Formal Modeling of Network-on-Chip and its Applications in Starvation and Deadlock Detection and in Developing Deadlock Free Routing Algorithms

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by

Surajit Das

Under the guidance of

Dr. Chandan Karfa and Prof. Santosh Biswas

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

December, 2021

Dedicated to my

Aradhya

and

Parents & Wife

and

All Other Family Members

Who always picked me up on time
and encouraged me to go on every adventure,
specially this one.

Declaration

I, Surajit Das, hereby confirm that:

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisors.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.

[illegible]

....

Surajit Das
Research Scholar
Department of SCE
IIT Guwahati
Guwahati, Assam, India 781039
d.surajit@iitg.ac.in
Date:
Place:

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor *Dr. Chandan Karfa* for his consistent support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for his ethical beliefs and philosophy which made me mature not only as a scientific researcher but also as a human. I am grateful to my second supervisor *Prof. Santosh Biswas*, who guided me, encourage me in critical phase of my research. I am thankful for his motivation and consistent support during my research.

I am highly grateful to *Prof. Jatindra Kumar Deka* for his invaluable support and encouragement throughout my Ph.D. I would also like to thank the other members of my Doctoral Committee - *Prof. Purandar Bhaduri* and *Dr. John Jose* for their insightful comments and suggestions which made me improve the quality and clarity of my work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IITG - *Prof. Diganta Goswami*, *Prof. S. V. Rao* and *Prof. Jatindra Kumar Deka* for allowing me to use the facilities and the available resources. I would like to thank Prof. Sukumar Nandi, Prof. Hemangee K. Kapoor, Dr. Aryabartta Sahu, Dr. John Jose and all the other faculty members of Computer Science and Engineering, IITG, from whom I get opportunity to learn during my PhD.

I acknowledge the Technical staff of the Department of Computer Science and Engineering - Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah, Mr. Hemanta Kumar Nath, Mr. Pranjitt Talukdar, Mr. Raktajit Pathak and Mr. Nava Kumar Boro for solving any engineering related issues. I am deeply thankful to - Mr. Monojit Bhattacharjee, Ms. Gauri Khuttiya Deori and Mr. Prabin Bharali for efficiently handling the administrative work. I am obliged to all the the staff and security personnel for their constant help and support. I would also like to thank the staff at the Academic Affairs office who were supportive to process my applications and grant requests.

I would like to gratefully acknowledge *MHRD*, *Govt. of India* for the financial support rendered throughout my years of Ph.D. without which this research could not have taken shape.

I am indeed thankful to my fellow lab mates Ramanuj, Debabrata, Priyanka, Mohammad, Nilotpala, Alakesh, Ujjwal, Dipojwal, Swarup, Chitra, Pawan, Arijit, Palash, Parikshit, Partha, Pallabi, Nayantara, Dipika, Deepak, Sukarn, Saptarchi, Balaprakash, Manoj, Hemakumar and many more for creating a wonderful experience during my PhD. It was a good experience to work with Ajinka and Pavan for a few months and I like to thank them for being part of my PhD journey. I would like to thank my senior lab mates Dr. Shirshendu Das for his support and helping me in PARSEC benchmark. I would like to thank other senior lab mates Dr. Shounak Chakraborty , Dr. Mayank Agarwal, Dr. Pradeep Kumar Biswal, Dr. Shilpa Budhkar, Dr. Nilakanta Sahu, Dr. Manojit Gosh for their guidance. I would also like to thank all the research scholars of the department of Computer Science and Engineering at IITG for creating a warm atmosphere of mutual support and encouragement. The stimulating discussions, brainstorming and sleepless night working together contribute a significant portion towards my development as an independent researcher.

I want to thank the friends from my school days - Biswajit, Dipankar, Mukul, Pankaj, Rupam, Pulin, Pranab and many more for the beautiful memories and their encouragement for PhD. Finally yet importantly, I would like to thank Lord Krishna and my family - Maa, Deuta, Aradhya, Junuk, Dishant, Kakali, Dada, Baideu, Bau, Vindeu, Mama, Mami and Partha for their unconditional love, support, caring, warmth and profound encouragement all these years. They never doubted my intentions and whole-heartedly supported me in all my endeavours. I fall short of words to express my gratitude to them.

Surajit Das
Research Scholar
Department of SCE
IIT Guwahati
Guwahati, Assam, India 781039
d.surajit@iitg.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India

This is to certify that this thesis entitled, **"Formal Modeling of Network-on-Chip and its Applications in Starvation and Deadlock Detection and in Developing Deadlock Free Routing Algorithms"**, being submitted by **Mr. Surajit Das**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulation of the institute.

To the best of my knowledge, this thesis has not been submitted in part or full to any other university or institute for the award of any degree or diploma.

....	....
Dr. Chandan Karfa	**Prof. Santosh Biswas**
Associate Professor	Professor
Department of Computer Science and Engineering	Department of Electrical Engineering and Computer Science
IIT Guwahati	IIT Bhilai
Date:	Date:
Place:	Place:

Abstract

The increasing demand for high computation speed in devices ranging from a handheld smartphone to a time critical missile defense system along with the issue of heat generated by a high frequency processor motivates the researchers to explore Chip Multi-Processors (CMP) architecture. In CMP, multiple processors with moderate clock frequencies are integrated into single chip. For facilitating a fast and efficient communicating infrastructure between processors, a programmable network having routers connected to each processor is used in CMPs which is formally known as Network-on-Chip (NoC). A routing algorithm in NoC helps in directing packets efficiently between processors via routers. While routing packets, situations like starvation or deadlock may occur that degrade the performance of NoC. Formal modeling of NoC is helpful to detect such scenarios either by using a model checker or by developing a simulation framework on the formal model.

In this thesis, formal modeling of NoC using Finite State Machines (FSMs) is presented first. NoC components like a buffer for the temporary storage of packets, switch for directing packets towards proper output ports and arbiter for resolving conflict at an output port are modeled using FSMs. Two arbitration logic, namely fixed-priority arbiter and round-robin arbiter, are modeled as FSMs. The synchronization between NoC components is maintained using dedicated FSMs. The correctness of the FSM based NoC model is checked and verification of starvation freedom is also presented using model checker in this work. Since modeling synchronization amongst components in an NoC is a complex task in FSM model and model checking based verification of global properties like deadlock is not feasible due to large state space, we next model the NoC using Communicating Finite State Machines (CFSMs). In CFSM based NoC model, the synchronization between NoC components is maintained with the help of message passing. No special state machine for maintaining system wide synchronization is required. We have automated the CFSM based NoC model generation process for Mesh and Torus NoCs. A CFSM based formal simulation framework is then developed for application specific deadlock detection in NoC. The CFSM based simulation framework detects confirmed deadlock conditions in given traffic with respect to a given routing algorithm and given topology. On detecting a confirmed deadlock, the framework reports the deadlock with the instance of cyclic resource dependencies.

The instance of cyclic resource dependency obtained as an output from the CFSM based framework helps in formulating deadlock avoidance. Representation of deadlock in a formal and informative way helps in understanding the root cause of deadlock and in turn helps in avoiding them. The Turn model [1] and Channel Dependency Graph (CDG) [2] are two classical approaches used for theoretical deadlock detection and representation of deadlock scenarios. We have proposed Directional Dependency Graph (DDG) for representing deadlock cycles in an informative way by incorporating Turn, wraparound channel and direction information with CDG in a simplified form. The DDG is helpful in showing deadlock scenarios or deadlock freedom for a routing procedure. It is found that Torus NoC is more deadlock prone in comparison to Mesh NoC due to the presence of ring networks with inherent cyclic paths. Virtual Channels (VCs) or additional buffers are mostly used for avoiding deadlock in Torus NoC. Therefore, an Arc model is proposed for avoiding deadlock in Torus NoC without using additional resources like VC or additional buffer. Deadlock behaviour of the proposed Arc model with respect to XY-Turns is also presented. An approach of developing deadlock free routing algorithms using a subset of Arcs and a given set of Turns is also presented. Finally, the utility of the Arc Model and DDG are presented by demonstrating the deadlock free algorithms for Torus NoC. Our proposed Arc based routing algorithms are found to be efficient than the competitive deadlock free routing algorithms in Torus NoC.

