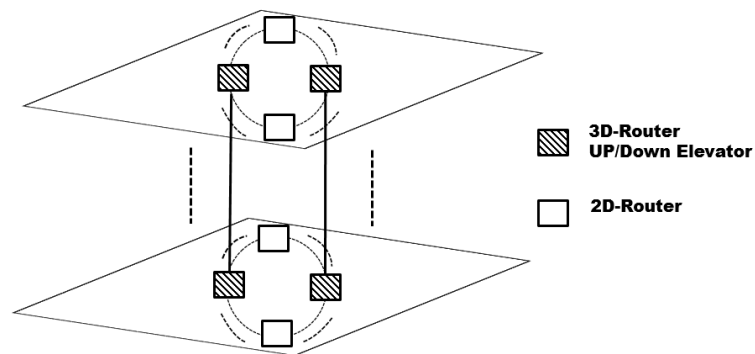


Figure 6.3: The 3D-NoC in 2D Ring Topology

6.2 The 3D-NOCET Tool User Guide

In this section, the flow of all the automation infrastructure and the System Verilog designs within the tool implementation and development will be illustrated. The tools mainly contains two components as following:

1. Automation infrastructure that has been implemented using different scripts to guarnantee the flexibility of creating different configurational 3D-NoCs.
2. A 3D router design that has been implemented to support the Direct-Elevator routing algorithm functionalities, and also the two 2D topologies (mesh and ring).

6.2.1 Automation Infrastructure Scripts

The automation infrastructure is based on different developed scripts to provide the 3D-NOCET tool with the ability to create generic and flexiabile configurational range of 3D-NoCs at a high speed and efficiency.

As the number of routers per tier, the number of tiers per 3D-NoC and the 2D topology change, the implementation of the connections certainly change.

The automation infrastructure contains the following scripts:

- Startup.tcl: a TK/TCL script that has been created to implement an easy friendly Graphical User Interface (GUI) for the tool as shown inFigure 6.1.

- `Master_Script.sh`: a shell script that has been created to call either the Mesh or Ring flow scripts according to the selected topology by the user.
- `Mesh_RoutingTables_Add.bash`: a shell script which is responsible of creating the addresses for all the routers in the network, the mesh connections between them in the 2D and 3D levels, and also routing tables.
- `Mesh_Topology.sh`: a shell script which is responsible of creating the right syntax of instantiations of the router module with mesh topology then append them in the design top module file. Also, the parameters in the parameters header file according to the user entries.
- `Ring_RoutingTables_Add.bash`: a shell script which is responsible of creating the addresses for all the routers in the network and also the mesh connections between them in the 2D and 3D levels.
- `Ring_Topology.sh`: a shell script which is responsible of creating the right syntax of instantiations of the router module with ring topology then append them in the design top module file. Also, the parameters in parameters header file according to the user entries. The ring routers have no routing tables.
- `Final_msg.tcl`: a tcl script to display the final information (info) message after creating the design is done as shown in Figure 6.4.



Figure 6.4: The 3D-NOCET tool final information message

6.2.2 Register Transfer Level Design files

The design files are created to support the user specifications dynamically. The 3D router support the logic of a mesh or a ring 2D topology, and also the Direct-Elevator routing mechanism. The design files implemented for the 3D-NoC are as follows:

- `param.h`: the parameters header file which includes the initial parameters of the design files. This file is automatically editable according to the user entries by the automation infrastructure flow.
- `topmod.sv`: the design top module that includes the router instantiations. This file is automatically editable according to the user entries by the automation scripts above.

- testBn.sv: a test bench to test the network in the Questa simulations by sending packets from a reserved address all ones.
- router_module_mesh.sv: the Mesh router module design.
- router_module_ring.sv: the ring router module design.

6.2.3 The 3D-NOCET tool implementation and execution flow

The automation infrastructure scripts work in a certain flow to create the 3D-NoC design files to match the user selected configurations. In Figure 6.5, a flow chart for how the 3D-NOCET tool is shown and illustrated to elaborate more on how and when every script is used to create the 3D-NoC design files.

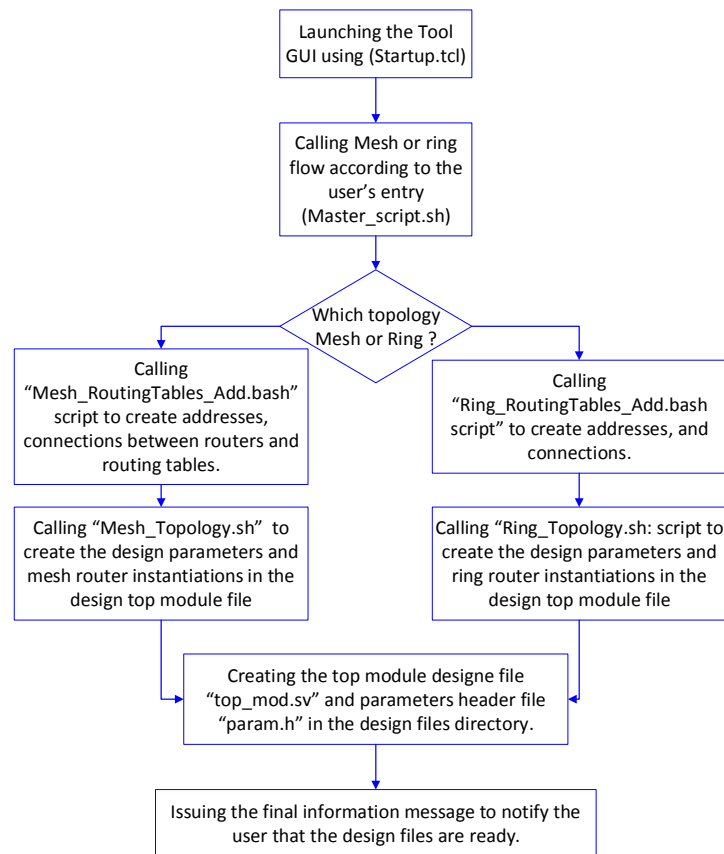


Figure 6.5: The flow chart of the 3D-NOCET tool implementation

The flow chart consists of the following items:

- Launching the tool GUI using the Startup.tcl script: The Startup.tcl script contains the structure of the GUI main window. It also shows all the possible selections of mesh and ring topologies as shown in Figure 6.6.

```
#!/usr/local/bin/wish
lappend auto_path [file dirname [info script]]
package require Tk
package require BWidget
#Main Window size, title and formatting
wm title . "3D NoCs Design"
set x [expr { ( [wininfo vrootwidth .] - 800 ) / 2 }]
set y [expr { ( [wininfo vrootheight .] - 800 ) / 2 }]
wm geometry . 800x800+$x+$y
wm protocol . WM_DELETE_WINDOW {
    destroy .
}
set family Courier
set font "$family 14 bold"
set font_small "$family 12 bold"
frame .home -borderwidth 10 -width 500 -height 500
pack .home -side top -fill both

label .home.topo -font $font -wraplength 4i -justify left -text "Select Topology:"
checkboxbutton .home.mesh -text "Mesh" -variable mesh -command {set ring 0}
checkboxbutton .home.ring -text "Ring" -variable ring -command {set mesh 0}
place .home.topo -x 0 -y 0
place .home.mesh -x 250 -y 0
place .home.ring -x 320 -y 0
```

Figure 6.6: The main GUI window structure and the different valid topologies

Then, The number of tiers and number of routers per each tier are implemented and placed in the main window within the script as shown in Figure 6.7.

```
label .home.text -font $font -wraplength 4i -justify left -text "Select Number of tiers:"
place .home.text -x 0 -y 50

frame .home.pad
grid .home.pad -column 0 -row 0 -pady 50
frame .home.pad1
grid .home.pad1 -column 0 -row 1

frame .home.tmp1
checkboxbutton .home.tmp1.tier1 -text "Tier#1" -variable tier1 -command {packmyrouters .home.tmp1 $tier1 1}
pack .home.tmp1.tier1
grid .home.tmp1 -column 0 -row 2 -pady 20
frame .home.tmp2
checkboxbutton .home.tmp2.tier2 -text "Tier#2" -variable tier2 -command {packmyrouters .home.tmp2 $tier2 2}
pack .home.tmp2.tier2
grid .home.tmp2 -column 1 -row 2 -sticky nw -pady 20
frame .home.tmp3
checkboxbutton .home.tmp3.tier3 -text "Tier#3" -variable tier3 -command {packmyrouters .home.tmp3 $tier3 3}
pack .home.tmp3.tier3
grid .home.tmp3 -column 2 -row 2 -sticky nw -pady 20
frame .home.tmp4
checkboxbutton .home.tmp4.tier4 -text "Tier#4" -variable tier4 -command {packmyrouters .home.tmp4 $tier4 4}
pack .home.tmp4.tier4
grid .home.tmp4 -column 3 -row 2 -sticky nw -pady 20
```

Figure 6.7: The implementation of the functionality buttons

- Calling the mesh or ring flow according to the user's entry: The "Master_Script.sh" is implemented to choose between the two different possible flows either mesh or ring. It also parses the user entries and evaluate the number of routers per each tier and adds the parameters in the header file. Then it calls the group of scripts that matches either the ring and mesh topology as shown in Figure 6.8, Figure 6.9 and Figure 6.10.

```
#####
###Clean up
rm -rf no_routers_tiers.txt
#user entries
#Number_of_Routers_per_tier=("$@")
Number_of_Routers_per_tier=();
in=0;
topology="";
for i in $1
do
  if [ $i == "Mesh" ] || [ $i == "Ring" ]
  then
    topology=$i;
    break;
  fi
  Number_of_Routers_per_tier[in]=$i
  in=$((in+1));
done
```

Figure 6.8: Selection of the topology type

```
#Creating the parameters of routers per tier in the param.h file
param_file=Design_files/param.h
count_tier=1;
x=0;
for x in "${Number_of_Routers_per_tier[@]}"
do
  echo "\define NO_OF_ROUTERS "$count_tier"_TIER "$x" >> no_routers_tiers.txt
  count_tier=$((count_tier+1))
done
sed -i '/routers_tier/,/end_tiers/{//!d}' $param_file
sed -i '/routers_tier/rno_routers_tiers.txt' $param_file
rm -rf no_routers_tiers.txt
#####
```

Figure 6.9: Evaluating the parameters in the header file


```

if [ $topology == "Mesh" ]
then
#SOURCE scripts of Mesh topologies to create the topmod.sv and param.h files
. Mesh_RoutingTables_Add.bash "${Number_of_Routers_per_tier[@]}"
. Mesh_Topology.sh "${Number_of_Routers_per_tier[@]}"
elif [ $topology == "Ring" ]
then
#SOURCE Sscripts of Ring topologies to create the topmod.sv and param.h files
. Ring_RoutingTables_Add.bash "${Number_of_Routers_per_tier[@]}"
. Ring_Topology.sh "${Number_of_Routers_per_tier[@]}"
else
echo "Please enter a valid topology for the system Mesh or Ring."
fi
#cleaning up the working directory
rm -rf routing_tables.txt
rm -rf addresses.txt

```

Figure 6.10: Determining the script flow according to the topology type

- Creating the implementation according to the mesh topology including the router instantiations and other design parameters: The "Mesh_Topology" script first calculates the number of total wires needed by the system according to the user's entries which are represented in the number of tiers and the number of routers per each tier as shown in Figure 6.11.

```
#####
#Counting all the wires required in the system
design_file=Design_files/topmod.sv
param_file=Design_files/param.h
Number_of_wires_in_system=0;
Number_of_routers=0;
total_routers=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    Number_of_wires_in_system=$((2 + (($Number_of_routers - 1) * $Number_of_routers) / 2) + $Number_of_wires_in_system))
    total_routers=$((total_routers+$Number_of_routers))
done
Number_of_wires_in_system=$((Number_of_wires_in_system - 2))
echo "\define NUM_OF_WIRES \"$Number_of_wires_in_system\"" >> wires_number.txt
total_routers=$((total_routers-1))
echo "output [63:0] R_packet [\"$total_routers\":0];" >>routers_total.txt
#Append here number of wires to the param.h file and all router number
sed -i '/wires/,/end_here/{/!d}' $param_file
sed -i '/wires/wires_number.txt' $param_file
sed -i '/r_packet/,/end_r/{/!d}' $design_file
sed -i '/r_packet/routers_total.txt' $design_file
rm -rf wires_number.txt
rm -rf routers_total.txt
#####
for (( n = 0 ; n < $Number_of_wires_in_system ; n++ ))
do
    wires[$n]=$n
done
#####
```

Figure 6.11: Evaluating the number of wire in the 3D-NoC system using mesh topology

Additionally, the script then creates the connection between the routers to match the selected topology, in this case the mesh topology as shown in Figure 6.12. Then Figure 6.13, shows the instantiations of the mesh routers to be then added in the SystemVerilog file.

```

#2D_connections
for (( j = 0 ; j < $Number_of_routers - 1 ; j++ ))
do
    k=$((j+1))
    for (( k = $j+1 ; k < $Number_of_routers ; k++ ))
    do
        printf "R[$j] $tier_count links["$wires[$m]"],\n" >> connections.txt
        printf "R[$k] $tier_count links["$wires[$m]"],\n" >> connections.txt
        m=$((m+1))
    done
done
done
#####
#3D_connections
for (( l = 1 ; l < $tier_count ; l++ ))
do
    printf "R[0] $l links["$wires[$m]"],\n" >> connections.txt
    printf "R[0] $((l+1)) links["$wires[$m]"],\n" >> connections.txt
    printf "R[1] $l links["$wires[$((m+1))]"],\n" >> connections.txt
    printf "R[1] $((l+1)) links["$wires[$((m+1))]"],\n" >> connections.txt
    m=$((m+2))
done
done
#####
#Preparing the final connections
tier_count_1=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    tier_count_1=$((tier_count_1+1))
    for (( h = 0 ; h < $Number_of_routers ; h++ ))
    do
        grep -w "R\[\"$h\" \] \"$tier_count_1\"" connections.txt | awk '{print}' ORS=' ' >> inter.txt
        echo $'\n' >> inter.txt
        sed -i 's/, $//' inter.txt
        sed -i '/^$/d' inter.txt
        sed -i 's/R\[\"[0-9]\" \][0-9]* \\ /g' inter.txt
        sed -i 's/ //g' inter.txt
    done
done
#####
#Rearranging each line from MS to LS
while read line; do
    conct=$line
    IFS=', ' read -a array <<< "$conct"
    length=${#array[*]}
    for ((i=${#array[@]}-1; i>=0; i--)); do
        printf "%${array[$i]}," >> inter_rev.txt
    done
    echo $'\n' >> inter_rev.txt
    sed -i 's/, $//' inter_rev.txt
    sed -i '/^$/d' inter_rev.txt
done < inter.txt
#####

```

Figure 6.12: Creating the connections between the routers in the mesh topology

```

#####
#Adding the addresses to an array
IFS='\\n' read -d '' -r -a addresses < addresses.txt
#Preparing the final version of the mesh_connections for each router per each tier with Verilog required syntax
lines_count=0
tier_count_0=0
a=0;
tiers=${#Number_of_Routers_per_tier[@]}
while read line; do
  lines[$a]=$line
  a=$((a + 1))
done < inter_rev.txt
b=0
for i in "${Number_of_Routers_per_tier[@]}"
do
  tier_count_0=$((Tier_count_0+1))
  Number_of_Routers=$i
  for (( j = 0 ; j < $Number_of_routers; j++ ))
  do
    add=${addresses[$b]}
    if [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 == "1" ]
    then
      printf "Router_mesh #(.address(16'b$add"),.ramSize(\\NO_OF_ROUTERS_"$tier_count_0"_TIER)) router_"$b" (.port( {"${lines[$lines_count]}" } )
      .clk(clk),.packet(packet),.ram_values(ram_values_"$b"),.R_packet(R_packet [{"$b"}]);\\n" >> mesh_connections.txt
      printf "wire [15:0] ram_values_"$b" [\\NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\\n" >> wires_decl.txt
    fi
  done
done