Keywords- *Network-on-Chip*, *Formal Verification*, *Starvation*, *Deadlock Detection*, *Deadlock Avoidance*, *Finite State Machine*, *Communicating Finite State Machine*

Contents

List of Figures

List of Tables

List of Abbreviation

BDD	Binary Decision Diagram
CDG	Channel Dependency Graph
CFSM	Communicating Finite State Machine
CMP	Chip Multi Processor
DDG	Directional Dependency Graph
DFA	Deterministic Finite Automata
DP	Dynamic Programming
FIFO	First In First Out
FP	Fixed-Priority Arbiter
FSM	Finite State Machine
IoT	Internet of Things
LTL	Linear Temporal Logic
NoC	Network-on-Chip
NuSMV	New Symbolic Model Verifier
RC	Route Computation
RR	Round-Robin Arbiter
SGM	State Graph Manipulator
SMV	Symbolic Model Verifier
SoC	System-on-Chip
VC	Virtual Channel

1

Introduction

Moore's law states that the number of transistors on a microchip doubles in every two years [4]. On the flip side, Dennard scaling states that it is feasible to reduce transistor dimension and put more transistors in the same die area where overall power consumption remains almost the same [5]. To take advantage of the Dennard scaling, more number of transistors are packed into a die by reducing the transistor dimension. To keep the dynamic power consumption in a limit, clock frequencies in a processor are also drastically raised at the same time. After a certain point, the thermal limitation creates a bottleneck on increasing the number of transistors in single processor due to electric leakage and heat up problems. The state-of-the-art cooling mechanism also fails to dissipate the generated heat, which might burn out the chip. Therefore, computer architects are forced to move into Chip Multiprocessor (CMP) architecture to cope with the ever increasing demand for high computation speed in various fields ranging from a smartphone to a time critical missile defence system. The CMP is a multi-core system architecture with the integration of multiple processing units in the same die [6]. Instead of using a single processor with high frequency, multiple processors with moderate frequency are used in CMP to achieve the computation demand.

1.1 Network-on-Chip

Performance in a CMP is limited by the communication between cores or processing units. Therefore, an efficient means of communication between cores in a CMP is required for better performance. Using dedicated wire or common bus communication has lot of overhead including area and power. The use of an interconnection network where processors communicate via routers and share the bandwidth to route packets amongst processors is more efficient rather than dedicated wire connections. This interconnection network in CMP is called Network-On-Chip (NoC). In simple terms, NoC is a programmable network that transfers data between the processors (cores) via routers [3].

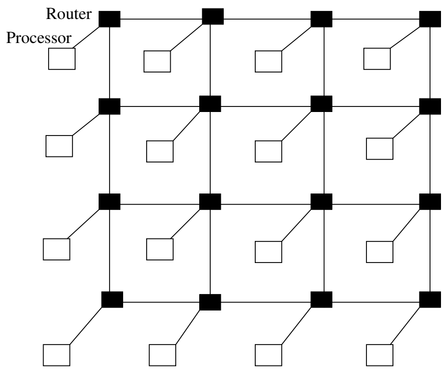


Figure 1.1: *A* 4x4 *Mesh NoC*

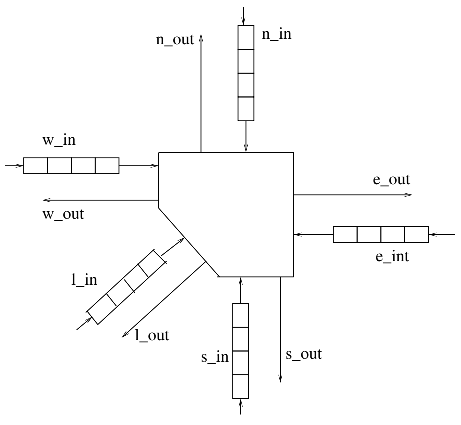


Figure 1.2: *5 Port NoC router*

Router is considered as the backbone of NoC. Based on the router interconnection, different topologies like Mesh, Torus, Butterfly, etc. have evolved [3]. A 4x4 Mesh NoC is shown in Fig. 1.1. The associated processors, routers and communication wires are shown in that figure. Routers are present at the intersection of the wires. Block diagram of a five port NoC router is shown in Fig. 1.2. A router has five input ports and five output ports named as Local (L), East (E), West (W), North (N) and South (S). Local input and output ports are used to communicate with the local core. The East input port channel is used to get packet from its neighbour in the East direction and the East output port channel is used to transmit packet towards the East neighbour. All other ports are used in a similar way.

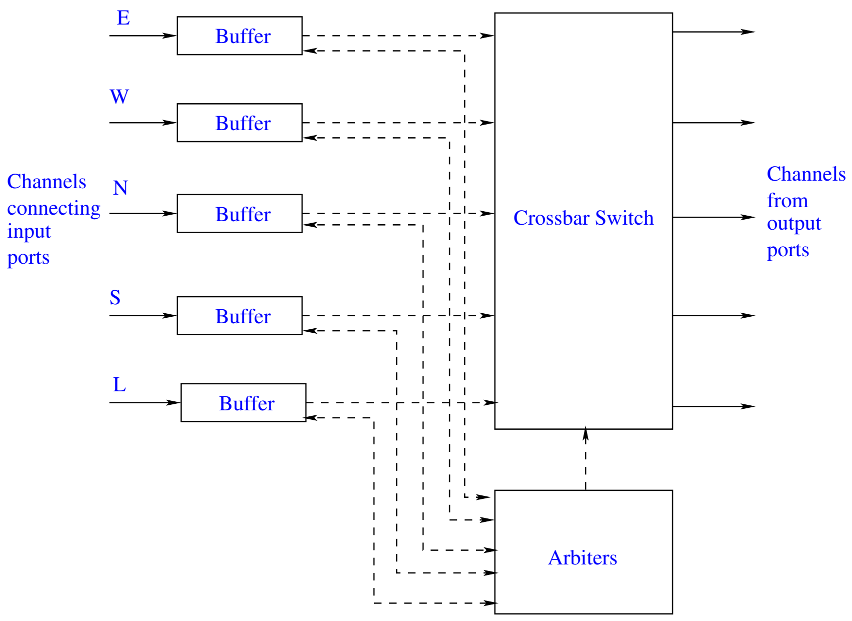


Figure 1.3: *Simplified block diagram depicting functional units in an NoC router [3]*

A simplified block diagram showing components of an NoC router with input buffers is shown in Fig. 1.3. The solid lines in the left hand side are input channels connecting input ports. A buffer is associated with each input channel where an incoming packet is stored until it is transmitted through the desired output port. The solid lines in the right hand side are channels from the output ports connecting adjacent routers or the local core. Connections between buffers and output ports within a router is configured by crossbar switch. Routing algorithms help in choosing the connection between an input port buffer and an output port. Some routing algorithms are static in nature, where the route to be followed is fixed based on the source and destination of the packet. On the other hand, some routing algorithms are adaptive in nature, where routes are decided based on current network parameters like congestion and connection failure [7, 8]. When packets from more than one input ports request the same output port at the same time, the conflict is resolved by an Arbiter. Arbiter resolves the conflict between packets at an output port using different scheduling policies like Round-robin or Fixed-priority or Weighted-round-robin, etc.

1.1.1 Correct Functioning of NoC

An NoC might suffer from scenarios like starvation or deadlock or livelock if sufficient measures are not taken according to the design specification in selecting a suitable routing algorithm or NoC topology or resource allocation scheme during the design phase [3]. Ideally, an NoC must be free from these issues. Deadlock is said to occur in an NoC when a group of packets are unable to make any progress because of the cyclic wait on one another for the release of resources e.g., channels or buffers [3]. Deadlock has catastrophic effects on a network. Initially, a few resources are occupied in holding and waiting mode by a set of packets. Eventually, more packets keep on getting blocked on these resources. These resource dependencies are spread across the network that halt the overall network functionalities. To get rid of such situation, deadlock avoidance or deadlock recovery has to be incorporated in an NoC. Designing a routing algorithm which is deadlock free with respect to a given topology has utmost importance in NoC. In deadlock avoidance approach, different precautionary measures like routing restrictions, using of Virtual Channels (VCs) or dedicated buffers are implemented to guarantee that deadlock does not occur [1, 9–14]. On the other hand, the deadlock is permitted to occur in deadlock recovery approach. With help of periodic operation, deadlock cycle is detected and actions are taken to break the resource dependency cycle for deadlock recovery [15–17]. Livelock, on the other hand, is a situation when packets keep on traveling indefinitely through the network without reaching their destinations [3]. An NoC is subject to starvation when some packets keep on waiting for resources indefinitely and are unable to make progress because access to certain resources are not granted to them [3]. This is due to accessing of those resources continuously by another class of packets. It is the responsibility of a designer to avoid these issues by applying suitable measures. This work aims at detection of such scenarios with the help of formal models. The detected scenario helps in understanding root causes of failure and in formulating their prevention as well.

1.1.2 Verification of NoC

In the pre-silicon evaluation of a system design, the design is tested or verified against a given set of design specifications before the actual chip is manufactured. This process targets verification of functional correctness and reliability of the design along with detection of different bugs [18–20]. It helps in rectifying defects in design phase itself. Pre-silicon evaluation is performed to avoid huge costs and time involved in post-silicon fixes. Full system simulators and formal verification methods are used for evaluating a system before manufacturing the actual hardware. In simulation based verification of a system, it needs to be evaluated for all possible inputs and checks if outputs are the same with the golden outputs. Exhaustive checking of all system behaviors covering all possible test cases is practically not feasible for a huge system. While simulators help for evaluating NoC in terms of throughput, overall performance, delay, power consumption, etc., formal verification method is suitable for ensuring compliance with crucial properties like starvation-freedom, deadlock-freedom, livelock-lock freedom, etc.. In formal verification, no exhaustive simulation is required. One requirement for formal verification is the need of a formal model of the system. The verification of the system is done by providing a formal proof on the formal model of the system. If the system under consideration has a large number of functional units, performing exhaustive testing to cover all possible corner cases for a crucial property is challenging as well as time consuming. Formal modeling and verification is helpful for such purposes. To ensure that verified properties function correctly after manufacturing the real hardware, modeling of NoC components in detail level is essential.

1.2 Challenges in Formal Modeling and Verification of NoC

This thesis targets verification of two properties, starvation and deadlock in NoC using its formal models. In the following, we have discussed about challenges in starvation and deadlock detection in NoC. We have also briefly discussed the existing works on formal modeling of NoC and their applications.

1.2.1 Starvation Freedom in NoC

Starvation-freedom ensures fairness in resource allocation. In NoC context, the output port and its corresponding buffer in the next router is the resource for conflict. If multiple packets from different input ports in a router try to exit through the same output port continuously, there should be a fairness such that competing packets from all input ports get a fair chance to be transmitted. Starvation freedom primarily depends upon the output port allocation policies by an arbiter. For verification of starvation-freedom, considering input port selection policies within a router for the next transmission via an output port is sufficient, instead of considering every router in an NoC. Therefore, starvation is considered as a locally dependent property in an NoC router. Since complete NoC need not be considered, state space explosion [21] is avoided for verification of starvation freedom. Model checkers like NuSMV or Spin can also be used for verification of starvation-freedom [22,23].

1.2.2 Deadlock Detection in NoC

In case of a deadlock in NoC, a group of packets are involved in a cyclic wait by holding and waiting for resources on one another. The resource dependency cycle might spread across a large number of NoC routers. Thus, deadlock might involve overall NoC and it has to be considered as globally dependent scenarios. Deadlock is considered as a globally dependent property since all routers in an NoC need to be examined at a time to detect deadlock scenarios. Therefore, detection of deadlock in an NoC is a challenging task, due to the prohibitive state space required to encode the complete NoC model and the routing algorithm therein. The use of model checker like NuSMV [22] for detection of deadlock in NoC is not feasible due to extensive state space.

NoC simulators handle deadlock in a different way. Booksim2.0 [24] is a cycle accurate NoC simulator that simulates NoC on a cycle by cycle basis. This simulator uses a threshold number of cycles to determine deadlock. If the number of cycles exceeds the threshold and yet packets remain to be delivered, simulation is aborted giving a warning message for possible deadlock. No deadlock scenario showing resource dependency is produced after a deadlock warning which would be helpful in understanding real cause of deadlock and to avoid them. Gem5 [25] is another widely used full system simulator that uses a similar philosophy for deadlock detection. It internally uses Garnet [26], an interconnection network model, for NoC simulation. It may be noted that since cyclic dependency for the resource is not checked in Booksim or in Gem5, it does not give a guarantee that the scenario, in fact, is a deadlock. Also, the threshold value may overshoot due to other circumstances like failure, livelock or starvation.

Since neither model checker is suitable for deadlock detection nor simulator gives confirmed deadlock with deadlock snapshot, there is a requirement of an alternate approach for detecting confirmed experimental deadlock which can provide the instance of the deadlock scenario as well. In experimental deadlock detection, whether deadlock is present or not in a given traffic pattern with respect to a given routing algorithm and topology is reported. Whereas, theoretical approach considers all possible scenarios while predicting deadlock [1, 2, 9]

1.2.2.1 Application Specific Deadlock Detection

A certain input traffic may or may not lead to deadlock even though the routing method is deadlock prone as per theoretical deadlock analysis. We have illustrated this with two examples.

Two traffic patterns are considered for a 2x2 Mesh NoC with buffer capacity of one in

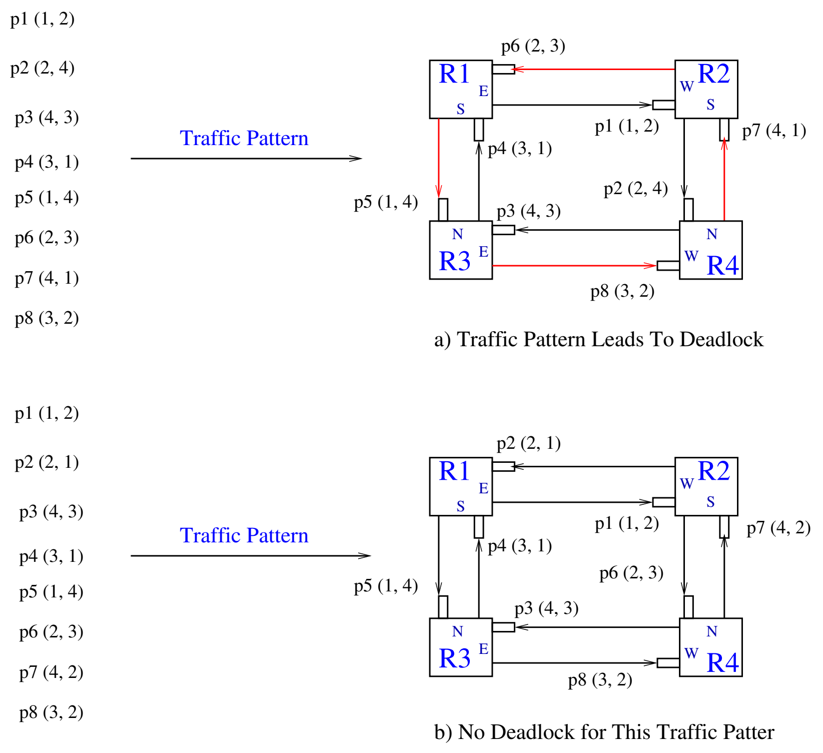


Figure 1.4: *Deadlock example: (a) Traffic patterns leads to deadlock, (b) No deadlock in another traffic pattern.*