```

Figure 6.13: The instantiations of the mesh routers

Eventually in the script, the instantiations are added in the top level design of the system as shown in Figure 6.14.

```

#####
#Append to the design file
sed -i '/ROUTER_INSTANCES/,/END_Routers/{//!d}' $design_file
sed -i '/Lookup_tables/,/end_luts/{//!d}' $design_file
sed -i '/Wires_declaration/,/end_wires/{//!d}' $design_file
sed -i '/ROUTER_INSTANCES/rmesh_connections.txt' $design_file
sed -i '/Lookup_tables/rrouting_tables.txt' $design_file
sed -i '/Wires_declaration/rwires_decl.txt' $design_file
#####
#Cleaning up
rm -rf connections.txt
rm -rf mesh_connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf addresses.txt
rm -rf routing_tables.txt
rm -rf wires_decl.txt
#####

```

Figure 6.14: Adding the mesh routers' instantiations into the SystemVerilog top design file

The "Mesh_Routing_Tables_Add.bash" script then will be called to create an address for each router in the system and its routing table as shown in Figure 6.15 and Figure 6.16.

```

#Converting from Dec to Bin.
Number_of_Routers_per_tier=("${#@}")
tiers=${#Number_of_Routers_per_tier[@]}
last_tier=$((tiers - 1))
D2B=({0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1})
last_tier_bin=${D2B[$last_tier]}
before_last=$((last_tier - 1))
before_last_bin=${D2B[$before_last]}
tier_count_0=0
#####
#Creating the addresses for all routers in the network
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    for (( j = 0 ; j < $Number_of_routers; j++ ))
    do
        D2B=({0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1})
        tier_no=${D2B[$tier_count_0]}
        address_no=${D2B[$j]}
        echo "$tier_no"$address_no" >> addresses.txt
    done
    tier_count_0=$((tier_count_0+1))
done
#####

```

Figure 6.15: Creating address for each router in mesh topology

```

#Creating the routing table for each router
count=0;
while read add; do
  rm -rf add_inter.txt
  most_add=${add:0:8}
  most_add_dec=$(echo "ibase=2;$most_add" | bc)
  les_add=${add:8:16}
  grep "^$most_add" addresses.txt >> add_inter.txt
  sed -i "/$add/d" add_inter.txt
#Check if the tier is the first one
  if [[ $les_add == "00000000" ]] && [[ $most_add == "00000000" ]]
  then
    echo "0000000100000000" >> add_inter.txt
  elif [[ $les_add == "00000001" ]] && [[ $most_add == "00000000" ]]
  then
    echo "0000000100000001" >> add_inter.txt
#check if the tier is the last one
  elif [[ $les_add == "00000000" ]] && [[ $most_add == $last_tier_bin ]]
  then
    echo ""$before_last_bin"00000000" >> add_inter.txt
  elif [[ $les_add == "00000001" ]] && [[ $most_add == $last_tier_bin ]]
  then
    echo ""$before_last_bin"00000001" >> add_inter.txt
#check if the tier is intermediate one
  elif (( $most_add > 0 )) && (( $most_add < $last_tier_bin ))
  then
    if [ $les_add == "00000000" ]
    then
      upper_tier=$((most_add_dec+1))
      upper_tier_bin=${D2B[$upper_tier]}
      lower_tier=$((most_add_dec-1))
      lower_tier_bin=${D2B[$lower_tier]}
      echo ""$lower_tier_bin"00000000" >> add_inter.txt
      echo ""$upper_tier_bin"00000000" >> add_inter.txt
    elif [ $les_add == "00000001" ]
    then
      upper_tier=$((most_add_dec+1))
      upper_tier_bin=${D2B[$upper_tier]}
      lower_tier=$((most_add_dec-1))
      lower_tier_bin=${D2B[$lower_tier]}
      echo ""$lower_tier_bin"00000001" >> add_inter.txt
      echo ""$upper_tier_bin"00000001" >> add_inter.txt
    fi
  fi
#Creating the routing tables format to append them to top module file
  sed -i -e "s/^/16'b/" add_inter.txt
  count=$((count+1))
  paste -d, -s add_inter.txt >> inter_routing.txt
done < addresses.txt
no_router=0;
line_add=0;
while read line_add; do
  echo "assign ram_values_$no_router" = {"$line_add"};" >> routing_tables.txt
  no_router=$((no_router+1))
done < inter_routing.txt

```

Figure 6.16: Creating the routing tables in mesh topology

The router address length is two bytes. The Most significant byte defines the number of tier in which the router is located, while the least significant one defines the order of the router in this tier as illustrated in Figure 6.17. This router is located in the first tier and it comes as the second router in this tier. Also, the two first routers in each tier are reserved to be 3D elevator routers. Other than these two routers, all routers are 2D routers.

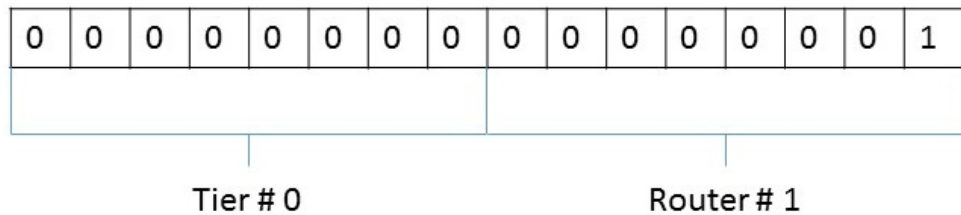


Figure 6.17: The router address

- Creating the implementation according to the ring topology including the router instantiations and other design parameters: The "Ring_Topology" script first calculates the number of total wires needed by the system according to the user's entries which are represented in the number of tiers and the number of routers per each tier as shown in Figure 6.18.

```

L#Counting all the wires required in the system
param_file=Design_files/param.h
Number_of_wires_in_system=0;
total_routers=0;
Number_of_routers=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
Number_of_routers=$i
Number_of_wires_in_system=$((2 + $Number_of_routers + $Number_of_wires_in_system))
total_routers=$((($total_routers+$Number_of_routers))
done
Number_of_wires_in_system=$((($Number_of_wires_in_system - 2))
echo "\`define NUM_OF_WIRES \"$Number_of_wires_in_system\"" >> wires_number.txt
total_routers=$((($total_routers-1))
echo "output [63:0] R_packet [\"$total_routers\":0];" >>routers_total.txt
#Append here number of wires to the param.h file
sed -i '/wires/,/end_here/{//!d}' $param_file
sed -i '/wires/rwires_number.txt' $param_file
sed -i '/r_packet/,/end_r/{//!d}' $design_file
sed -i '/r_packet/routers_total.txt' $design_file
rm -rf routers_total.txt
rm -rf wires_number.txt
#####
for (( n = 0 ; n < $Number_of_wires_in_system ; n++ ))
do
wires[$n]=$n
done
#####

```

Figure 6.18: Evaluating the number of wire in the 3D-NoC system using ring topology

Additionally, the script then creates the connection between the routers to match the selected topology, in this case the ring topology as shown in Figure 6.19. Then Figure 6.20, shows the instantiations of the ring routers to be then added in the SystemVerilog file.

```

#2D_connections
last_router=0;
for (( j = 0 ; j < $Number_of_routers - 1 ; j++ ))
do
next_router=0;
next_router=$((j+1))
if [ $j == "0" ]
then
d=$((m+1))
last_router=$((Number_of_routers-1))
printf "R["$j"] "$tier_count" links["${wires[$m]}"],\n" >> connections.txt
printf "R["$last_router"] "$tier_count" links["${wires[$m]}"],\n" >> connections.txt
printf "R["$j"] "$tier_count" links["${wires[$d]}"],\n" >> connections.txt
printf "R["$next_router"] "$tier_count" links["${wires[$d]}"],\n" >> connections.txt
m=$((m+2))
else
printf "R["$j"] "$tier_count" links["${wires[$m]}"],\n" >> connections.txt
printf "R["$next_router"] "$tier_count" links["${wires[$m]}"],\n" >> connections.txt
m=$((m+1))
fi
done
done
#####
#3D_connections
for (( l = 1 ; l < $tier_count ; l++ ))
do
printf "R[0] "$l" links["${wires[$m]}"],\n" >> connections.txt
printf "R[0] "$((l+1))" links["${wires[$m]}"],\n" >> connections.txt
printf "R[1] "$l" links["${wires[$((m+1))]}"],\n" >> connections.txt
printf "R[1] "$((l+1))" links["${wires[$((m+1))]}"],\n" >> connections.txt
m=$((m + 2))
done
#####
#Preparing the final connections
tier_count_1=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
Number_of_routers=$i
tier_count_1=$((tier_count_1+1))
for (( h = 0 ; h < $Number_of_routers ; h++ ))
do
grep -w "R\["$h"\]" "$tier_count_1" connections.txt | awk '{print}' ORS=' ' >> inter.txt
echo $'\n' >> inter.txt
sed -i 's/, $//' inter.txt
sed -i '/^$/d' inter.txt
sed -i 's/R\[[0-9][0-9]*\] [0-9][0-9]* //g' inter.txt
sed -i 's/ //g' inter.txt
done
done
#####

```

Figure 6.19: Creating the connections between the routers in the ring topology


```

#####
#Adding the addresses to an array
IFS=$'\n' read -d '' -r -a addresses < addresses.txt
#Preparing the final version of the ring_connections for each router per each tier with Verilog required syntax
lines_count=0
tier_count_0=0
a=0;
tiers=${#Number_of_Routers_per_tier[@]}
while read line; do
  lines[$a]=$line
  a=$((a + 1))
done < inter_rev.txt
b=0
for i in "${Number_of_Routers_per_tier[@]}"
do
  tier_count_0=$((tier_count_0+1))
  Number_of_routers=$i
  for (( j = 0 ; j < $Number_of_routers; j++ ))
  do
    add=${addresses[$b]}
    if [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 == "1" ]
    then
      printf "Router_ring #(.address(16'b"$add"),.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
      .clk(clk),.packet(packet),.R_packet(R_packet ["$b"]); \n" >> ring_connections.txt
    fi
  done
done

```

Figure 6.20: The instantiations of the ring routers

Eventually in the script, the instantiations are added in the top level design of the system as shown in Figure 6.21.

```

#####
#Append to the design file
design_file=Design_files/topmod.sv
sed -i '/ROUTER INSTANCES/,/END_Routers/{//!d}' $design_file
sed -i '/ROUTING TABLES/,/END_Tables/{//!d}' $design_file
sed -i '/ROUTER INSTANCES/rring_connections.txt' $design_file
sed -i '/Wires_declartion/,/end_wires/{//!d}' $design_file
sed -i '/Lookup_tables/,/end_luts/{//!d}' $design_file
#sed -i '/ROUTING TABLES/rrouting_tables.txt' $design_file
#####
#Cleaning up
rm -rf connections.txt
rm -rf ring_connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf addresses.txt
#####

```

Figure 6.21: Adding the ring routers instantiations into the SystemVerilog top design file

The "Ring_Routing_Tables_Add.bash" script then will be called to create an address for each router in the system as shown in Figure 6.22.

```
#####
Number_of_Routers_per_tier=("${@}")
D2B=({0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1})
tiers=${#Number_of_Routers_per_tier[@]}
last_tier=$((tiers-1))
last_tier_bin=${D2B[$last_tier]}
before_last=$((last_tier-1))
before_last_bin=${D2B[$before_last]}
tier_count_0=0
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    for (( j = 0 ; j < $Number_of_routers; j++ ))
    do
        D2B=({0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1})
        tier_no=${D2B[$tier_count_0]}
        address_no=${D2B[$j]}
        echo "$tier_no"$address_no" >> addresses.txt
    done
    tier_count_0=$((tier_count_0+1))
done
```

Figure 6.22: Creating address for each router in ring topology

- Creating the top module file "top_mod.sv" and the head parameters in "param.h" files: As illustrated above the connections between routers, the router instantiations and the header file parameters are evaluated through an automation infrastructure. These data are added to the top level design file and header file to create different configurable 3D-NoCs.
 1. Top level design file: The top design file includes the instantiations of the routers both in mesh or ring according to the user's entry as shown in Figure 6.23. Also, it includes only in the case of the mesh topology, the entries to be stored in the routing table as shown in Figure 6.24.

```

`include "param.h"
module topmod (packet, clk, R_packet);
input [63:0] packet;
input      clk;
wire [63:0] links [ `NUM_OF WIRES-1:0];
//r_packet
output [63:0] R_packet [3:0];
//end_r
//Wires_declaration
wire [15:0] ram_values_0 [ `NO_OF ROUTERS_1_TIER-1:0];
wire [15:0] ram_values_1 [ `NO_OF ROUTERS_1_TIER-1:0];
wire [15:0] ram_values_2 [ `NO_OF ROUTERS_2_TIER-1:0];
wire [15:0] ram_values_3 [ `NO_OF ROUTERS_2_TIER-1:0];
//end_wires
//ROUTER INSTANCES
Router_mesh #(.address(16'b0000000000000000),.ramSize(`NO_OF ROUTERS_1_TIER)) router_0 (.port( {links[2],links[0]} )
,.clk(clk),.packet(packet),.ram_values(ram_values_0),.R_packet(R_packet [0]));
Router_mesh #(.address(16'b0000000000000001),.ramSize(`NO_OF ROUTERS_1_TIER)) router_1 (.port( {links[3],links[0]} )
,.clk(clk),.packet(),.ram_values(ram_values_1),.R_packet(R_packet [1]));
Router_mesh #(.address(16'b0000000100000000),.ramSize(`NO_OF ROUTERS_2_TIER)) router_2 (.port( {links[2],links[1]} )
,.clk(clk),.packet(),.ram_values(ram_values_2),.R_packet(R_packet [2]));
Router_mesh #(.address(16'b0000000100000001),.ramSize(`NO_OF ROUTERS_2_TIER)) router_3 (.port( {links[3],links[1]} )
,.clk(clk),.packet(),.ram_values(ram_values_3),.R_packet(R_packet [3]));
//END_Routers

```

Figure 6.23: The router instantiations in the top module design

```

L//Lookup_tables
assign ram_values_0 = {16'b0000000000000001,16'b0000000100000000};
assign ram_values_1 = {16'b0000000000000000,16'b0000000100000001};
assign ram_values_2 = {16'b0000000100000001,16'b0000000000000000};
assign ram_values_3 = {16'b0000000100000000,16'b0000000000000001};
//end_luts
endmodule

```

Figure 6.24: The data in the routing table

2. Parameters header file: The header file includes the common parameters of the design that are adjusted according to the user's entry as shown Figure 6.25.

```

//This file includes the paramters needed to the design.
//Number of routers per tier in the design, they vary according to the user entry.
//routers_tier
`define NO_OF ROUTERS_1_TIER 2
`define NO_OF ROUTERS_2_TIER 2
//end_tiers
//wires
`define NUM_OF WIRES 4
//end_here

```

Figure 6.25: The global parameters of the 3D-NoC system

3. Testbench SystemVerilog file The testbench file is used only in the simulations to send a packet to be routed through the 3D-NoC system Figure 6.26. The packets are transmitted from a reserved address of all ones.

```
//This file includes the packets to be sent through the network in series
//TestBench
`include "param.h"
module testBn ();
reg [63:0] pp ;
reg clk;
topmod temp(.packet(pp), .clk(clk), .R_packet() );
initial
begin
# 10
pp = 64'b11111111_11111111_00000001_00000011_11111111_11111111_11111111_11111111 ;
# 10
pp= 'bz;
end
//Clock
initial
begin
clk = 0;
forever #5 clk = ~clk;
end
endmodule
```

Figure 6.26: The testbench for simulations

- The mesh router module The mesh router module includes the important parameters which are needed for this simple structure as shown in Figure 6.27. The implementation of the 2D routing based on the mesh topology is shown in Figure 6.28, while the logic of the Direct-Elevator 3D routing algorithm is shown in Figure 6.29.

```

//This file includes the router module in the mesh topology
#include "param.h"
module Router_mesh (port, clk, packet, ram_values, R_packet);
//Module inputs,outputs and parameters
//Design inputs
input [63:0] packet; //packet transmitted
input clk;
//Design parameters
parameter ramSize=0; //ramsize
parameter [15:0] address=16'b0; //Router address
//Design inputs/outputs
inout [63:0] port [ramSize-1:0]; // Router ports
input [15:0] ram_values [ramSize-1:0];
output [63:0] R_packet;
//Design registers
wire [15:0] ram [ramSize-1:0]; // memory cells (16Bit Wide) //Routing Table
reg [ramSize-1:0] enable ;
reg [63:0] outlatch[ramSize-1:0];
reg [63:0] inlatch[ramSize:0];
reg [63:0] R_packet;
//Design bits
bit flg; //Flag for 3D routers round robin
assign ram = ram_values;
//Design outputs
genvar m;
generate
for(m=0; m<ramSize; m=m+1)
assign port [m]= enable[m] ? outlatch[m]:64'bz;
endgenerate
generate
for(m=0; m<ramSize; m=m+1)
assign inlatch [m]= port[m];
assign inlatch [ramSize]= packet;
endgenerate

```

Figure 6.27: The parameters of the mesh router module

```

always @ (posedge clk)
begin
    integer i;
    integer j;
    integer k;
    enable = 'b0;

    for(k=0;k<ramSize+1;k=k+1)
        //status <= "--";
    for(k=0;k<ramSize+1;k=k+1)
    begin
        if (address == inlatch[k][47:32])
        begin
            $display("I am the destination");
            R_packet <= inlatch[k];
            //status[k] <= "destination";

            // Save the packet in the inlatch of the router to keep it. (inlatch or final_destination register)
        end
        else if (address == inlatch[k][63:48])
        begin
            $display("I am the source");
            //status[k] <= "source";

            //
        end
        else if (address [15:8] != inlatch[k][47:40]) //Case of the source and destination not in the same tier.
        begin
            //status[k] <= "DTier-2DS";

            if ( ~(address [7:0] == 8'b00000000 || address [7:0] == 8'b00000001) ) // 2D router case
            begin
                for (i = 0; i<ramSize; i++)
                begin
                    if (ram [i][7:0] == 0 && flg == 0 )
                    begin
                        outlatch[i] <= inlatch[k]; //Packet to be output on the inlatch connected to the router
                        outlatch[i][63:48] <= address;
                        enable[i]<=1;
                        flg<=1;
                    end
                    else if (ram [i][7:0] == 1 && flg == 1 )
                    begin
                        outlatch[i] <= inlatch[k]; //Packet to be output on the inlatch connected to the router
                        outlatch[i][63:48] <= address;
                        enable[i]<=1;
                        flg<=0;
                    end
                end
            end
        end
    end
end

```

Figure 6.28: The implementation of the 2D routing with mesh topology

```

-----
else if (address [7:0] == 8'b00000000 || address [7:0] == 8'b00000001) //3D router case
//status[k] <= "DTier-3DS";

begin
  if (address [15:8] < inlatch[k][47:40])
  begin
    for (i = 0; i < ramSize; i++)
    begin
      if (ram [i][15:8] > address [15:8])
      begin
        //status[k] <= "DTier-3DS-down";
        outlatch[i] <= inlatch[k]; //Packet to be output on the inlatch connected to the router
        outlatch[i][63:48] <= address;
        enable[i] <=1;
      end
    end
  end
  else if(address [15:8] > inlatch[k][47:40])
  begin
    for (i = 0; i<ramSize; i++)
    begin
      //status[k] <= "DTier-3DS-UP";
      if (ram [i][15:8] < address [15:8])
      begin
        outlatch[i] <= inlatch[k]; //Packet to be output on the inlatch connected to the router
        outlatch[i][63:48] <= address;
        enable[i]<=1;
      end
    end
  end
end
end
end
////////////////////////////////////////////////////
else if (address [15:8] == inlatch[k][47:40]) //Case of the soure and destination in the same tier
begin
  $display("Destination and soure are in the same tier");
  //status [k] <= "STier";
  for (j = 0; j<ramSize; j++)
  begin
    if (ram[j] == inlatch[k][47:32])
    begin
      outlatch [j] <= inlatch[k]; //Packet to be output on the port connected to the router
      outlatch [j][63:48] <= address;
      enable [j] <=1;
    end
  end
end
end
end
endmodule

```

Figure 6.29: The implementation of the Direct-Elevator algorithm in the 3D mesh router module

- The ring router module The ring router module includes the important parameters which are needed for this simple structure as shown in Figure 6.30. The implementation of the 2D routing based on the ring topology is shown in Figure 6.31, while the logic of the Direct-Elevator 3D routing algorithm is shown in Figure 6.32.

```

//This file includes the router module in the ring topology
`include "param.h"
module Router_ring (port, clk, packet, R_packet);
//Module inputs,outputs and parameters
//Design inputs
input [63:0] packet; //packet transmitted
input clk;
//Design parameters
parameter ramSize=0; //ramsize
parameter [15:0] address=16'b0;//Router address
//Design inputs/outputs
inout [63:0] port [ramSize-1:0]; // Router ports
output [63:0] R_packet;
//Design registers
reg [15:0] ram [ramSize-1:0];// memory cells (16Bit Wide) //Routing Table
//reg [500:0] status [ramSize:0];
reg [ramSize-1:0] enable ;
reg [63:0] outlatch[ramSize-1:0];
reg [63:0] inlatch[ramSize:0];
reg [63:0] R_packet;
//Design outputs
genvar m;
generate
for(m=0; m<ramSize; m=m+1)
assign port [m]= enable[m] ? outlatch[m]:64'bz;
endgenerate

generate
for(m=0; m<ramSize; m=m+1)
assign inlatch [m]= port[m];

assign inlatch [ramSize]= packet;
endgenerate

```

Figure 6.30: The parameters of the mesh router module


```

always @ (posedge clk)
begin
    integer i;
    integer j;
    integer k;
    enable = 'b0;

    for(k=0;k<ramSize+1;k=k+1)
        //status <= "--";
    for(k=0;k<ramSize+1;k=k+1)
    begin
        if (address == inlatch[k][47:32])
        begin
            $display("I am the destination");
            R_packet <= inlatch[k];
            //status[k] <= "destination";

            // Save the packet in the inlatch of the router to keep it. (inlatch or final_destination register)
        end
        else if (address == inlatch[k][63:48])
        begin
            $display("I am the source");
            //status[k] <= "source";

            //
        end
        else if (address [15:8] != inlatch[k][47:40]) //Case of the source and destination not in the same tier.
        begin
            //status[k] <= "DTier-2DS";

            if ( ~(address [7:0] == 8'b00000000 || address [7:0] == 8'b00000001)) // 2D router case
            begin
                $display("Destination and source are in the same tier");
                //status [k] <= "STier";
                outlatch [1] <= inlatch[k]; //Packet to be output on the port connected to the router
                outlatch [1][63:48] <= address;
                enable [1] <=1;
            end
        end
    end
end

```

Figure 6.31: The implementation of the 2D routing with ring topology

```

        else if (address [7:0] == 8'b00000000 || address [7:0] == 8'b00000001) //3D router case
        //status[k] <= "DTier-3DS";

        begin
            if (address [15:8] > inlatch[k][47:40])
            begin
                outlatch [2] <= inlatch[k]; //Packet to be output on the port connected to the router
                outlatch [2][63:48] <= address;
                enable [2] <=1;
            end
            else if(address [15:8] < inlatch[k][47:40])
            begin
                outlatch [ramSize-1] <= inlatch[k]; //Packet to be output on the port connected to the router
                outlatch [ramSize-1][63:48] <= address;
                enable [ramSize-1] <=1;
            end
        end
    end
//end
////////////////////////////////////////////////////
else if (address [15:8] == inlatch[k][47:40]) //Case of the source and destination in the same tier
begin
    $display("Destination and source are in the same tier");
    //status [k] <= "STier";
    outlatch [1] <= inlatch[k]; //Packet to be output on the port connected to the router
    outlatch [1][63:48] <= address;
    enable [1] <=1;
end
end
end
endmodule

```

Figure 6.32: The implementation of the Direct-Elevator algorithm in the ring router module

- Issuing a message to notify the user that the design files are now ready: A TK script as shown in Figure 6.33 is then implemented to issue an information message to the user that the design files are now ready.

```
#!/usr/local/bin/wish
package require Tk
#package require BWidget
#frame .message
    set ans [tk_messageBox -icon info -title "3D NoCs Designs" -message "Please check the design files created under Design_files directory." -type ok ]
    switch -- $ans {
        ok { destroy . }
    }

```

Figure 6.33: The final information message

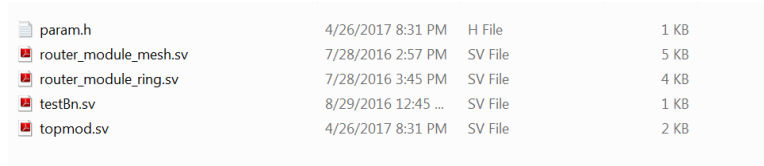
6.2.4 How to use the 3D-NOCET tool suite

To use the 3D-NOECT tool suite, the user needs to make sure about the following prerequisites:

- If the user uses a Windows machine, the user needs to install "Cygwin" to able to run the TK/TCL and Shell scripts.
- The user can use a Linux machine directly without installing any other tool.
- Providing a working area to unzip the tool bundle in and create the design files.

A user needs to follow the next steps in order to create the required 3D-NoC:

1. Copy the tool bundle into the working area.
2. Unzip the tool bundle into the working area.
3. Launch the tool GUI by the command "wish Startup.tcl"
4. Enter the specifications of the 3D-NoC by entering the number of routers per each tier, number of tiers and the 2D network topology.
5. Press "create design files" button within the GUI window.
6. Check the "Design_files" directory under the working area, the following files can be found as shown in Figure 6.34



param.h	4/26/2017 8:31 PM	H File	1 KB
router_module_mesh.sv	7/28/2016 2:57 PM	SV File	5 KB
router_module_ring.sv	7/28/2016 3:45 PM	SV File	4 KB
testBn.sv	8/29/2016 12:45 ...	SV File	1 KB
topmod.sv	4/26/2017 8:31 PM	SV File	2 KB

Figure 6.34: Design_files directory

7. The design files of the 3D-NoC are now ready for synthesis.

Chapter 7: Discussion on A Comparative and Performance Study for Different Structures of 3D-NoCs

7.1 Introduction

A Comparative and Performance study has been performed in order to evaluate and compare between the performance of the different 3D-NoCs structures. The study compare the diverse 3D-NoCs with respect to latency, power and area factors. The Synthesis processes have been all performed in the same environmental settings to guarantee a fair evaluation for the 3D-NoCs. The unified environmental settings are as follows:

1. Questa Simulation tool.
2. Xilinx-Virtex7 technology.
3. All designs are synthesized at frequency of 100MHz.

7.2 Comparative Results Analysis

All the calculations are taken at the worst conditions. Therefore, the power is calculated as the worst possible power can be consumed by the 3D-NoC, also the latency is calculated over the worst network path between two nodes in the system.

The impact of the vertical complexity, tier complexitiy and changing the 2D network topology is measured on the latency, power and area respectively.

7.2.1 Vertical Network Complexity

The vertical complexity measures the impact of increasing the number of tiers on the 3D-NoC system. This impact is measured with respect to the latency, power and area aspects while varying the 2D network topology between mesh and ring.

7.2.1.1 Impact of Vertical Complexity on The Latency

The Latency is calculated over the worst path while increasing the number of tiers from two to 16. Figure 7.1 shows the latency in clock cycles.

The latency in the mesh network topology increases linearly while increasing the number of the 3D-NoC tiers. Additionally, No change happens if the number of the routers per tier doubled from four to eight routers, because the routers in each tier are connected in a fully-mesh network. Meanwhile, the latency in the ring network topology increases linearly with the number of tiers, yet the latency increases in the case of doubling the number of routers. While doubling the number of routers which are connected in ring network topology that lengthen the network worst path which reflects certainly reflects on the network latency.

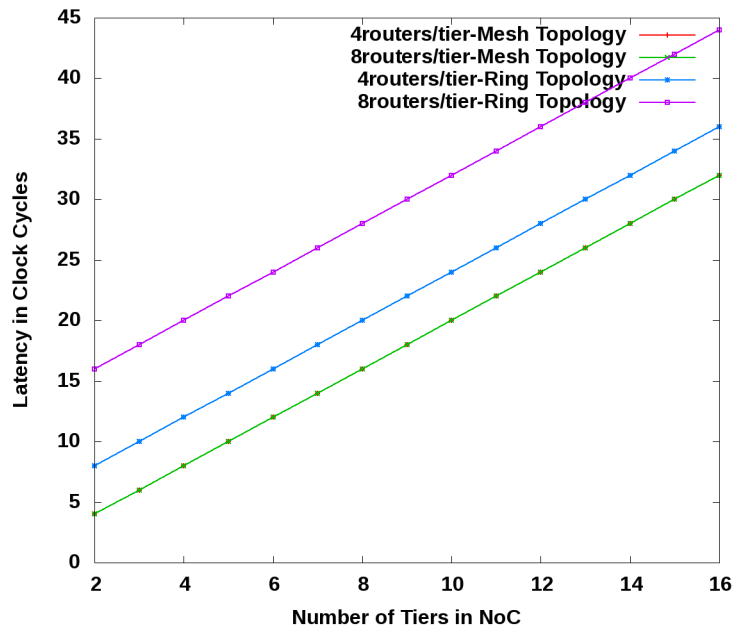


Figure 7.1: Latency in Clock Cycles Vs. Number of Tiers in NoC

7.2.1.2 Impact of Vertical Complexity on The Power

the power is calculated in Watts to measure the worst consumption. Figure 7.2 shows that while increasing the number of tiers in the 3D-NoC system, The number of tiers increases slightly.

Moreover, while doubling the number of routers per each tier, the system becomes more complex, this complexity reflects on increasing the system power consumption.

On the other hand, the power in the ring topology remains almost constant and relatively small while increasing the number of tiers in the vertical dimension. Also, after doubling the number of routers per each tier, the extra number of interconnects does not leave a significant effect on the power consumption.