Fig. 1.4. Deadlock appears for one traffic pattern as shown in Fig. 1.4(a) but no deadlock is found for the second traffic pattern in Fig. 1.4(b). In Fig. 1.4(a), eight packets are considered. For each packet, the source and destination addresses are written in bracket. Here, p1(1, 2) means, the source and destination of Packet1 is router R1 and router R2, respectively. Let us assume, p1(1, 2), p2(2, 4), p3(4, 3) and p4(3, 1) are transmitted from Router R1, R2, R4 and R3 at a same time. These packets have just occupied the buffer of the next router before getting delivered at its destination, as shown with black arrow in Fig. 1.4(a). We assume, at this moment p5(1, 4), p6(2, 3), p7(4, 1) and p8(3, 2) are transmitted towards their destination from routers R1, R2, R4 and R3, respectively. They follow the path indicated by the red arrow, as the buffer in the path shown by the black arrow are fully occupied. The current buffer position of p5, p6, p7 and p8 are also shown with red arrow in Fig. 1.4(a). As there is no resource dependency for packet p1, p2, p3 and p4, eventually they would be delivered at the destination core. Whereas, the scenario is different for packet p5, p6, p7 and p8. Here, p5 is waiting for the buffer being held by p8,

p8 is waiting for the buffer being held by p7, p7 is waiting for the buffer being held by p6 and p6 is waiting for the buffer being held by p5. Thus it creates a cyclic dependency on holding and waiting for resources. Thus, this traffic pattern in Fig. 1.4(a) leads to deadlock. Another sequence of eight packets are shown in Fig. 1.4(b). Current positions of the packets in the buffer of adjacent routers are shown. In this case all packets would gets delivered eventually. There is no dependency cycle formation.

Reporting a confirmed deadlock with a deadlock scenario on a given traffic pattern with respect to a given routing algorithm and topology is useful. It has also importance in context of deadlock avoidance or deadlock recovery. In this work, we, therefore, target application specific deadlock detection.

1.2.3 Deadlock Representation and its Avoidance

An informative way of deadlock representation helps in formulating deadlock avoidance. Channel Dependency Graph (CDG) [2] and Turn model [1, 9] are two classical theoretical approaches used for deadlock detection and its representation.

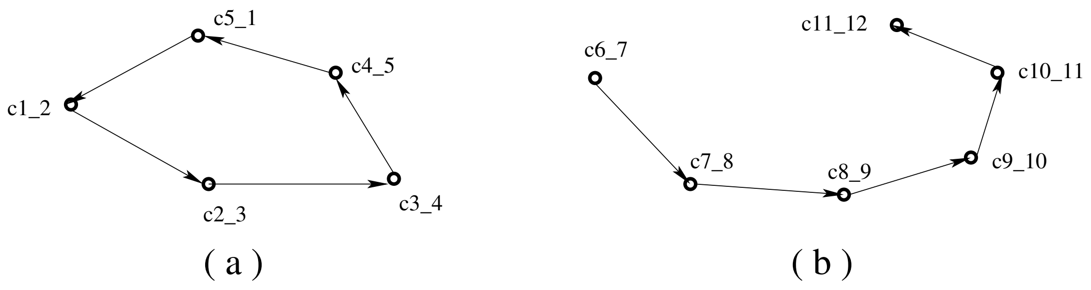


Figure 1.5: *Channel Dependency Graph: (a) Deadlock cycle, (b) No deadlock*

1.2.3.1 Channel Dependency Graph

Deadlock occurs in an NoC due to channel and buffer dependency across a path. The CDG is used for presenting channel-buffer dependency across a path in NoC [2]. Each router in a CDG represents a combination of channel with its associated buffer. Dally's theory states that a routing algorithm is deadlock free if there is no cycle in any CDG and vice versa [2]. Fig. 1.5(a) shows a deadlock cycle represented using CDG. All vertices in CDG represent a channel-buffer combination. Here, the vertex ci_j represents the channel from router i to router j associating with connected buffer in router j. Another CDG representing buffer dependency without forming cycle is shown in Fig. 1.5(b).

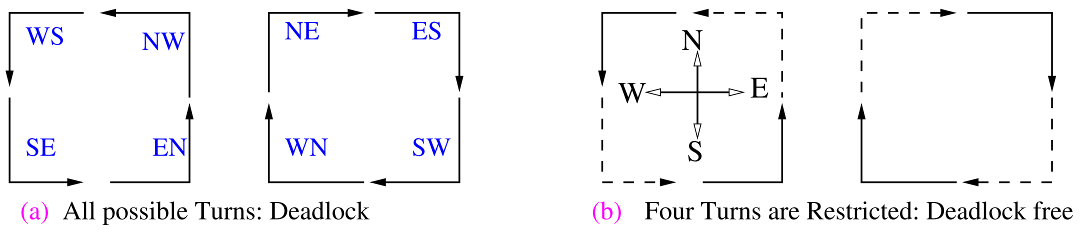

WS
NW
SE
EN
NE
ES
WN
SW
(a) All possible Turns: Deadlock
N
W
E
S
(b) Four Turns are Restricted: Deadlock free

Figure 1.6: *Turn model: (a) All possible Turns create deadlock, (b) XY-Turns (solid lines) and YX-Turns (dotted lines) are deadlock free*

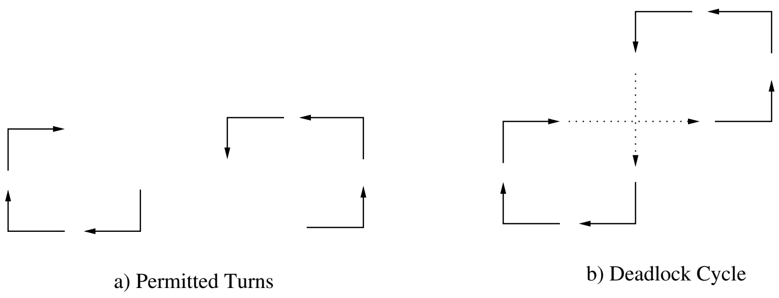


Figure 1.7: *Deadlock cycle from Turn model: (a) Permitted Turns by a routing algorithm, (b) Deadlock cycle*

1.2.3.2 Turn Model

Turn model is an approach to avoid deadlock in NoC by restricting certain Turns by packets [1]. Turn represents the change in direction by a packet. There are four clock wise Turns and four anti-clock wise Turns as shown in Fig. 1.6(a). A routing is deadlock prone if permitted Turns in that routing algorithm complete either a clockwise or anti-clockwise cycle. If a packet is moving towards X-direction, i.e., in East or West direction, and changes its direction towards Y-direction, i.e., towards North or South direction, the Turn involved is called XY-Turns. Similarly, a packet is moving in Y-direction and changes its direction towards X-direction is called YX-Turns. The solid Turns in Fig. 1.6(b) are XY-Turns and the dotted Turns are YX-Turns. XY-Turns and YX-Turns are individually deadlock free because they can not complete a cycle as shown in Fig. 1.6(b). An algorithm is deadlock prone if it is possible to form a direct or indirect cycle by using all permitted Turns in a routing algorithm. A set of permitted Turns and a possible deadlock representation using these Turns are shown in Fig. 1.7(a) and Fig. 1.7(b), respectively. It is possible to avoid deadlock by restricting certain Turn involved in the formation of a deadlock cycle. Therefore, representation of a CDG cycle along with Turn information will help in deadlock avoidance. However, Turn information is not used in CDG.