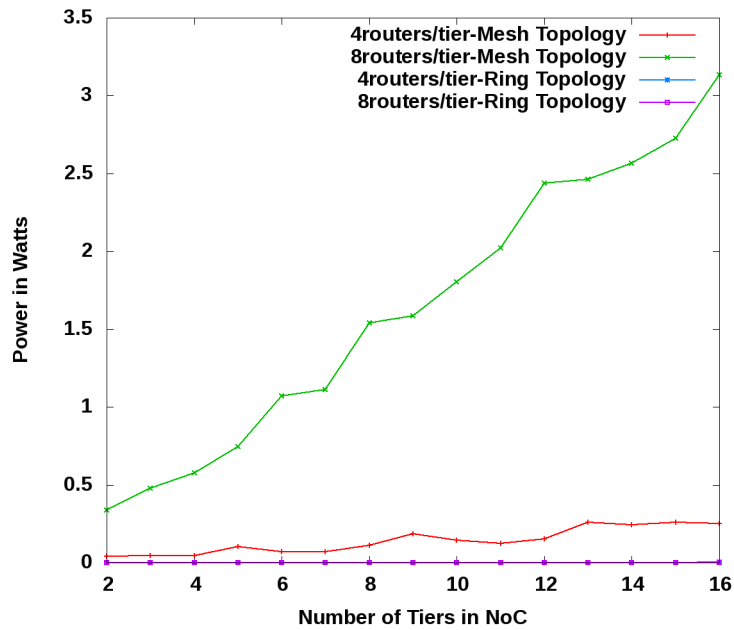


Figure 7.2: Power in Watts Vs. Number of Tiers in NoC

7.2.1.3 Impact of Vertical Complexity on The Area

The area is represented in the number of Lookup Tables (LUTs) and the number of FlipFlops (FFs). In Figure 7.3 and Figure 7.4, the number of LUTs and FFs increase rapidly while doubling the number of routers in the mesh topology. The number of interconnections increase exponentially while adding an extra router in each tier, that definitely reflects on the high increase of the number of LUTs and FFs. Otherwise, the ring topology provides a moderate number of interconnects between the router in the planar level and that certainly affects the consumed area positively.

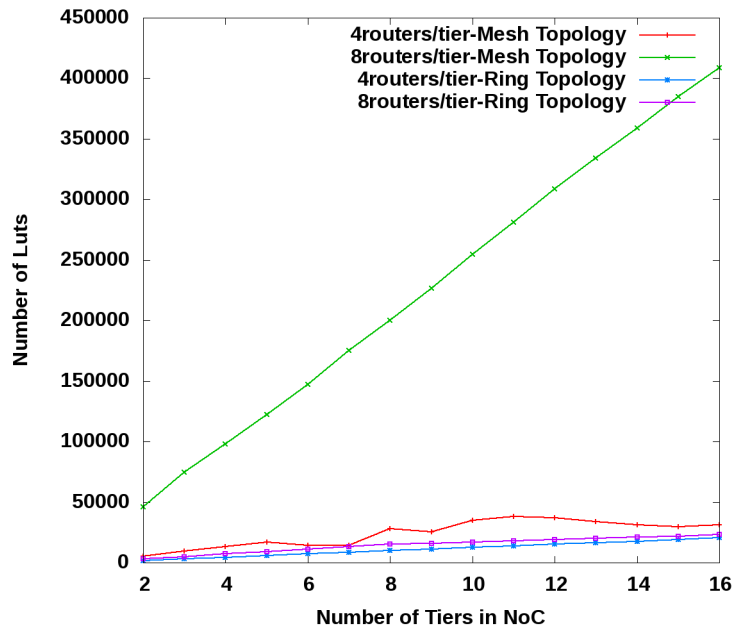


Figure 7.3: Number of Luts Vs. Number of Tiers in NoC

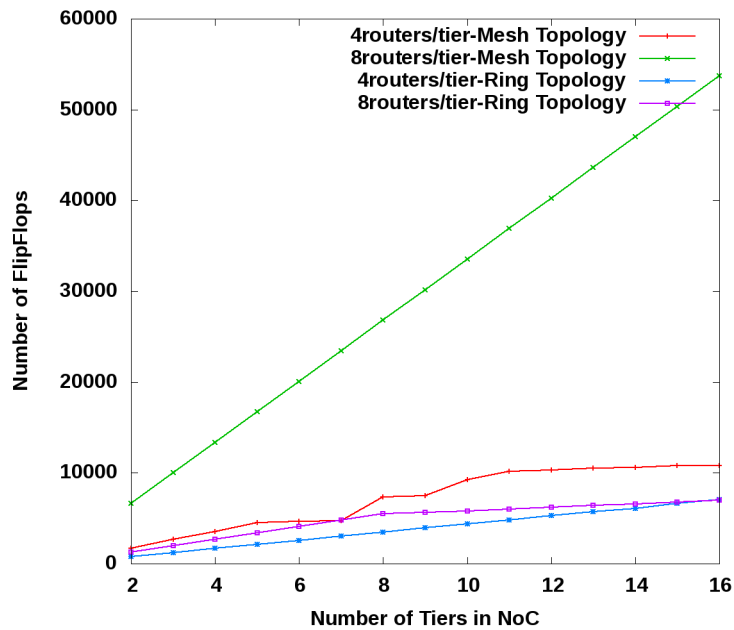


Figure 7.4: Number of FlipFlops Vs. Number of Tiers in NoC

7.2.2 Tier Network Complexity

The tier complexity measure the effect of increasing the number of routers per each tier whether the routers are connected in the 2D level with mesh network topology or ring. The effect is studied on latency, power and area aspects.

7.2.2.1 Impact of Tier Complexity on The Latency

The latency is fixed across the variation of the number of routers per each tier as shown in Figure 7.5. Meanwhile, it increases while doubling the number of tiers in the 3D-NoC, yet it remains constant in the function of the number of routers per each tier. The Latency is constant in the mesh topology, as it depends only on the number of vertical interconnects between tiers which remains constant.

On the other hand, when the routers are connected in ring topology, the latency increases linearly with the increase in the number of routers per each tier. As by adding an extra router in the tier, that will increase the number of interconnections between the routers.

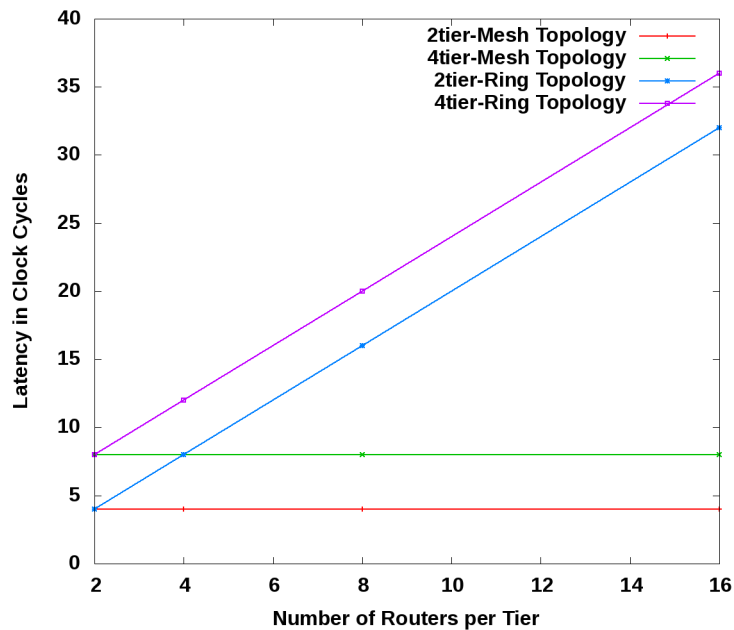


Figure 7.5: Latency in Clock Cycles Vs. Number of Routers in Tier

7.2.2.2 Impact of Tier Complexity on The Power

In Figure 7.6, the power consumption is shown across increasing the number of routers per each tier. The power consumption increases while increasing the number of routers per each tier in case of the routers are connected in mesh topology. Meanwhile, the power consumption remains at a low level while connecting the routers in a ring topology.

Therefore, the ring topology in this case proves to be less complex and more efficient when it comes to increasing the tier complexity.

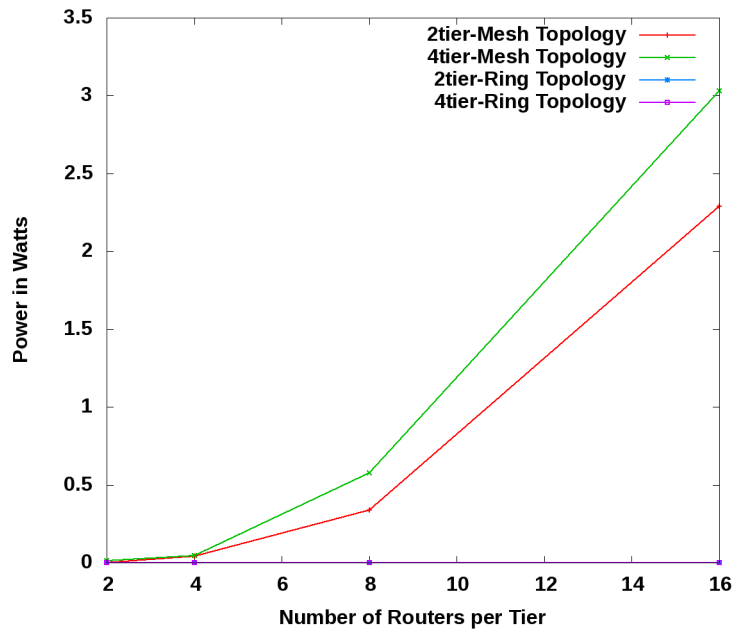


Figure 7.6: Power in Watts Vs. Number of Routers in Tier

7.2.2.3 Impact of Tier Complexity on The Area

The implementation of the 3D-NoC in which the routers in each tier are connected in 2D mesh network topology, requires a large hardware to support all the interconnections in the system. That appears clearly in Figure 7.7 and Figure 7.8 while increasing the number of routers per tier, the consumed area increases. Moreover, the consumed area in the case of using the ring topology becomes less because of the moderate interconnections between the routers in each tier.

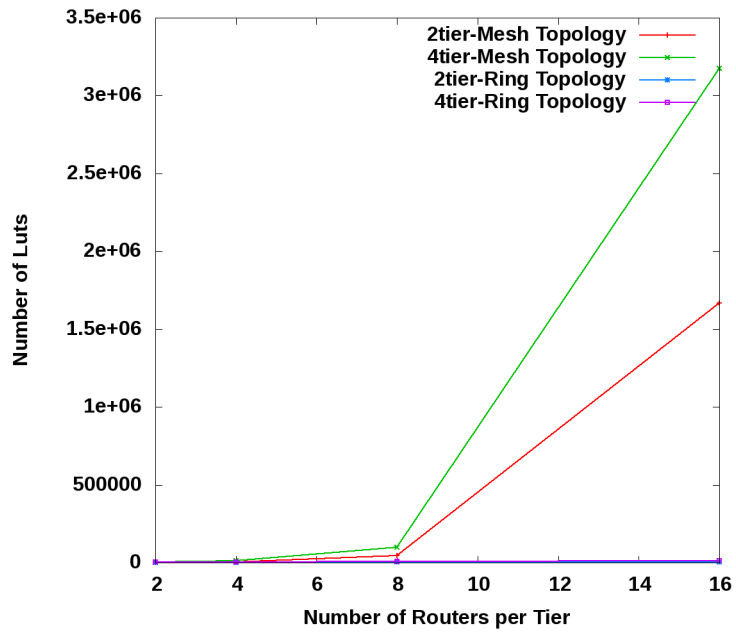


Figure 7.7: Number of Luts Vs. Number of Routers in Tier

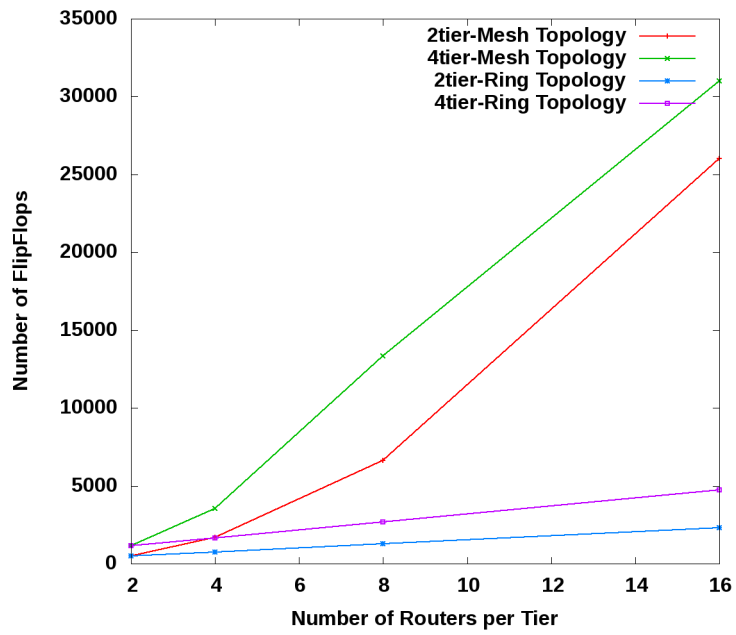


Figure 7.8: Number of FlipFlops Vs. Number of Routers in Tier

7.3 Work Conclusion

Implementing a tool to create different structures of 3D-NoCs gives clear insights about the different network aspects and tradeoffs.

The user can easily for one application try many diverse configurations of a 3D-NoCs and study the tradoffs regarding the latency, power and area. The tool also enables different distributions for the 3D-NoC whether it is regular or hierarchal that widens the range of the experimental evaluations.

Using a fully-mesh topology as a 2D network topology guarantees a faster 3D-NoC, but with a high area cost. Meanwhile, the ring topology offers a relatively low area and power consumption in the 3D-NoC system, yet the system becomes slower in routing the packets.

7.4 Future Work

Nevertheless, The 3D-NOCET tool provides a flexible and generic solution to create different 3D-NoCs, there are many ideas that can be applied to improve and stenghten the tool functionality which are as follows:

- Adding the support of other 2D network topology such as partially mesh, star, and torus topologies, that will enable the user to create 3D-NoCs with more configurations.
- Another router implementations can be added instead of the simple proposed router to test the dependency of the tool on the different router.
- Optimizing the necessary hardware to implement the fully-mesh and ring 2D topologies.
- Creating a web-based Graphical User Interface for the tool to be more friendly.

References

- [1] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, “A detailed and flexible cycle-accurate Network-on-Chip simulator,” in 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 86–96, April 2013.
- [2] A. B. Achballah and S. B. Saoud, “A Survey of Network-On-Chip Tools,” in (IJACSA) International Journal of Advanced Computer Science and Applications, pp. 132–135, Sept 2013.
- [3] Y.-C. Lu, “3D technology based circuit and architecture design,” in 2009 International Conference on Communications, Circuits and Systems, pp. 1124–1128, July 2009.
- [4] W. Davis, Wilson, J., S. Mick, J. Xu, H. Hua, C. Mineo, A. Sule, M. Steer, and P. Franzon, “Demystifying 3D ICs: the Pros and Cons of Going Vertical,” Design Test of Computers, IEEE, vol. 22, pp. 498–510, Nov 2005.
- [5] R. Francis and S. Moore, “Exploring hard and soft networks-on-chip for FPGAs,” in 2008 International Conference on Field-Programmable Technology, pp. 261–264, Dec 2008.
- [6] <http://www.cringely.com>. Accessed: 2019-06-17.
- [7] A. Ahmed and A. Abdallah, “LA-XYZ: Low Latency, High Throughput Look-Ahead Routing Algorithm for 3D Network-on-Chip (3D-NoC) architecture,” in 2012 IEEE 6th International Symposium on Embedded Multicore Socs (MCSoc), pp. 167–174, Sept 2012.
- [8] K.-C. Chen, S.-Y. Lin, H.-S. Hung, and A. Wu, “Topology-Aware Adaptive Routing for Nonstationary Irregular Mesh in Throttled 3D NoC Systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 24, pp. 2109–2120, Oct 2013.
- [9] B. M., S. A., P. F., D. F., and D. P., “A 3D-NoC Router Implementation Exploiting Vertically-Partially-Connected Topologies,” in 2012 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 9–14, Aug 2012.
- [10] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, “Noxim: An open, extensible and cycle-accurate network on chip simulator,” in 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 162–163, July 2015.
- [11] M.S.Gaur, B.M.Al-Hashimi, V.Laxmi, N. R, N. Choudhary, L. Jain, M. Ahmed, K. K. Paliwal, Varsha, Rekha, and Vineetha, “NIRGAM:A Simulator for NoC Interconnect Routing and Applications Modeling,” p. 27, 2007.
- [12] Papamichael, M. K., and J. C. Hoe, “CONNECT: Re-examining Conventional Wisdom for Designing NoCs in the Context of FPGAs,” in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA ’12, (New York, NY, USA), pp. 37–46, ACM, 2012.

- [13] B. Black, Nelson, D.W., Webb, C., and N. Samra, “3D Processing Technology and its Impact on iA32 Microprocessors,” in *VLSI in Computers and Processors*, 2004. ICCD 2004. Proceedings. IEEE International Conference on Computer Design, pp. 316–318, Oct 2004.
- [14] N. P. Pham, N. Tutunjan, D. Volkaerts, L. Peng, G. Jamison, and D. S. Tezcan, “3D integration technology using W2w direct bonding and TSV for CMOS based image sensors,” in *2015 IEEE 17th Electronics Packaging and Technology Conference (EPTC)*, pp. 1–5, Dec 2015.
- [15] Knickerbocker, J.U., A. P.S., C. E., D. B., D. T., G. X., H. C., J. C., L. Y., M. J., P. R.J., T. C.K., T. L., W. B.C., W. L., and W. S.L., “2.5D and 3D Technology Challenges and Test Vehicle Demonstrations,” in *Electronic Components and Technology Conference (ECTC)*, 2012 IEEE 62nd, pp. 1068–1076, May 2012.
- [16] <http://www.amd.e-technik.uni-rostock.de/noc>. Accessed: 2019-06-17.
- [17] T. A. Bartic, J. Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, “Highly scalable network on chip for reconfigurable systems,” in *Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748)*, pp. 79–82, Nov 2003.
- [18] J. Wang, Z. Zhang, and J. Lai, “A NoC Architecture for high-speed Dynamic Partial Reconfiguration,” in *2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology*, pp. 1–3, Oct 2012.
- [19] B. S. Feero and P. P. Pande, “Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation,” *IEEE Transactions on Computers*, vol. 58, pp. 32–45, Jan 2009.
- [20] <http://www.cs.colostate.edu>. Accessed: 2019-06-17.
- [21] K. Goossens, M. Bennebroek, J. Y. Hur, and M. A. Wahlah, “Hardwired Networks on Chip in FPGAs to Unify Functional and Configuration Interconnects,” in *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pp. 45–54, April 2008.
- [22] E. Todorovich, M. Leonetti, and R. Brinks, “An advanced NoC with debug services on FPGA,” in *2014 IX Southern Conference on Programmable Logic (SPL)*, pp. 1–6, Nov 2014.
- [23] <https://cacm.acm.org/magazines>. Accessed: 2019-06-17.
- [24] <http://electroiq.com>. Accessed: 2019-06-17.
- [25] M. Beheiry, A. Aly, H. Mostafa, and A. M. Soliman, “Direct-Elevator: A modified routing algorithm for 3D-NoCs,” in *2015 27th International Conference on Microelectronics (ICM)*, pp. 222–225, Dec 2015.
- [26] H. Kim, A. Vitkovskiy, P. V. Gratz, and V. Soteriou, “Use it or lose it: Wear-out and lifetime in future chip multiprocessors,” in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 136–147, Dec 2013.

- [27] B. L. and D. M. G., “Networks on Chips: A New SoC Paradigm,” *Computer*, vol. 35, pp. 70–78, Jan 2002.
- [28] Y. L. Lee, J. W. Yang, and J. M. Jou, “Design of a distributed JPEG encoder on a scalable NoC platform,” in 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 132–135, April 2008.
- [29] L. O., S. T., R. S.-A., and T. I., “Layered Routing in Irregular Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 51–65, Jan 2006.
- [30] <http://www.artemis.com>. Accessed: 2019-06-17.
- [31] <http://www.inocs.com>. Accessed: 2019-06-17.
- [32] R. Gutmann, A. Zeng, S. Devarajan, J.-Q. Lu, , and K. Rose, “Wafer-Level Three-Dimensional Monolithic Integration for Intelligent Wireless Terminals,” in *JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE*, VOL.4, NO.3, Sept 2004.
- [33] S.-M. Jung, Y. Rah, T. Ha, H. Park, C. Chang, S. Lee, J. Yun, W. Cho, H. Lim, J. Park, J. Jeong, B. Son, J. Jang, B. Choi, H. Cho, and K. Kim, “Highly cost effective and high performance 65nm S3 (stacked single-crystal si) SRAM technology with 25F2, 0.16um² cell and doubly stacked SSTFT cell transistors for ultra high density and high speed applications,” in *Digest of Technical Papers. 2005 Symposium on VLSI Technology, 2005.*, pp. 220–221, June 2005.
- [34] L. P., de Crecy F., F. M., C. B., E. T., Z. M., J. B., J.-C. Barbe, K. N., S. N., M. S., and L. D., “Challenges for 3D IC integration: Bonding Quality and Thermal Management,” in *International Interconnect Technology Conference, IEEE 2007*, pp. 210–212, June 2007.
- [35] B. S. Feero and P. P. Pande, “Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation,” *IEEE Transactions on Computers*, vol. 58, pp. 32–45, Jan 2009.
- [36] S. Abba and J. A. Lee, “Examining the Performance Impact of NoC Parameters for Scalable and Adaptive FPGA-Based Network-on-Chips,” in 2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation, pp. 364–372, Sept 2013.
- [37] D. F., S. A., P. F., and B. M., “Elevator-First: A Deadlock-Free Distributed Routing Algorithm for Vertically Partially Connected 3D-NoCs,” *IEEE Transactions on Computers*, vol. 62, pp. 609–615, March 2013.

Appendix A: 3D-NOCET Tool Source Files

This appendix includes the 3D-NOCET tool SystemVerilog files and automation scripts

*Automation Scripts

*Startup.tcl

```
#Description: a TK/TCL script that has been created to
#implement an easy friendly Graphical User Interface (GUI)
#for the tool.
```

```
#Author: Maha Beheiry
```

```
#!/usr/local/bin/wish
```

```
lappend auto_path [file dirname [info script]]
```

```
package require Tk
```

```
package require BWidget
```

```
#Main Window size, title and formatting
```

```
wm title . "3D NoCs Design"
```

```
set x [expr { ( [winfo vrootwidth .] - 800 ) / 2 }]
```

```
set y [expr { ( [winfo vrootheight .] - 800 ) / 2 }]
```

```
wm geometry . 800x800+$x+$y
```

```
wm protocol . WM_DELETE_WINDOW {
    destroy .
}
```

```
set family Courier
```

```
set font "$family 14 bold"
```

```
set font_small "$family 12 bold"
```

```
frame .home -borderwidth 10 -width 500 -height 500
```

```
pack .home -side top -fill both
```

```
label .home.topo -font $font -wraplength 4i -justify left
-text "Select Topology:"
```

```
checkbutton .home.mesh -text "Mesh" -variable mesh
-command {set ring 0}
```

```
checkbutton .home.ring -text "Ring" -variable ring
-command {set mesh 0}
```

```
place .home.topo -x 0 -y 0
```

```
place .home.mesh -x 250 -y 0
```

```
place .home.ring -x 320 -y 0
```

```
label .home.text -font $font -wraplength 4i -justify left
-text "Select Number of tiers:"
```

```
place .home.text -x 0 -y 50
```

```
frame .home.pad
```

```
grid .home.pad -column 0 -row 0 -pady 50
```

```

frame .home.pad1
grid .home.pad1 -column 0 -row 1

frame .home.tmp1
checkboxbutton .home.tmp1.tier1 -text "Tier#1" -variable tier1
-command {packmyrouters .home.tmp1 $tier1 1}
pack .home.tmp1.tier1
grid .home.tmp1 -column 0 -row 2 -pady 20
frame .home.tmp2
checkboxbutton .home.tmp2.tier2 -text "Tier#2" -variable tier2
-command {packmyrouters .home.tmp2 $tier2 2}
pack .home.tmp2.tier2
grid .home.tmp2 -column 1 -row 2 -sticky nw -pady 20
frame .home.tmp3
checkboxbutton .home.tmp3.tier3 -text "Tier#3" -variable tier3
-command {packmyrouters .home.tmp3 $tier3 3}
pack .home.tmp3.tier3
grid .home.tmp3 -column 2 -row 2 -sticky nw -pady 20
frame .home.tmp4
checkboxbutton .home.tmp4.tier4 -text "Tier#4" -variable tier4
-command {packmyrouters .home.tmp4 $tier4 4}
pack .home.tmp4.tier4
grid .home.tmp4 -column 3 -row 2 -sticky nw -pady 20
frame .home.tmp5
checkboxbutton .home.tmp5.tier5 -text "Tier#5" -variable tier5
-command {packmyrouters .home.tmp5 $tier5 5}
pack .home.tmp5.tier5
grid .home.tmp5 -column 0 -row 3 -sticky nw -pady 20
frame .home.tmp6
checkboxbutton .home.tmp6.tier6 -text "Tier#6" -variable tier6
-command {packmyrouters .home.tmp6 $tier6 6}
pack .home.tmp6.tier6
grid .home.tmp6 -column 1 -row 3 -sticky nw -pady 20
frame .home.tmp7
checkboxbutton .home.tmp7.tier7 -text "Tier#7" -variable tier7
-command {packmyrouters .home.tmp7 $tier7 7}
pack .home.tmp7.tier7
grid .home.tmp7 -column 2 -row 3 -sticky nw -pady 20
frame .home.tmp8
checkboxbutton .home.tmp8.tier8 -text "Tier#8" -variable tier8
-command {packmyrouters .home.tmp8 $tier8 8}
pack .home.tmp8.tier8
grid .home.tmp8 -column 3 -row 3 -sticky nw -pady 20
frame .home.tmp9
checkboxbutton .home.tmp9.tier9 -text "Tier#9" -variable tier9
-command {packmyrouters .home.tmp9 $tier9 9}

```



```

pack .home.tmp9.tier9
grid .home.tmp9 -column 0 -row 4 -sticky nw -pady 20
frame .home.tmp10
checkboxbutton .home.tmp10.tier10 -text "Tier#10" -variable tier10
-command {packmyrouters .home.tmp10 $tier10 10}
pack .home.tmp10.tier10
grid .home.tmp10 -column 1 -row 4 -sticky nw -pady 20
frame .home.tmp11
checkboxbutton .home.tmp11.tier11 -text "Tier#11" -variable tier11
-command {packmyrouters .home.tmp11 $tier11 11}
pack .home.tmp11.tier11
grid .home.tmp11 -column 2 -row 4 -sticky nw -pady 20
frame .home.tmp12
checkboxbutton .home.tmp12.tier12 -text "Tier#12" -variable tier12
-command {packmyrouters .home.tmp12 $tier12 12}
pack .home.tmp12.tier12
grid .home.tmp12 -column 3 -row 4 -sticky nw -pady 20
frame .home.tmp13
checkboxbutton .home.tmp13.tier13 -text "Tier#13" -variable tier13
-command {packmyrouters .home.tmp13 $tier13 13}
pack .home.tmp13.tier13
grid .home.tmp13 -column 0 -row 5 -sticky nw -pady 20
frame .home.tmp14
checkboxbutton .home.tmp14.tier14 -text "Tier#14" -variable tier14
-command {packmyrouters .home.tmp14 $tier14 14}
pack .home.tmp14.tier14
grid .home.tmp14 -column 1 -row 5 -sticky nw -pady 20
frame .home.tmp15
checkboxbutton .home.tmp15.tier15 -text "Tier#15" -variable tier15
-command {packmyrouters .home.tmp15 $tier15 15}
pack .home.tmp15.tier15
grid .home.tmp15 -column 2 -row 5 -sticky nw -pady 20
frame .home.tmp16
checkboxbutton .home.tmp16.tier16 -text "Tier#16" -variable tier16
-command {packmyrouters .home.tmp16 $tier16 16}
pack .home.tmp16.tier16
grid .home.tmp16 -column 3 -row 5 -sticky nw -pady 20

button .b1 -text "Create Design Files" -width 20
-command { createFiles }
place .b1 -x 300 -y 600 -anchor nw

proc packmyrouters {frame var arg} {
global routers_tier_1
global routers_tier_2
global routers_tier_3

```

```

global routers_tier_4
global routers_tier_5
global routers_tier_6
global routers_tier_7
global routers_tier_8
global routers_tier_9
global routers_tier_10
global routers_tier_11
global routers_tier_12
global routers_tier_13
global routers_tier_14
global routers_tier_15
global routers_tier_16
    set family Courier
    set font_small "$family 9 bold"
if { $var == 1 } {
if { [winfo exists $frame.text] != 1 } {
label $frame.text -font $font_small -wraplength 4i
-justify left -text "#Routers:"
entry $frame.router -textvar routers_tier_$arg
-width 5 -bg white
    }
        pack $frame.text
        pack $frame.router
    } else {
pack forget $frame.text
pack forget $frame.router

}
}
proc createFiles {} {
global routers_tier_1
global routers_tier_2
global routers_tier_3
global routers_tier_4
global routers_tier_5
global routers_tier_6
global routers_tier_7
global routers_tier_8
global routers_tier_9
global routers_tier_10
global routers_tier_11
global routers_tier_12
global routers_tier_13
global routers_tier_14
global routers_tier_15

```

```

global routers_tier_16
global topology
set topology 0
global ring
global mesh
set myList {}
array set arr {}

for {set i 1} {$i < 17} {incr i} {
if { [info exists routers_tier_$i ] == 1} {

lappend myList [set routers_tier_$i]
}
}
if { $mesh == 1} {
set topology Mesh
}
if { $ring == 1} {
set topology Ring
}
lappend myList $topology
#puts $topology
#exec "bash" Master_Script.sh "$myList"
catch {exec bash Master_Script.sh "$myList"} res
puts $res

}

*Master_Script.sh
#!/bin/sh
#Description:a shell script that has been created to call
#either the Mesh or Ring flow scripts according to
#the selected topology by the user.
#Author: Maha Beheiry
#####
####Clean up
rm -rf no_routers_tiers.txt
#user entries
#Number_of_Routers_per_tier=("$@")
Number_of_Routers_per_tier=();
in=0;
topology="";
for i in $1
do
if [ $i == "Mesh" ] || [ $i == "Ring" ]
then

```

```

        topology=$i;
        break;
    fi
    Number_of_Routers_per_tier[in]=$i
    in=$((in+1));
done
#Creating the parameters of routers per tier in the
#param.h file
param_file=Design_files/param.h
count_tier=1;
x=0;
for x in "${Number_of_Routers_per_tier[@]}"
do
    echo "\`define NO_OF_ROUTERS_"$count_tier"_TIER "$x""
>> no_routers_tiers.txt
count_tier=$((count_tier+1))
done
sed -i '/routers_tier/,/end_tiers/{//!d}' $param_file
sed -i '/routers_tier/rno_routers_tiers.txt' $param_file
rm -rf no_routers_tiers.txt
#####
if [ $topology == "Mesh" ]
then
    #SOURCE scripts of Mesh topologies to create the topmod.sv
    #and param.h files
    . Mesh_RoutingTables_Add.bash
    "${Number_of_Routers_per_tier[@]}"
    . Mesh_Topology.sh "${Number_of_Routers_per_tier[@]}"
elif [ $topology == "Ring" ]
then
    #SOURCE Sscripts of Ring topologies to create the topmod.sv
    #and param.h files
    . Ring_RoutingTables_Add.bash
    "${Number_of_Routers_per_tier[@]}"
    . Ring_Topology.sh "${Number_of_Routers_per_tier[@]}"
else
    echo "Please enter a valid topology for the system
    Mesh or Ring."
fi
#cleaning up the working directory
rm -rf routing_tables.txt
rm -rf addresses.txt
*Mesh_RoutingTables_Add.bash
#!/bin/sh
#Description:a shell script which is responsible of creating
#the addresses for all the routers in the network and also the