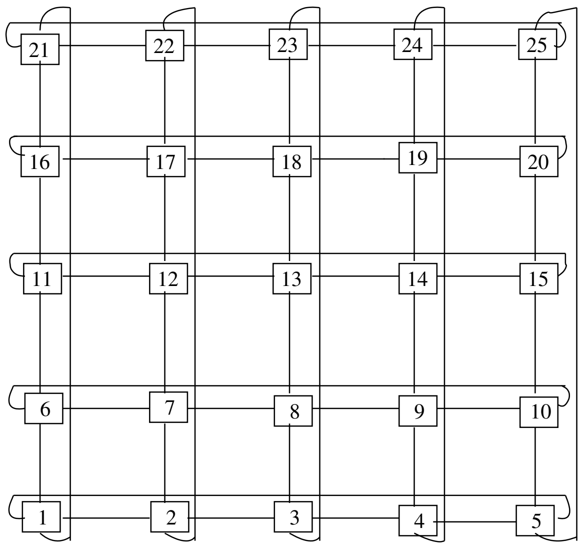


Figure 1.8: *A* 5x5 *Torus NoC composed of ring networks*

1.2.3.3 Deadlock in Torus NoC and its Avoidance

A Torus topology is composed of a set of ring networks. A 5x5 Torus NoC is shown in Fig. 1.8. Inherent cyclic path due to ring network in each row and column of a Torus NoC makes the topology deadlock prone. Though, the Torus NoC is a symmetric structure and looks like a both ended closed pipe, it is convenient to visualise Torus as a combination of wraparound channels and Mesh sub-network. Wraparound channels are the channels that connect two routers in opposite boundary of a Mesh NoC. Literally, all channels in a Torus NoC are symmetric and of same characteristics. For convenience in formulating deadlock avoidance in a routing algorithm, a set of channels are marked as wraparound channels. Using Up*/Down* [10] and FirstHop [1] routing approaches, resource dependency through wraparound channels are discontinued. In Up*/Down* routing, a spanning tree is formed to obtain an acyclic routing path corresponding to the given Torus NoC. In FirstHop routing, a packet is allowed to use the wraparound channel only at its first hop so that cyclic path is discontinued. The leverage of wraparound channels are not utilised for packets that do not originate from a boundary router [1]. Thought Virtual Channels (VC) [11, 13, 27] or additional buffer [28–30] are used for avoiding deadlock in Torus, there are scopes for new research if deadlock in Torus NoC can be avoided without using any additional buffer or VC and at the same time increase the utility of wraparound channels.

1.2.4 Formal Modeling of NoC

In this subsection we have discussed briefly about a few relevant works on formal modeling and verification of NoC.

In [31, 32], the communication infrastructure verification of NoC is targeted. A formal model for NoC is developed and implemented in ACL2 theorem prover. Reachability of packets between routers are verified in these works where detailed modeling of individual NoC components are not considered [31], [32]. Progress verification of a communication network is performed using xMAS based model in [33]. Since verification of global progress for huge system is not scalable due to state space explosion, global progress is broken down into localized progress property. Verifying local progress between components leads to verification of overall progress. A credit based flow control is designed to control number of packets that enter system and to maintain synchronization. A formal model of the Hermes NoC router architecture with its communication scheme is presented in [34]. Heterogeneous Protocol Automata (HPA) is proposed as a language to model Hermes NoC. Reachability of packets are verified in that work using Spin model checker [23]. Hermes NoC router is a five bi-directional port router with a buffer at each input port [35]. Verification of bidirectional NoC is performed in [36]. That work uses State Graph Manipulator (SGM) for model checking. Starvation freedom and mutual exclusion are verified in that work. In [37], process-algebra is used for formal verification of fault tolerant NoC. Scalability is not achieved in that work as only 2x2 NoC model is considered for experiments.

Due to the complexity and extensive number of NoC functional units, detailed modeling of NoCs are not performed in most of the existing works. Scalability of NoC or lack of internal details of a router are main issues in most of the existing NoC verification related works. The detailed modeling of NoC is important so that the verification results remain valid in real NoC as well. Therefore, we target a detailed modeling of NoC considering all functional units in this thesis.

1.3 Thesis Objectives

In most of the existing NoC verification works, the detailed modeling of the complete system is missing [31–34,36,38,39]. Without the detailed modeling of NoC, the formal analysis would not be complete. Moreover, the abstract model does not represent the actual implementation of the NoC. As discussed already, there are two primary challenges in NoC verification - modeling of NoC considering functional units in detail and achieving scalability of the verification method using the detailed model. Though it is not feasible to verify the complete NoC at a time due to state space explosion problem, it is possible to verify locally dependent properties like starvation-freedom using model checker. *Therefore, a detailed modeling of NoC using Finite State Machine (FSM) for verification of locally dependent properties like starvation-freedom is demonstrated in this thesis.* There are some specifications like deadlock-freedom that require global information of overall system for its verification. Model checker is not capable of handling the huge state space of the complete NoC. An application specific simulation framework that works on a formal model seems promising in that regard. *Therefore, it motivates us to do a formal modeling of NoC using a formalism named as Communicating Finite State Machine (CFSM) [40]. Due to inherent synchronization in CFSM, it is convenient to automate simulation and construct a simulation framework using CFSM based model. In case of confirmed deadlock detection, it is convenient to obtain resource dependency scenario from the CFSM based framework. The deadlock scenario with resource dependency helps us in formulating deadlock avoidance. It motivates us to generate experimental deadlock scenario while detecting confirmed deadlock* *using our CFSM based simulation framework [41].*

Representation of deadlock scenarios in an informative way helps in formulating deadlock avoidance. *Since Turn information is not used in CDG, an approach of combining both Turn information and CDG for deadlock representation would be more informative and helpful for avoiding deadlock. Therefore, it motivates us to propose a deadlock representation approach by incorporating Turn information along with CDG.* Deadlock primarily depends upon the underlying routing policy in a given topology. If there are many inherent cyclic paths in a given topology that topology structure may also be the culprit in inducing deadlock in NoC. Torus NoC consists of many connected rings as shown in Fig. 1.8. Due to the presence of inherent cyclic paths via ring networks, Torus NoC is more vulnerable to deadlock as compared to Mesh NoC. Avoiding deadlock for Torus NoC is possible using VC and additional buffers [12, 28]. Whereas, there is a scope for formulating deadlock avoidance for Torus without using VC or additional buffer by applying restricted routing. There are not many deadlock free routing algorithms in Torus without using VCs or additional buffers. *Inspired from Turn model, our another objective is proposing an deadlock avoidance approach applying restriction on packet movement in Torus NoC.*

From the above motivations, the following four objectives are identified:

- Formal modeling of NoC components in detail level using FSM. Verification of correctness of the model and locally dependent properties like starvation using model checker within a manageable state space.

- Formal modeling of NoC components in detail level using CFSM. Automate the process of NoC model creation by taking NoC grid size as input for avoiding human error in model creation. Develop a formal model based simulation framework using the CFSM based NoC model to detect globally dependent deadlock with respect to a given traffic pattern and a given routing algorithm. Report the deadlock scenario with the detailed resource dependency in case of a confirmed deadlock.

- Propose an approach for representing deadlock in a more informative way named as Direction Dependency Graph (DDG). The DDG has to obey basic characteristics of Channel Dependency Graph (CDG). The Turn information is also needed to be incorporated in DDG to obtain direction information. Formulate a deadlock avoidance approach named as Arc model for Torus NoC.

- Demonstrate deadlock free routing algorithms using the proposed Arc model for Torus NoC. Use the proposed DDG for theoretical deadlock analysis while using the Arc model.

1.4 Contributions of the Thesis

The contribution of the thesis to meet the objectives identified are briefly described in this section. A high-level overview of our contributions is shown in Fig. 1.9, which is elaborated in the sub-sequence sections.