```

```

# mesh connections between them in the 2D and 3D levels
#and routing tables.
#Author: Maha Beheiry
#####
#Cleaning up the current working directory
rm -rf addresses.txt
rm -rf add_inter.txt
rm -rf inter_routing.txt
rm -rf routing_tables.txt
#####
#Converting from Dec to Bin.
Number_of_Routers_per_tier=("$@")
tiers=${#Number_of_Routers_per_tier[@]}
last_tier=$((tiers - 1))
D2B={0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}
last_tier_bin=${D2B[$last_tier]}
before_last=$((last_tier - 1))
before_last_bin=${D2B[$before_last]}
tier_count_0=0
#####
#Creating the addresses for all routers in the network
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    for (( j = 0 ; j < $Number_of_routers; j++ ))
    do
        D2B={0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}
        tier_no=${D2B[$tier_count_0]}
        address_no=${D2B[$j]}
        echo "$tier_no"$address_no" >> addresses.txt
    done
    tier_count_0=$((tier_count_0+1))
done
#####
#Creating the routing table for each router
count=0;
while read add; do
    rm -rf add_inter.txt
    most_add=${add:0:8}
    most_add_dec=$(echo "ibase=2;$most_add" | bc)
    les_add=${add:8:16}
    grep "^$most_add" addresses.txt >> add_inter.txt
    sed -i "/$add/d" add_inter.txt
#Check if the tier is the first one
    if [[ $les_add == "00000000" ]] &&
        [[ $most_add == "00000000" ]]

```

```

    then
        echo "000000001000000000" >> add_inter.txt
    elif [[ $les_add == "00000001" ]] &&
        [[ $most_add == "00000000" ]]
        then
            echo "000000001000000001" >> add_inter.txt
#check if the tier is the last one
    elif [[ $les_add == "00000000" ]] &&
        [[ $most_add == $last_tier_bin ]]
        then
            echo ""$before_last_bin"00000000" >> add_inter.txt
    elif [[ $les_add == "00000001" ]] &&
        [[ $most_add == $last_tier_bin ]]
        then
            echo ""$before_last_bin"00000001" >> add_inter.txt
#check if the tier is intermediate one
    elif (( $most_add > 0 )) &&
        (( $most_add < $last_tier_bin ))
        then
if [ $les_add == "00000000" ]
    then
        upper_tier=$((most_add_dec+1))
        upper_tier_bin=${D2B[$upper_tier]}
        lower_tier=$((most_add_dec-1))
        lower_tier_bin=${D2B[$lower_tier]}
        echo ""$lower_tier_bin"00000000" >> add_inter.txt
        echo ""$upper_tier_bin"00000000" >> add_inter.txt
    elif [ $les_add == "00000001" ]
        then
            upper_tier=$((most_add_dec+1))
            upper_tier_bin=${D2B[$upper_tier]}
            lower_tier=$((most_add_dec-1))
            lower_tier_bin=${D2B[$lower_tier]}
            echo ""$lower_tier_bin"00000001" >> add_inter.txt
            echo ""$upper_tier_bin"00000001" >> add_inter.txt
        fi
    fi
#Creating the routing tables format to append them to
#top module file
    sed -i -e "s/^/16'b/" add_inter.txt
    count=$((count+1))
    paste -d, -s add_inter.txt >> inter_routing.txt
done < addresses.txt
no_router=0;
line_add=0;
while read line_add; do

```

```

    echo "assign ram_values_"$no_router" = {"$line_add"};"
    >> routing_tables.txt
    no_router=$((no_router+1))
done < inter_routing.txt
#Cleaning up
rm -rf add_inter.txt
rm -rf inter_routing.txt

*Mesh_Topology.sh
#!/bin/sh
#Description: a shell script which is responsible of
#creating the right syntax of instantiations of the
#Router_mesh module and the routers routing tables then
#append them in the topmod.sv file. Also, the parameters
#in the param.h according to the user entries.
#Author: Maha Beheiry
#Argument bypassed through the GUI
Number_of_Routers_per_tier=("$@")
#####
#Cleaning up the current working directory
rm -rf connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf mesh_connections.txt
rm -rf wires_number.txt
rm -rf wires_decl.txt
#####
#Counting all the wires required in the system
design_file=Design_files/topmod.sv
param_file=Design_files/param.h
Number_of_wires_in_system=0;
Number_of_routers=0;
total_routers=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    Number_of_wires_in_system=$((2 + (($Number_of_routers - 1)
    * $Number_of_routers) / 2) + $Number_of_wires_in_system))
    total_routers=$((total_routers+$Number_of_routers))
done
Number_of_wires_in_system=$((($Number_of_wires_in_system - 2))
echo "\define NUM_OF_WIRES "$Number_of_wires_in_system""
>> wires_number.txt
total_routers=$((total_routers-1))
echo "output [63:0] R_packet ["$total_routers":0];"
>>routers_total.txt

```

```

#Append here number of wires to the param.h file
#and all router number
sed -i '/wires/,/end_here/{//!d}' $param_file
sed -i '/wires/rwires_number.txt' $param_file
sed -i '/r_packet/,/end_r/{//!d}' $design_file
sed -i '/r_packet/routers_total.txt' $design_file
rm -rf wires_number.txt
rm -rf routers_total.txt
#####
for (( n = 0 ; n < $Number_of_wires_in_system ; n++ ))
do
wires[$n]=$n
done
#####
m=0;
tier_count=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
tier_count=$((tier_count+1))
Number_of_routers=$i
#####
#2D_connections
for (( j = 0 ; j < $Number_of_routers - 1 ; j++ ))
do
k=$j+1
for (( k = $j+1 ; k < $Number_of_routers ; k++ ))
do
printf "R["$j"] "$tier_count" links["${wires[$m]}"],\n"
>> connections.txt
printf "R["$k"] "$tier_count" links["${wires[$m]}"],\n"
>> connections.txt
m=$((m+1))
done
done
done
#####
#3D_connections
for (( l = 1 ; l < $tier_count ; l++ ))
do
printf "R[0] "$l" links["${wires[$m]}"],\n"
>> connections.txt
printf "R[0] "$((l+1))" links["${wires[$m]}"],\n"
>> connections.txt
printf "R[1] "$l" links["${wires[$((m+1))]}"],\n"
>> connections.txt
printf "R[1] "$((l+1))" links["${wires[$((m+1))]}"],\n"

```



```

    >> connections.txt
    m=$((m +2))
done
#####
#Preparing the final connections
tier_count_1=0;
for i in "${Number_of_Routers_per_tier[@]}"
do
    Number_of_routers=$i
    tier_count_1=$((tier_count_1+1))
    for (( h = 0 ; h < $Number_of_routers ; h++ ))
    do
        grep -w "R\[\"$h\"\]" "$tier_count_1" connections.txt
| awk '{print}' ORS=' ' >> inter.txt
        echo $'\n' >> inter.txt
        sed -i 's/, $//' inter.txt
        sed -i '/^$/d' inter.txt
        sed -i 's/R\[\"[0-9][0-9]*\" [0-9][0-9]* //g' inter.txt
        sed -i 's/ //g' inter.txt
    done
done
#####
#Rearranging each line from MS to LS
while read line; do
    conct=$line
    IFS=', ' read -a array <<< "$conct"
length=${#array[*]}
    for ((i=${#array[@]}-1; i>=0; i--)); do
        printf "\"${array[$i]}\", " >> inter_rev.txt
    done
echo $'\n' >> inter_rev.txt
sed -i 's/, $//' inter_rev.txt
sed -i '/^$/d' inter_rev.txt
done < inter.txt
#####
#Adding the addresses to an array
IFS=$'\n' read -d '' -r -a addresses < addresses.txt
#Preparing the final version of the mesh_connections for
#each router per each tier with Verilog required syntax
lines_count=0
tier_count_0=0
a=0;
tiers=${#Number_of_Routers_per_tier[@]}
while read line; do
    lines[$a]=$line
    a=$((a +1))

```

```

done < inter_rev.txt
b=0
for i in "${Number_of_Routers_per_tier[@]}"
do
    tier_count_0=$((tier_count_0+1))
    Number_of_routers=$i
    for (( j = 0 ; j < $Number_of_routers; j++ ))
    do
        add=${addresses[$b]}
        if [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 == "1" ]
        then
            printf "Router_mesh #(.address(16'b"$add"),
            .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER)) router_"$b"
            (.port( {"${lines[$lines_count]}" } ),.clk(clk),
            .packet(packet),.ram_values(ram_values_"$b"),
            .R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt
            printf "wire [15:0] ram_values_"$b"
            [\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n" >> wires_decl.txt
            elif [ $j == "0" ] && [ $b != "0" ] && [ $tier_count_0 == "1" ]
            then
                printf "Router_mesh #(.address(16'b"$add"),
                .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER))
                router_"$b" (.port( {"${lines[$lines_count]}" } ),.clk(clk),
                .packet(),.ram_values(ram_values_"$b"),
                .R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt
                printf "wire [15:0] ram_values_"$b"
                [\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n"
                >> wires_decl.txt
            elif [ $j == "1" ] && [ $tier_count_0 == "1" ]
            then
                printf "Router_mesh #(.address(16'b"$add"),
                .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER))
                router_"$b" (.port( {"${lines[$lines_count]}" } ),
                .clk(clk),.packet(),.ram_values(ram_values_"$b"),
                .R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt
                printf "wire [15:0] ram_values_"$b"
                [\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n"
                >> wires_decl.txt
            elif [ $j == "0" ] && [ $b == "0" ] &&
            [ $tier_count_0 == $tiers ]
            then
                printf "Router_mesh #(.address(16'b"$add"),
                .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER))
                router_"$b" (.port( {"${lines[$lines_count]}" } ),
                .clk(clk),.packet(packet),.ram_values(ram_values_"$b"),
                .R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt

```

```

                printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n"
>> wires_decl.txt
                elif [ $j == "0" ] && [ $b != "0" ]
&& [ $tier_count_0 == $tiers ]
                then
                    printf "Router_mesh #(.address(16'b"$add"),
.ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER))
router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.ram_values(ram_values_"$b"),
.R_packet(R_packet ["$b"]));\n"
>> mesh_connections.txt
                printf "wire [15:0] ram_values_"$b" [
\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n"
>> wires_decl.txt
                    elif [ $j == "1" ] && [ $tier_count_0 == $tiers ]
                    then
                        printf "Router_mesh #(.address(16'b"$add"),
.ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER))
                        router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.ram_values(ram_values_"$b"),
.R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt
                        printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1:0];\n"
>> wires_decl.txt
                            elif [ $j == "0" ] && [ $b == "0" ] &&
[ $tier_count_0 != "1" ]
                            then
                                printf "Router_mesh #(.address(16'b"$add"),
.ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER+1))
                                router_"$b" (.port( {"${lines[$lines_count]}" } )
,.clk(clk),
.packet(packet),.ram_values(ram_values_"$b"),
.R_packet(R_packet ["$b"]));\n"
>> mesh_connections.txt
                                    printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER:0];
\n" >> wires_decl.txt
                                        elif [ $j == "0" ] && [ $b != "0" ] &&
[ $tier_count_0 != "1" ]
                                        then
                                            printf "Router_mesh #(.address(16'b"$add"),
.ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER+1))
router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.ram_values(ram_values_"$b"),
.R_packet(R_packet ["$b"]));\n" >> mesh_connections.txt

```

```

        printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER:0];\n" >>
wires_decl.txt
        elif [ $j == "1" ] && [ $tier_count_0 != "1" ]
        then
            printf "Router_mesh #(.address(16'b"$add"),
            .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER+1))
            router_"$b" (.port( {"${lines[$lines_count]}" } ),.clk(clk),
            .packet(),.ram_values(ram_values_"$b"),.R_packet(R_packet
            ["$b"]));\n" >> mesh_connections.txt
            printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER:0];\n"
            >> wires_decl.txt
        else
            printf "Router_mesh #(.address(16'b"$add"),
            .ramSize(\`NO_OF_ROUTERS_"$tier_count_0"_TIER-1))
            router_"$b" (.port( {"${lines[$lines_count]}" } ),.clk(clk),.
            packet(),.ram_values(ram_values_"$b"),.R_packet(R_packet ["$b"]));
            \n" >> mesh_connections.txt
            printf "wire [15:0] ram_values_"$b"
[\`NO_OF_ROUTERS_"$tier_count_0"_TIER-2:0];\n"
            >> wires_decl.txt
        fi
lines_count=$(( $lines_count + 1))
b=$(( $b + 1))
done
done
#####
#Append to the design file
sed -i '/ROUTER INSTANCES/,/END_Routers/{//!d}' $design_file
sed -i '/Lookup_tables/,/end_luts/{//!d}' $design_file
sed -i '/Wires_declartion/,/end_wires/{//!d}' $design_file
sed -i '/ROUTER INSTANCES/rmesh_connections.txt' $design_file
sed -i '/Lookup_tables/rrouting_tables.txt' $design_file
sed -i '/Wires_declartion/rwires_decl.txt' $design_file
#####
#Cleaning up
rm -rf connections.txt
rm -rf mesh_connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf addresses.txt
rm -rf routing_tables.txt
rm -rf wires_decl.txt
#####
#popup a finished message in the tcl window