1.4.1 Formal Modeling of NoC using FSM and Verification of Starvation using Model Checker

In our first work, we have used Finite State Machine (FSM) for modeling NoC in detail level. FSM is chosen for modeling because it is convenient to implement an FSM based model in a model checker like NuSMV [22]. The NoC components like buffer, switch, arbiter are modeled using FSM. Synchronization between these components are maintained using dedicated FSMs. Two arbitration policies namely Fixed-priority arbiter and Round-robin arbiter are also modeled. Model of all these NoC functional units are encoded using NuSMV model checker. For verifying a property, its specification is represented using linear temporal logic (LTL) [42, 43] and given as input to the model checker. NuSMV internally composes all the FSMs and checks for all possible scenarios [43]. The model checker reports whether given LTL property is satisfied or not. For ensuring correctness of the FSM based model, the synchronization within a router, progress between router components, transfer of

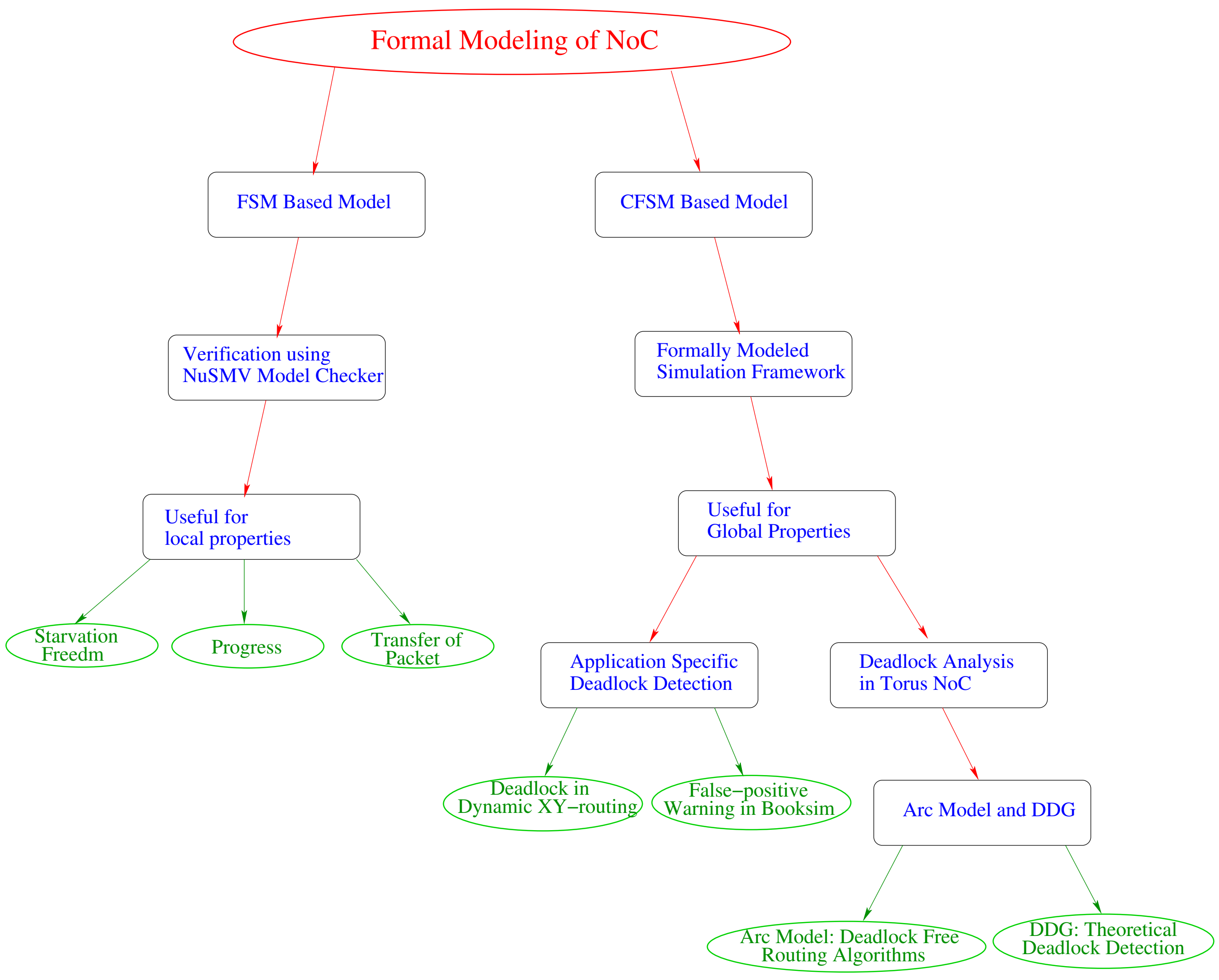

Formal Modeling of NoC
FSM Based Model
CFSM Based Model
Verification using NuSMV Model Checker
Formally Modeled Simulation Framework
Useful for local properties
Useful for Global Properties
Starvation Freedm
Progress
Transfer of Packet
Application Specific Deadlock Detection
Deadlock Analysis in Torus NoC
Deadlock in Dynamic XY−routing
False−positive Warning in Booksim
Arc Model and DDG
Arc Model: Deadlock Free Routing Algorithms
DDG: Theoretical Deadlock Detection

Figure 1.9: *A high-level overview of the contributions from the thesis*

packets between routers, correctness in setting priority for Round-robin arbiter are verified. Progress property in a model implies that current state of an FSM does not stuck in a state, i.e., state of an FSM keeps on changing provided that there is a packet present as input for transmission.

As an application of the model, starvation-freedom for Fixed-priority arbiter and Round-robin arbiter are verified. Since starvation-freedom primarily depends upon resource allocation logic, it is possible to verify starvation without considering the complete NoC at a time. We partition the NoC into active regions and perform verification on each active region in parallel to improve the overall verification time. Experimental results show that starvation freedom is satisfied for all input ports when Round-robin arbiter is used. For Fixed-priority arbiter, we have considered the priority order for input ports as Local $>$ East $>$ West $>$

North > South. Though it seems the starvation-freedom should be satisfied only for the highest priority port, experimental results show that starvation-freedom is satisfied for both first and second highest priority ports. This happens due to the synchronization latency after transferring a packet in the model.

1.4.2 Formal Modeling of NoC using CFSM and Developing a Simulation Framework for Deadlock Detection

Our second contribution is to confirm whether deadlock is present or not on a given input traffic pattern with respect to a given routing algorithm and given topology. We have developed a simulation framework using a formally modeled NoC with help of Communicating Finite State Machine (CFSM) [41]. When a deadlock is confirmed, the deadlock scenario representing the cyclic resource dependency is also reported by the simulation framework.

We model NoC components using CFSM in this work. Transitions in all CFSMs are carried out with help of sending and receiving of unique messages [40]. The synchronization between interacting functional units or CFSMs are inherently controlled by this process of sending and receiving of unique messages. Therefore, no dedicated CFSM unit is required to maintain synchronization between functional units. This is one advantage of using CFSM over FSM for modeling a system like NoC where a large number of units continue functioning in parallel. Therefore, it is convenient to develop a simulation framework for NoC using CFSM based model.

When a system is halted due to occurrence of deadlock, no transitions are possible in the equivalent CFSM design of the complete system. If all packets have not reached their destinations and no further transition is occurring in the CFSM model, it is a clear indication of deadlock. Due to these behaviours, the CFSM is a suitable modeling formalism for detection of deadlock in a complex system like NoC. In case of occurrence of deadlock, using the current global state of CFSMs along with associated packets, an exact deadlock scenario depicting the resource dependency is obtained from the simulation framework.

We have detected deadlock in dynamic XY-routing [8] using the CFSM based simulation framework. Deadlock is detected for XY-routing in Torus NoC as well. Thus, experimental findings reinforce that besides NoC routing algorithm deadlock is dependent on NoC topology as well. The deadlock scenarios obtained from experimental resource dependency help us to carry forward our next research on formulating deadlock avoidance and developing deadlock free routing algorithm for Torus NoC.

1.4.3 Deadlock Avoidance in Torus NoC using Arc Model and DDG

The third work of this thesis is on deadlock avoidance in Torus NoC. For representing deadlock with additional information like Turn and directional information, Directional Dependency Graph (DDG) is proposed in this work. An Arc model is also proposed to avoid deadlock in Torus NoC.