```

```
tclsh final_msg.tcl
```

```
*Ring_RoutingTables_Add.bash
```

```
#!/bin/sh
```

```
#Description:a shell script which is responsible of creating  
#the addresses for all the routers in the network and also  
#the ring connections between them in the 2D and 3D levels.
```

```
#Author: Maha Beheiry
```

```
###Printing the all the routers' addresses
```

```
#Clean up old addresses file
```

```
rm -rf addresses.txt
```

```
rm -rf add_inter.txt
```

```
rm -rf add_inter_all.txt
```

```
rm -rf inter_routing.txt
```

```
rm -rf routing_tables.txt
```

```
#####
```

```
Number_of_Routers_per_tier=("$@")
```

```
D2B={0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}
```

```
tiers=${#Number_of_Routers_per_tier[@]}
```

```
last_tier=$((tiers-1))
```

```
last_tier_bin=${D2B[$last_tier]}
```

```
before_last=$((last_tier-1))
```

```
before_last_bin=${D2B[$before_last]}
```

```
tier_count_0=0
```

```
for i in "${Number_of_Routers_per_tier[@]}"
```

```
do
```

```
    Number_of_routers=$i
```

```
    for (( j = 0 ; j < $Number_of_routers; j++ ))
```

```
    do
```

```
        D2B={0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}{0..1}
```

```
        tier_no=${D2B[$tier_count_0]}
```

```
        address_no=${D2B[$j]}
```

```
        echo "$tier_no"$address_no" >> addresses.txt
```

```
    done
```

```
    tier_count_0=$((tier_count_0+1))
```

```
done
```

```
#####
```

```
#Creating the routing table for each router
```

```
while read add; do
```

```
    rm -rf add_inter.txt
```

```
    rm -rf add_inter_all.txt
```

```
    most_add=${add:0:8}
```

```
    most_add_dec=$(echo "ibase=2;$most_add" | bc)
```

```
    les_add=${add:8:16}
```

```
    #2D connection in routing table
```

```
    grep "^$most_add" addresses.txt >> add_inter_all.txt
```

```

IFS=$'\n' read -d '' -r -a array_add < add_inter_all.txt
number_add=${#array_add[@]}
last_add_2d=0;
prev_last=0;
last_add_2d=$((number_add-1))
prev_last=$((last_add_2d-1))
if [ $add == ${array_add[0]} ]
then
echo ${array_add[1]} >> add_inter.txt
echo ${array_add[$last_add_2d]} >> add_inter.txt
elif [ $add == ${array_add[$last_add_2d]} ]
then
echo ${array_add[0]} >> add_inter.txt
echo ${array_add[$prev_last]} >> add_inter.txt
else
q=0;
count_array=0;
for q in "${array_add[@]}"
do
next_one=0;
prev_one=0;
if [ "$q" == "$add" ] ; then
next_one=$((count_array+1))
prev_one=$((count_array-1))
echo ${array_add[$prev_one]} >> add_inter.txt
echo ${array_add[$next_one]} >> add_inter.txt
fi
count_array=$((count_array+1))
done
fi

#3D connections in routing table
#Check if the tier is the first one
if [[ $les_add == "00000000" ]] && [[ $most_add == "00000000" ]]
then
echo "000000001000000000" >> add_inter.txt
elif [[ $les_add == "00000001" ]] && [[ $most_add == "00000000" ]]
then
echo "000000001000000001" >> add_inter.txt
#check if the tier is the last one
elif [[ $les_add == "00000000" ]] &&
[[ $most_add == $last_tier_bin ]]
then
echo ""$before_last_bin"00000000" >> add_inter.txt
elif [[ $les_add == "00000001" ]] &&
[[ $most_add == $last_tier_bin ]]

```

```

        then
            echo ""$before_last_bin"00000001" >> add_inter.txt
#check if the tier is intermediate one
            elif [ $most_add_dec \> 0 ] &&
[ $most_add_dec \< $last_tier ]
                then
                    upper_tier=$((most_add_dec+1))
                    upper_tier_bin=${D2B[$upper_tier]}
                    lower_tier=$((most_add_dec-1))
                    lower_tier_bin=${D2B[$lower_tier]}
if [ $les_add == "00000000" ]
then
    echo ""$lower_tier_bin"00000000" >> add_inter.txt
    echo ""$upper_tier_bin"00000000" >> add_inter.txt
    elif [ $les_add == "00000001" ]
        then
            echo ""$lower_tier_bin"00000001" >> add_inter.txt
            echo ""$upper_tier_bin"00000001" >> add_inter.txt
fi
fi
#Creating the routing tables format to append them
#to top module file
sed -i -e "s/^/16'b/" add_inter.txt
routing_line="$(paste -d, -s add_inter.txt)"
echo $routing_line >> inter_routing.txt
done < addresses.txt
count=0
echo "initial" >> routing_tables.txt
echo "begin" >> routing_tables.txt
while read line_route; do
    echo "router_"$count".ram={"$line_route"};"
    >> routing_tables.txt
    count=$((count+1))
done < inter_routing.txt
echo "end" >> routing_tables.txt
#Cleaning up
rm -rf add_inter.txt
rm -rf add_inter_all.txt
rm -rf inter_routing.txt

*Ring_Topology.sh

#!/bin/sh
#Description: a shell script which is responsible of creating
#the right syntax of instantiations of the Router_ring
#module then append them in the topmod.sv file.

```

```

#Also, the parameters in the param.h according to the user
#entries. Please note that the ring routers have no routing tables.
#Author: Maha Beheiry
#Argument bypassed through the GUI
Number_of_Routers_per_tier=("$@")
#####
#Cleaning up the current working directory
rm -rf connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf ring_connections.txt
#####
#Counting all the wires required in the system
param_file=Design_files/param.h
Number_of_wires_in_system=0;
total_routers=0;
Number_of_routers=0;
  for i in "${Number_of_Routers_per_tier[@]}"
  do
    Number_of_routers=$i
    Number_of_wires_in_system=$((2 + $Number_of_routers
+ $Number_of_wires_in_system))
    total_routers=$((total_routers+$Number_of_routers))
  done
  Number_of_wires_in_system=$((Number_of_wires_in_system - 2))
  echo "\`define NUM_OF_WIRES \"$Number_of_wires_in_system\""
  >> wires_number.txt
  total_routers=$((total_routers-1))
  echo "output [63:0] R_packet [\"$total_routers\":0];"
  >>routers_total.txt
#Append here number of wires to the param.h file
sed -i '/wires/,/end_here/{//!d}' $param_file
sed -i '/wires/rwires_number.txt' $param_file
sed -i '/r_packet/,/end_r/{//!d}' $design_file
sed -i '/r_packet/routers_total.txt' $design_file
rm -rf routers_total.txt
rm -rf wires_number.txt
#####
for (( n = 0 ; n < $Number_of_wires_in_system ; n++ ))
do
  wires[$n]=$n
done
#####
m=0;
tier_count=0;
  for i in "${Number_of_Routers_per_tier[@]}"

```



```

do
  tier_count=$((tier_count+1))
  Number_of_routers=$i
#####
#2D_connections
  last_router=0;
  for (( j = 0 ; j < $Number_of_routers -1 ; j++ ))
  do
    next_router=0;
    next_router=$((j+1))
    if [ $j == "0" ]
    then
      d=$((m+1))
      last_router=$((Number_of_routers-1))
      printf "R["$j"] "$tier_count" links["${wires[$m]}"],\n"
    >> connections.txt
      printf "R["$last_router"] "$tier_count" links["${wires[$m]}"]
      ,\n" >> connections.txt
      printf "R["$j"] "$tier_count" links["${wires[$d]}"]
      ,\n" >> connections.txt
      printf "R["$next_router"] "$tier_count" links["${wires[$d]}"]
      ,\n" >> connections.txt
      m=$((m+2))
    else
      printf "R["$j"] "$tier_count" links["${wires[$m]}"]
      ,\n" >> connections.txt
      printf "R["$next_router"] "$tier_count" links["${wires[$m]}"]
      ,\n" >> connections.txt
      m=$((m+1))
    fi
  done
done
#####
#3D_connections
  for (( l = 1 ; l < $tier_count ; l++ ))
  do
    printf "R[0] "$l" links["${wires[$m]}"],\n" >> connections.txt
    printf "R[0] "$(($l+1))" links["${wires[$m]}"],\n" >> connections.txt
    printf "R[1] "$l" links["${wires[$(($m+1))]}"],\n" >> connections.txt
    printf "R[1] "$(($l+1))" links["${wires[$(($m+1))]}"],\n"
    >> connections.txt
    m=$((m +2))
  done
#####
#Preparing the final connections
tier_count_1=0;

```

```

for i in "${Number_of_Routers_per_tier[@]}"
do
  Number_of_routers=$i
  tier_count_1=$((tier_count_1+1))
  for (( h = 0 ; h < $Number_of_routers ; h++ ))
  do
    grep -w "R\[\"$h\"\]" "$tier_count_1" connections.txt |
awk '{print}' ORS=' ' >> inter.txt
    echo $'\n' >> inter.txt
    sed -i 's/, $//' inter.txt
    sed -i '/^$/d' inter.txt
    sed -i 's/R\[([0-9][0-9]*\) [0-9][0-9]* //g' inter.txt
    sed -i 's/ //g' inter.txt
  done
done
#####
#Rearranging each line from MS to LS
count_occ=1;
var=links;
while read line1; do
  number_occ[$count_occ]=$(echo $line1 |grep -o "links"| wc -l)
  count_occ=$((count_occ+1))
done < inter.txt
for ((f=0; f<${#number_occ[@]}; f++));
do
  if [[ ${number_occ[$f]} = "2" ]]
  then
    u=$((f+1))
    number_occ[$u]="*"
    f=$((f+1))
  fi
done
occ=1;
while read line; do
  conct=$line
  if [[ ${number_occ[$occ]} = "*" ]]
then
  IFS=', ' read -a array <<< "$conct"
  length=${#array[*]}
  for ((i=${#array[@]}-1; i>=0; i--)); do
    printf ""${array[$i]}", " >> inter_rev.txt
  done
echo $'\n' >> inter_rev.txt
sed -i 's/,$//' inter_rev.txt
sed -i '/^$/d' inter_rev.txt
else

```

```

    echo $line >> inter_rev.txt
fi
occ=$((occ+1))
done < inter.txt

#####
#Adding the addresses to an array
IFS=$'\n' read -d '' -r -a addresses < addresses.txt
#Preparing the final version of the ring_connections for each router per
#each tier with Verilog required syntax
lines_count=0
tier_count_0=0
a=0;
tiers=${#Number_of_Routers_per_tier[@]}
while read line; do
    lines[$a]=$line
    a=$((a + 1))
done < inter_rev.txt
b=0
for i in "${Number_of_Routers_per_tier[@]}"
do
    tier_count_0=$((tier_count_0+1))
    Number_of_routers=$i
    for (( j = 0 ; j < $Number_of_routers; j++ ))
    do
        add=${addresses[$b]}
        if [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 == "1" ]
        then
            printf "Router_ring #(.address(16'b"$add"),.ramSize(3))
router_"$b" (.port( {"${lines[$lines_count]}" } ),.clk(clk)
,.packet(packet),
.R_packet(R_packet ["$b"]));\n" >> ring_connections.txt
            elif [ $j == "0" ] && [ $b != "0" ] && [ $tier_count_0 == "1" ]
            then
                printf "Router_ring #(.address(16'b"$add"),
.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),.
clk(clk),.packet(),.R_packet(R_packet ["$b"]));\n"
                >> ring_connections.txt
            elif [ $j == "1" ] && [ $tier_count_0 == "1" ]
            then
                printf "Router_ring #(.address(16'b"$add")
,.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.R_packet(R_packet ["$b"]));\n"
                >> ring_connections.txt
            elif [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 == $tiers ]
            then

```

```

printf "Router_ring #(.address(16'b"$add")
.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(packet),.R_packet(R_packet ["$b"]));\n"
>> ring_connections.txt
    elif [ $j == "0" ] && [ $b != "0" ] && [ $tier_count_0 == $tiers ]
    then
        printf "Router_ring #(.address(16'b"$add"),
.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.R_packet(R_packet ["$b"]));\n"
>> ring_connections.txt
    elif [ $j == "1" ] && [ $tier_count_0 == $tiers ]
    then
        printf "Router_ring #(.address(16'b"$add"),
.ramSize(3)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.R_packet(R_packet ["$b"]));\n"
    >> ring_connections.txt
elif [ $j == "0" ] && [ $b == "0" ] && [ $tier_count_0 != "1" ]
    then
        printf "Router_ring #(.address(16'b"$add"),.ramSize(4))
router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(packet));\n" >> ring_connections.txt
    elif [ $j == "0" ] && [ $b != "0" ] &&
[ $tier_count_0 != "1" ]
        then
            printf "Router_ring #(.address(16'b"$add"),
.ramSize(4)) router_"$b"
(.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.R_packet(R_packet ["$b"]));\n"
>> ring_connections.txt
    elif [ $j == "1" ] && [ $tier_count_0 != "1" ]
    then
        printf "Router_ring #(.address(16'b"$add"),
.ramSize(4)) router_"$b" (.port( {"${lines[$lines_count]}" } )
,.clk(clk),.packet(),.R_packet
(R_packet ["$b"]));\n" >> ring_connections.txt
    else
        printf "Router_ring #(.address(16'b"$add"),.
ramSize(2)) router_"$b" (.port( {"${lines[$lines_count]}" } ),
.clk(clk),.packet(),.R_packet
(R_packet ["$b"]));\n" >> ring_connections.txt
    fi
    lines_count=$((lines_count + 1))
    b=$((b + 1))
done
done
#####

```

```

#Append to the design file
design_file=Design_files/topmod.sv
sed -i '/ROUTER INSTANCES/,/END_Routers/{//!d}' $design_file
sed -i '/ROUTING TABLES/,/END_Tables/{//!d}' $design_file
sed -i '/ROUTER INSTANCES/r/ring_connections.txt' $design_file
sed -i '/Wires_declartion/,/end_wires/{//!d}' $design_file
sed -i '/Lookup_tables/,/end_luts/{//!d}' $design_file
#sed -i '/ROUTING TABLES/r/routing_tables.txt' $design_file
#####
#Cleaning up
rm -rf connections.txt
rm -rf ring_connections.txt
rm -rf inter.txt
rm -rf inter_rev.txt
rm -rf addresses.txt
#####
#popup a finished message in the tcl window
tclsh final_msg.tcl

*final_msg.tcl
#Description: a tcl script to display
# the final info message after creating the design files.
#Author: Maha Beheiry
#!/usr/local/bin/wish
package require Tk
#package require BWidget
#frame .message
set ans [tk_messageBox -icon info -title
"3D NoCs Designs" -message
"Please check the design files created
under Design_files directory." -type ok ]
switch -- $ans {
ok { destroy . }
}
*Design Files
*param.h
//This file includes the paramters needed to the design.
//Number of routers per tier in the design,
//they vary according to the user entry.
//routers_tier
'define NO_OF_ROUTERS_1_TIER 4
//end_tiers
//wires
'define NUM_OF_WIRES 6

*topmod.sv

```

```

//This file is the top module for the design which
// includes the instantiation of the routers and its routing tables.
#include "param.h"
module topmod (packet, clk, R_packet);
input [63:0] packet;
input      clk;
wire [63:0] links  ['NUM_OF_WIRES-1:0];
//r_packet
output [63:0] R_packet  [3:0];
//end_r
//Wires_declartion
wire [15:0] ram_values_0 ['NO_OF_ROUTERS_1_TIER-1:0];
wire [15:0] ram_values_1 ['NO_OF_ROUTERS_1_TIER-1:0];
wire [15:0] ram_values_2 ['NO_OF_ROUTERS_1_TIER-2:0];
wire [15:0] ram_values_3 ['NO_OF_ROUTERS_1_TIER-2:0];
//end_wires
//ROUTER INSTANCES
Router_mesh #(.address(16'b0000000000000000),
.ramSize('NO_OF_ROUTERS_1_TIER))
router_0 (.port( {links[2],links[1],links[0]} ),
.clk(clk),.packet(packet),
.ram_values(ram_values_0),.R_packet(R_packet [0]));
Router_mesh #(.address(16'b0000000000000001),
.ramSize('NO_OF_ROUTERS_1_TIER))
router_1 (.port( {links[4],links[3],links[0]} ),
.clk(clk),.packet(),
.ram_values(ram_values_1),.R_packet(R_packet [1]));
Router_mesh #(.address(16'b0000000000000010),
.ramSize('NO_OF_ROUTERS_1_TIER-1)) router_2
(.port( {links[5],links[3],links[1]} ),.clk(clk),
.packet(),.ram_values(ram_values_2),.R_packet(R_packet [2]));
Router_mesh #(.address(16'b0000000000000011),
.ramSize('NO_OF_ROUTERS_1_TIER-1))
router_3 (.port( {links[5],links[4],links[2]} ),
.clk(clk),.packet(),.ram_values(ram_values_3),.R_packet(R_packet [3]));
//END_Routers
//Lookup_tables
assign ram_values_0 = {16'b0000000000000001,16'b0000000000000010,
16'b0000000000000011,16'b0000000100000000};
assign ram_values_1 = {16'b0000000000000000,16'b0000000000000010,
16'b0000000000000011,16'b0000000100000001};
assign ram_values_2 = {16'b0000000000000000,16'b0000000000000001,
16'b0000000000000011};
assign ram_values_3 = {16'b0000000000000000,16'b0000000000000001,
16'b0000000000000010};
//end_luts

```

```

endmodule

*testBn.sv
//This file includes the packets to be sent through
//the network in series
//TestBench
#include "param.h"
module testBn ();
reg [63:0] pp ;
reg clk;
topmod temp(.packet(pp), .clk(clk), .R_packet() );
initial
begin
# 10
pp = 64'b11111111_11111111_00000001_00000011__11111111_
11111111_11111111_11111111 ;
//# 10
//pp = 64'b11111111_11111111_00000000_00000011__11111111_
11111111_11111111_11111111 ;
//# 10
//pp = 64'b11111111_11111111_00000010_00000001__11111111_
11111111_11111111_11111111 ;
// # 10
// pp = 64'b11111111_11111111_00000001_00000000__11111111_
11111111_11111111_11111111 ;
// # 10
// pp = 64'b11111111_11111111_00000001_00000001__11111111_
11111111_11111111_11111111 ;
// # 10
// pp = 64'b11111111_11111111_00000001_00000010__11111111_
11111111_11111111_11111111 ;
// # 10
// pp = 64'b11111111_11111111_00000001_00000011__11111111_
11111111_11111111_11111111 ;
# 10
pp= 'bz;

end
//Clock
initial
begin
clk = 0;
forever #5 clk = ~clk;
end

endmodule

```

```

//end_here

*router_module_mesh.sv
//This file includes the router module in the mesh topology
#include "param.h"
module Router_mesh (port, clk, packet, ram_values, R_packet);
//Module inputs,outputs and parameters
////////////////////////////////////
//Design inputs
input [63:0] packet; //packet transmitted
input      clk;
//Design parameters
parameter      ramSize=0; //ramsize
parameter [15:0] address=16'b0;//Router address
//Design inputs/outputs
inout [63:0] port      [ramSize-1:0]; // Router ports
input [15:0] ram_values [ramSize-1:0];
output [63:0]      R_packet;
//Design registers
wire [15:0] ram      [ramSize-1:0];// memory cells (16Bit Wide)
//Routing Table
reg [ramSize-1:0] enable ;
reg [63:0]      outlatch[ramSize-1:0];
reg [63:0]      inlatch[ramSize:0];
reg [63:0]      R_packet;
//Design bits
bit      flg; //Flag for 3D routers round robin
assign ram = ram_values;
////////////////////////////////////
genvar m;
generate
for(m=0; m<ramSize; m=m+1)
assign port [m]= enable[m] ? outlatch[m]:64'bz;
endgenerate

generate
for(m=0; m<ramSize; m=m+1)
assign inlatch [m]= port[m];

assign inlatch [ramSize]= packet;
endgenerate

// Logic of 3D Routing algorithm goes here with each clock rising edge
always @ (posedge clk)

```



```

begin
integer i;
integer j;
integer k;
    enable = 'b0;

for(k=0;k<ramSize+1;k=k+1)
//status <= "--";
for(k=0;k<ramSize+1;k=k+1)
begin
if (address == inlatch[k][47:32])
begin
$display("I am the destination");
R_packet <= inlatch[k];
//status[k] <= "destination";

// Save the packet in the inlatch of the router
//to keep it. (inlatch or final_destination register)
end
else if (address == inlatch[k][63:48])
begin
$display("I am the source");
//status[k] <= "source";

//
end
else if (address [15:8] != inlatch[k][47:40])
//Case of the source and destination not in the same tier.
begin
//status[k] <= "DTier-2DS";

if ( ~(address [7:0] == 8'b00000000 ||
address [7:0] == 8'b00000001)) // 2D router case
begin
for (i = 0; i<ramSize; i++)
begin
if (ram [i][7:0] == 0 && flg == 0 )
begin
outlatch[i] <= inlatch[k];
//Packet to be output on the inlatch
//connected to the router
outlatch[i][63:48] <= address;
enable[i]<=1;
flg<=1;
end
else if (ram [i][7:0] == 1 && flg == 1 )

```

```

begin
outlatch[i] <= inlatch[k];
//Packet to be output on the inlatch connected to the router
outlatch[i][63:48] <= address;
enable[i]<=1;
flg<=0;
end
end
end
else if (address [7:0] == 8'b00000000
|| address [7:0] == 8'b00000001) //3D router case
//status[k] <= "DTier-3DS";

begin
if (address [15:8] < inlatch[k][47:40])
begin
for (i = 0; i < ramSize; i++)
begin
if (ram [i][15:8] > address [15:8])
begin
//status[k]          <= "DTier-3DS-down";
outlatch[i]          <= inlatch[k];
//Packet to be output on the inlatch connected to the router
outlatch[i][63:48] <= address;
enable[i]            <=1;
end
end
end
else if(address [15:8] > inlatch[k][47:40])
begin
for (i = 0; i<ramSize; i++)
begin
//status[k] <= "DTier-3DS-UP";
if (ram [i][15:8] < address [15:8])
begin
outlatch[i] <= inlatch[k];
//Packet to be output on the inlatch connected to the router
outlatch[i][63:48] <= address;
enable[i]<=1;
end
end
end
end
end
////////////////////////////////////
else if (address [15:8] == inlatch[k][47:40])

```

```

//Case of the source and destination in the same tier
begin
$display("Destination and source are in the same tier");
//status [k] <= "STier";
for (j = 0; j<ramSize; j++)
begin
if (ram[j] == inlatch[k][47:32])
begin
outlatch [j] <= inlatch[k];
//Packet to be output on the port connected to the router
outlatch [j][63:48] <= address;
enable [j] <=1;
end
end
end
end
end
endmodule

```

```

*router_module_ring.sv
//This file includes the router module in the ring topology
#include "param.h"
module Router_ring (port, clk, packet, R_packet);
//Module inputs, outputs and parameters
////////////////////////////////////
//Design inputs
input [63:0] packet; //packet transmitted
input clk;
//Design parameters
parameter ramSize=0; //ramsize
parameter [15:0] address=16'b0; //Router address
//Design inputs/outputs
inout [63:0] port [ramSize-1:0]; // Router ports
output [63:0] R_packet;
//Design registers
reg [15:0] ram [ramSize-1:0];
// memory cells (16Bit Wide) //Routing Table
//reg [500:0] status [ramSize:0];
reg [ramSize-1:0] enable ;
reg [63:0] outlatch[ramSize-1:0];
reg [63:0] inlatch[ramSize:0];
reg [63:0] R_packet;
////////////////////////////////////
genvar m;
generate
for(m=0; m<ramSize; m=m+1)

```

```

assign port [m]= enable[m] ? outlatch[m]:64'bz;
endgenerate

generate
for(m=0; m<ramSize; m=m+1)
assign inlatch [m]= port[m];

assign inlatch [ramSize]= packet;
endgenerate

// Logic of 3D Routing algorithm goes here with each clock rising edge
always @ (posedge clk)
begin
integer i;
integer j;
integer k;
enable = 'b0;

for(k=0;k<ramSize+1;k=k+1)
//status <= "--";
for(k=0;k<ramSize+1;k=k+1)
begin
if (address == inlatch[k][47:32])
begin
$display("I am the destination");
R_packet <= inlatch[k];
//status[k] <= "destination";

// Save the packet in the inlatch of the router to keep it.
//(inlatch or final_destination register)
end
else if (address == inlatch[k][63:48])
begin
$display("I am the source");
//status[k] <= "source";

//
end
else if (address [15:8] != inlatch[k][47:40])
//Case of the source and destination not in the same tier.
begin
//status[k] <= "DTier-2DS";

if ( ~(address [7:0] == 8'b00000000 ||
address [7:0] == 8'b00000001)) // 2D router case
begin

```

```

$display("Destination and soure are in the same tier");
//status [k] <= "STier";
outlatch [1] <= inlatch[k];
//Packet to be output on the port connected to the router
outlatch [1][63:48] <= address;
enable [1] <=1;
end
else if (address [7:0] == 8'b00000000 ||
address [7:0] == 8'b00000001) //3D router case
//status[k] <= "DTier-3DS";

begin
if (address [15:8] > inlatch[k][47:40])
begin
outlatch [2] <= inlatch[k];
//Packet to be output on the port connected to the router
outlatch [2][63:48] <= address;
enable [2] <=1;
end
else if(address [15:8] < inlatch[k][47:40])
begin
outlatch [ramSize-1] <= inlatch[k];
//Packet to be output on the port connected to the router
outlatch [ramSize-1][63:48] <= address;
enable [ramSize-1] <=1;
end
end
end
//end
////////////////////////////////////
else if (address [15:8] == inlatch[k][47:40])
//Case of the soure and destination in the same tier
begin
$display("Destination and soure are in the same tier");
//status [k] <= "STier";
outlatch [1] <= inlatch[k];
//Packet to be output on the port connected to the router
outlatch [1][63:48] <= address;
enable [1] <=1;
end
end
end
endmodule

```

الملخص

إن تكنولوجيا "التكامل ثلاثي الأبعاد" تتيح فرصة كبيرة لتنفيذ تطبيقات متنوعة و قوية ، الي جانب ذلك تحل تكنولوجيا "الشبكة علي الرقاقة" التحديات الجديدة المتعلقة بطول الأسلاك عن طريق وسيلة اتصال مبتكرة ما بين النقاط المختلفة علي الرقاقة نفسها. و من ثم فإن إدماج تكنولوجيا "التكامل ثلاثي الأبعاد" مع "الشبكة علي الرقاقة" يؤدي لظهور حل جديد للعديد من التطبيقات التي كانت تبدو صعبة المنال من قبل.

تكنولوجيا "الشبكة علي الرقاقة" ثلاثية الأبعاد تأتي مع العديد من التحديات البحثية للحفاظ علي كفاءة الأداء، مثل التوصيلات العمودية وطريقة تصميمها ووضعها و كذلك طريقة تصنيعها. إن تحدي "خوارزمية التوجيه" يعد أحد أهم التحديات في تنفيذ تلك الخوارزمية و تطويرها لتكون مرنة و موثوق بها لتنجز التواصل الهام بين الطبقات ثلاثية الأبعاد.

و بناءً على ما سبق و كجزء أصيل من العمل علي تلك الرسالة، يتم تقديم خوارزمية للتوجيه ثلاثي الأبعاد (المصعد المباشر) المعتمد علي خوارزمية (المصعد أولاً ثلاثي الأبعاد) كمدخل مهم لحل المشكلات سابقة الذكر. خوارزمية (المصعد المباشر) لا تعتمد علي كم التوصيلات أو مكانها أو بنية الشبكة ، كذلك فهي مصنوعة خصيصاً للتركيبات ثلاثية الأبعاد المختلفة. إنها تتيح كذلك وقت استجابة أقل مقارنة بخوارزمية "المصعد أولاً ثلاثي الأبعاد".

تطرح تلك الرسالة أداة جديدة و مبتكرة يتم توفير من خلالها حل لتنفيذ شبكات ثلاثية الأبعاد بشكل عام و ذلك لخدمة مختلف التطبيقات والتصميمات. الأداة المطروحة من خلال الرسالة و المسماه "3D-NOCET" تعتمد علي خوارزمية (المصعد المباشر). أداة "3D-NOCET" تسمح للمستخدم أن يصنع تشكيلات مختلفة من الشبكات ثلاثية الأبعاد معتمداً علي خوارزمية التوجيه ثنائي الأبعاد وعدد الطبقات و عدد الموجهات في كل طبقة. كل ذلك عن طريق بنية تحتية اوتوماتيكية تماماً مصممة خصيصاً لهذا الغرض.

هذا يمهد الطريق للقيام بتقييمات و قياسات مختبرية مختلفة للشبكات ثلاثية الأبعاد المتنوعة. تلك التقييمات المختبرية المستقبلية و التحليل الأدائي المقارن سيساعدا بدورهما علي ايجاد التركيبات الشبكية الأفضل للتطبيقات المختلفة.



مهندس: مها رمضان محمد بحيري
تاريخ الميلاد: 1990\6\25
الجنسية: مصرية
تاريخ التسجيل: 2012\9\20
تاريخ المنح: 2017/--/--
القسم: هندسة الإلكترونيات والاتصالات الكهربائية
الدرجة: ماجستير العلوم
المشرفون:

ا.د. أحمد محمد سليمان

د. حسن مصطفى حسن مصطفى

المتحنون:

أ.د. (المتحن الخارجي)
أ.د. (المتحن الداخلي)
أ.د. (المشرف الرئيسي)
أ.د. (عضو)

عنوان الرسالة:

تصميم شبكة قابلة للضبط على رقاقة ثلاثية الأبعاد باستخدام خوارزمية التوجيه المسماه بالمصعد المباشر ثلاثي الأبعاد.

الكلمات الدالة:

تكنولوجيا ثلاثية الأبعاد، خوارزمية توجيه ثلاثية الأبعاد، شبكات علي رقاقة ثلاثية الأبعاد، مولد لشبكات علي رقاقة ثلاثية الأبعاد.

ملخص الرسالة:

الهدف الرئيسي من هذه الرسالة هو مساعدة المصممين بأداة لتكوين تركيبات متغيرة لشبكة علي رقاقة ثلاثية الأبعاد. هذه الأداة ستستخدم لتحديد أفضل التركيبات لتصميم أو تطبيق محدد ، و قد تم بناء تصميم هذه الأداة بناءً علي خوارزمية التوجيه المعروفة اسماً بالمصعد المباشر ثلاثي الأبعاد. في هذه الرسالة يتم عرض تحليل مفصل لهذه الأداة و يشمل حالات مختبرية عديدة لتقييم المرونة و الفاعلية الخاصة بها.

و يتم أيضا عرض المقارنات الي تم تنفيذها من خلال محاكات ما بين المصعد المباشر و المصعد أولاً من خلال الوقت و المساحة و الطاقة المستنفذة خلال عمل الخوارزميات في توجيه المعلومات خلال الشبكات المتنوعة.

تصميم شبكة قابلة للضبط على رقاقة ثلاثية الأبعاد باستخدام خوارزمية
التوجيه المسماة بالمصعد المباشر ثلاثي الأبعاد

اعداد

مها رمضان محمد بحيري

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربائية

يعتمد من لجنة الممتحنين:

الممتحن الخارجي الاستاذ الدكتور:

الممتحن الداخلي الاستاذ الدكتور:

المشرف الرئيسى الاستاذ الدكتور:

عضو الاستاذ الدكتور:

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
شهر - سنة

تصميم شبكة قابلة للضبط على رقاقة ثلاثية الأبعاد باستخدام خوارزمية
التوجيه المسماة بالمصعد المباشر ثلاثي الأبعاد

اعداد

مها رمضان محمد بحيري

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربائية

تحت اشراف

اسم المشرف	اسم المشرف
د. حسن مصطفى حسن مصطفى	أ.د. أحمد محمد سليمان
مدرس	أستاذ متفرغ
قسم هندسة الإلكترونيات و الاتصالات الكهربية	قسم هندسة الإلكترونيات و الاتصالات الكهربية
كلية الهندسة - جامعة القاهرة	كلية الهندسة - جامعة القاهرة

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
شهر - سنة



تصميم شبكة قابلة للضبط على رقاقة ثلاثية الأبعاد باستخدام خوارزمية التوجيه المسماه بالمصعد المباشر ثلاثي الأبعاد

اعداد

مها رمضان محمد بحيري

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستير العلوم
في
هندسة الإلكترونيات و الاتصالات الكهربائية

كلية الهندسة - جامعة القاهرة
الجيزة - جمهورية مصر العربية
شهر - سنة