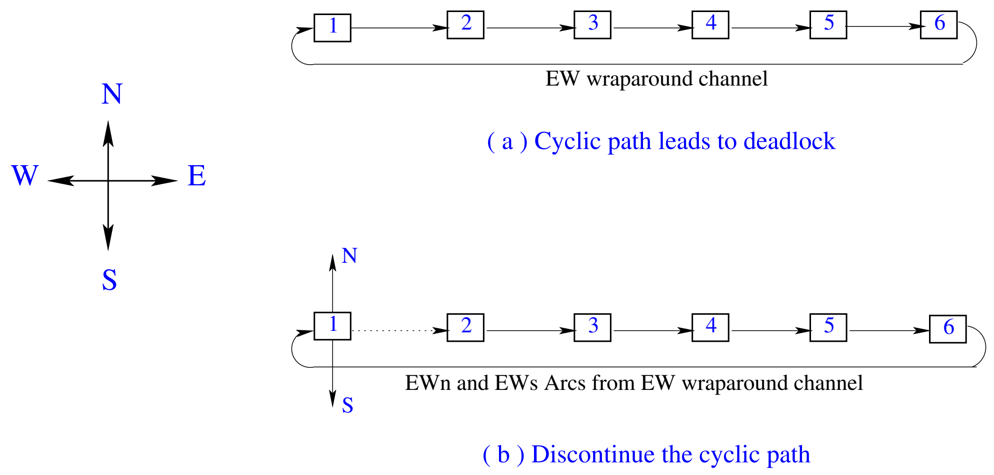

1
2
3
4
5
6
EW wraparound channel
(a) Cyclic path leads to deadlock
N
W
E
S
N
1
2
3
4
5
6
S
EWn and EWs Arcs from EW wraparound channel
(b) Discontinue the cyclic path

Figure 1.10: *Avoid deadlock by discontinuing the cyclic path*

The proposed DDG has taken advantage from both the Turn model and Channel Dependency Graph. DDG is useful for predicting deadlock theoretically by creating possible deadlock cycle. These deadlock cycles are helpful in formulating deadlock avoidance as well. A cyclic path in Torus NoC is shown in Fig. 1.10(a). Torus NoC consists of a number of ring networks that contribute to many such cyclic paths in Torus NoC. Though all channels are symmetric in Torus NoC, let the channel connecting router R6 and R1 be considered as wraparound channel in Fig. 1.10(a). Since this channel is from the East direction to the

West direction, we termed this as EW wraparound channel. One approach of discontinuing this cyclic path via wraparound channel is to restrict the immediate backward movement just after traversing through EW wraparound channel. After taking EW wraparound channel, it has to move towards the North (EWn) or South (EWs) direction for at least one hop. Thus, the EW wraparound channel is subdivided into EWn and EWs Arcs in Fig. 1.10(b). In this approach, there are four types of wraparound channels (EW, WE, NS, SN) and each wraparound channels are subdivided to obtain eight Arcs namely EWn, EWs, WEn, WEs, NSe, NSw, SNe and SNw.

We consider Torus NoC as a combination Mesh sub-network and wraparound channels for convenience. The Arc model is helpful in avoiding the immediate deadlock cycle. Nevertheless, behaviour of Arc model with respect to the set of permitted Turns in Mesh sub-network need to be analysed. We have analysed deadlock behaviour of Arc model with respect to XY-Turns in the Mesh sub-network using DDG. The analysis shows that all eight Arcs are individually deadlock free with respect to XY-Turns. We also performed deadlock analysis for all possible Arc pairs. From 8 Arcs, total $\binom{8}{2} = 28$ Arc pairs are possible. Using DDG, we have identified 14 Arc pairs as deadlock prone and 14 Arc pairs as deadlock free with respect to XY-routing in the Mesh sub-network. The experimental results from the CFSM based framework also support the findings from DDG analysis.

The Arc model helps in avoiding deadlock in Torus NoC where additional buffer or VC are not needed. In our next work, we have demonstrated deadlock free routing algorithms for Torus NoC using deadlock free Arc pairs with respect to XY-Turns.

1.4.4 Deadlock Free Routing Algorithms for Torus NoC using Arc Model

The fourth work of this thesis is the application of Arc model for designing deadlock free routing for Torus NoC. With the help of DDG analysis and CFSM based framework, we have already identified 14 deadlock free Arc pairs with respect to XY-Turns. Therefore, 14 deadlock free routing algorithms are possible using different Arc pairs with XY-Turns. We have presented one of them in this work. FirstHop [1] and Up*/Down* [10] routing are two other approaches for deadlock avoidance in Torus that do not use additional buffer or VC as like Arc model. We compare Arc based algorithm with FirstHop and Up*/Down* methods. The wraparound channels are helpful in saving overall hop count in a given traffic. Experimentally it is found that algorithm with two Arcs save more hop counts in comparison to Up*/Down* routing. In FirstHop routing [1], since a packet is allowed to use the wraparound channel only at its first hop, saving in hop count depends upon the percentage of traffic originated from boundary router. There are more percentage of boundary routers in a smaller Torus NoC. The percentage of boundary routers decreases with the increase of NoC grid size. Therefore, saving in hop count also decreases significantly with increase of NoC size in case of FirstHop algorithm. In our Arc based algorithm, saving in hop counts are not affected drastically by the changes in NoC size. The saving in hop counts by algorithms with two Arcs are closer to FirstHop algorithm. We have enhanced our Arc based algorithm by using three Arcs with XY-Turns. The saving in hop count by algorithm with three Arc is better than that of FirstHop algorithm for NoC with grid size 9x9 onward. To improve further, we have used three Arc based algorithm with FirstHop approach by applying it for one wraparound channel. The resultant algorithm further improves the saving in hop counts. We have used DDG to prove deadlock freedom for the proposed Arc based algorithms. No deadlock is detected as well for Arc based algorithms by the CFSM based framework while different input traffic patterns are applied.

1.5 Organization of the Thesis

The thesis is organized as follows:

Chapter 1 Introduction, motivation and contribution of the thesis are presented in this chapter.

Chapter 2 Detailed literature survey on formal modeling and verification of NoC, NoC routing algorithms, existing works on deadlock detection and avoidance approaches are presented in this chapter. We have identified research gaps in those areas and stated our objectives to bridge them.

Chapter 3 Formal modeling of NoC using FSM, verifying correctness of the NoC model and application of the FSM model in verifying starvation-freedom are presented in this chapter.

Chapter 4 Detailed modeling of NoC components using CFSM and automated NoC model generation are presented in this chapter. Detection of application specific confirmed deadlock with deadlock scenario using a CFSM based NoC model is also presented.

Chapter 5 Deadlock avoidance using Arc model for Torus NoC is presented in this chapter. DDG is also presented for representing and analysing deadlock with Turn and direction information.

Chapter 6 Application of Arc model in designing deadlock free routing algorithm for Torus NoC is presented in this chapter.

Chapter 7 The thesis concludes in this chapter indicating the scope for future direction of our research.

2 Background and Literature Survey

In this chapter we have discussed the existing works related to our research. At first, we discuss about formal verification techniques. For applying formal methods, formal modeling of the system is the prerequisite. An overview of existing works on formal modeling and verification of NoC is then presented in this chapter. Application of the existing formal models in detection of starvation, progress, deadlock along with the limitation of those approaches are also presented. We have discussed the challenges and research gap in formal modeling and verification of NoCs based on the existing works and then stated our objective. Next, we have briefly described about different routing algorithms for NoC. Deadlock freedom is an important aspect of NoC routing algorithms. The state-of-the-art works on deadlock avoidance approaches for NoC routing algorithms are presented. Research gaps for deadlock avoidance in Torus NoC are also identified. To bridge the research gap, we have identified our objectives in this chapter.

2.1 Formal Verification Techniques

Simulation and testing are widely used for verifying the correct functionality of System-on-Chip (SoC) and NoC. Formal methods have potential to offer early integration of verification in the design phase of an SoC and NoC [43]. The growing complexity and the pressure to reduce development cycle (time-to-market) make the delivery of no-defect system is an enormously challenging activity. Preventing bugs in hardware design has more importance. Because, hardware is subject to high fabrication cost and fixing them after delivery to customer is very costly and sometimes is not feasible. Whereas, software bugs can be handled by providing users with new updates or patches. Manufacturing of bug free products are essential to the growth and even survival of a company. In the early nineties, due to the floating point division unit bug in Intel's Pentium II, the company has to replace processors. This bug cost a loss of approximately 475 million US dollars and damaged Intel's reputation. Such errors when occurs in safety-critical control system such as chemical plants, nuclear power plants, automated traffic controller can have catastrophic consequences as well. The Ariane-5 space launch vehicle by European Space Agency (ESA) was crashed in 36 seconds on June 4, 1996. For the software flow in the controller of a radiation therapy machine named Therac-25 causes overdose of radiation. As a result six cancer patients were died during 1985 and 1987. Investigations on these incidence have shown that formal verification procedures would have detect the error in Ariane-5 space launch vehicle, Intel's Pentium II processor and Therac-25 radiation machine beforehand.

2.1.1 Model Checking

Model checking is a formal verification technique that explores all possible system states. Model checking approach is an automated technique where a finite-state system model and a set of formal properties are given as input. Model checker systematically checks whether these properties hold for that model. This technique is based on models which describe the behaviour of the system in a mathematically accurate design. The accurate modeling of a system discovers any incompleteness or inconsistency in the system. Such problems are usually discover at much later stage. Verification using a model is as accurate as the accuracy in designing the system model [43]. The major challenge in model checking is to examine the largest possible state space. State space of a system model increases exponentially with the incorporation of every new functional units.

Besides the model of the system under consideration, formal representation of the property to be checked are the prerequisite input for model checking. Characteristic of a transition system with respect to time is represented using temporal logic. Sequence of state in a transition system is called computation path. The expected change of state with respect to time and current state in computation paths are represented using Linear Temporal Logic (LTL). We have used LTL to represent properties in a given model in this work. These LTL specifications are given as input to the model checker. Model checker determines whether a given LTL specification is satisfied or not with a True or False value. If an LTL does not hold True a debug trace representing how the give LTL is not satisfied is given as output.

SMV (Symbolic Model Verifier) is a model checking tools developed by K. L. McMillan under the guidance of E. M. Clarke at Carnegie-Mellon University (Pittsburgh, PA, USA) [44]. It performs model checking of a system under consideration using BDD (Binary Decision Diagram). NuSMV [22] is the extended and reimplemented version of SMV. NuSMV symbolic model checker developed as a joint project between FBK (Future Built on Knowledge) research institute in Italy, Carnegie Mellon University (CMU), University of Genova and University of Trento. Version 2 of NuSMV [45] was developed in 2002 inheriting all functionalities of previous version with the integration of model checking techniques based on propositional satisfiability (SAT). The aim of NuSMV open source project is to provide a common research platform to the model checking community.

2.1.2 Equivalent Checking

Equivalence checking is another technique used for formal verification. This technique is used to prove that two representations of the same system using formal model exhibit the same behavior. Model checking is a functional verification process and is different from equivalence checking. Exhaustive simulation are used in functional verification to verify the correctness of a model. In case of equivalent checking, at first, a system model has to be verified. Let an alternate representation of the same system is available. Equivalence checking can be used for determining if the second alternative model exhibits same behaviour as that of the verified version. For example, equivalent checking is used in industry for determining whether functionality of a model has changed after applying synthesis operation. Finite state machines with data path (FSMDs) are mostly used in equivalent checking for modeling the system specification [46].

2.1.3 Theorem Prover

Theorem prover accepts a set of axioms and a theorem as input and returns as output a formal mathematical proof for the theorem [47]. Instead of proving new theorem, theorem prover formalises the outline of a proof. It provides mathematical reasoning on the correctness of system properties. Theorem proving reasons about the state space using system constraints only. All states on state space are not used as like a model checker. A theorem provers are used for software and hardware verifications. It is also used in assisting human mathematicians in conducting complex proofs. Usually a time limit is set for a theorem prover on a given theorem for its proof. If time limit expires and neither proof is complete nor a counter example is found, the prover terminates in such cases [48].

2.2 Formal Modeling and Verification of NoC

In this section, we discuss works related to formal modeling of NoC targeting different properties, formal model based and run-time deadlock detection and traffic modeling for performance analysis of NoC.

2.2.1 Formal Modeling using Different Formalism

Due to the enormous benefits of the formal models [43], there have been efforts on formal verification of NoC and deadlock detection using both model checker and theorem prover. D. Borrione et al. target validation of the communication infrastructure of NoC [31]. Generic NoC is based on an abstract view of the communications network of NoC is presented in this work. This work has node considered router components in detail. The generic NoC model is modeled as the composition of key components like routing, scheduling and interfaces. This model includes topologies, routing algorithms and scheduling policies. A finite number of routers are connected to generic NoC communication architecture. A generic formal model for NoC is implemented using ACL2 theorem prover and simulator. ACL2 provides two functionalities, as a theorem prover and as an execution engine in the same modeling environment. Timing information is not considered in that work, which is incorporated in their next work [32]. This work describes an extension work on generic model for NoC [31]. It is implemented in ACL2 theorem prover, which contains the executable logic as well. This model is useful for serving as a formal reference for the validation and simulation of NoC at the initial design phase. The work demonstrated the transmission of messages on generic communication architecture, with an arbitrary network topology and node interfaces, routing algorithm and switching technique. The model considers its main input, as a list of messages that can be injected in the network. Before injecting in the network, these messages are first encoded. Properties like deadlocks, starvation are not considered in this work and it considers only high-level description of NoC while modeling.

A polynomial-time algorithm using ACL2 interactive theorem prover for deadlock freedom in NoC is proposed in [49]. They identified the sufficient condition for occurrence of deadlock in wormhole networks. However, checking a necessary and sufficient condition is co-NP-complete [49]. Therefore, this approach cannot detect all possible deadlocks. In their next work [50], the author present proof for deadlock, livelock, starvation and functional correctness. They consider three main functionalities viz. an injection method for injecting message into the NoC, a routing function and a decision making function if a message can advance to next node. It considers high level of abstraction for the NoC. More refinement is needed to ensure that the properties established at the abstract level also hold on implementation of different NoC architectures. One main contributions of this work [50] are designing a new generic NoC model with capability of adaptive routing algorithms. It has added two new global theorems, namely evacuation and starvation freedom. The generic NoC approach has three main functionalities. There is an injection method which chooses messages amongst pending messages that can go into the network and can access network resources. The routing function determines the all possible next hops from the current position and the destination of hop for a message under consideration. The switching policy takes decision if a message can advance to a node. Generic NoC combines these three functionalities to form a network simulator. A network in generic NoC contains a set of resources R. Each resource R has a certain number of buffers. There are two special resources, source and sink. Source generates message and sink is used to consume messages.This work presents proof for deadlock, live lock, functional correctness, and starvation freedom in ACL2. But it considers only high-level descriptions of NoC. However, more refinement is needed to ensure that properties established on the abstract model keeps hold on the actual hardware implementation.

V. A. Palaniveloo et al. present a new formal model of the existing Hermes NoC router architecture along with its communication scheme in [34]. They propose Heterogeneous Protocol Automata (HPA) as a language to model Hermes NoC as an event based transition system. This work verifies reachability of flits using Spin model checker [23]. At input port of the Hermes Router [35] it models five bi-directional ports and the bounded buffers. It also includes the XY-routing algorithm, wormhole switching, arbitration logic with priority, and a handshake protocol of the communication scheme. With help of PROMELA, the automata model is mapped manually. PROMELA is specification language for the SPIN model checker [51]. It models XY-routing algorithm by writing codes in PROMELA. This work models NoC components like switch, arbitration by defining Heterogeneous Protocol Automata. It verifies reachability of a flit using SPIN model checker. This work has not checked critical issues like starvation, deadlock, livelock etc.

Y. C. Lan et al. [52] proposes a bidirectional channel Network-on-Chip architecture to enhance the performance of on-chip communication. It allows each communication channel between two routers to be dynamically self configured to transmit flits in either direction. It makes better utilization of on chip hardware resources. This bidirectional channel promises better bandwidth utilization, lower packet delivery latency, and higher packet consumption rate at each on-chip router. To implement this bidirectional traffic, this work proposed a control algorithm between two neighbouring routers. It allows neighboring routers to coordinate in setting the specific directions for the pair of channels between them for transmitting and receiving packet. A novel router architecture which supports dynamic self re-configurable bidirectional channels is also proposed. This paper claims that the hardware overhead is negligible for bidirectional channels. Y. R. Chen et al. present a verification approach on bidirectional NoC in [36]. They have verified mutual exclusion and starvation freedom. Formal Modeling of NoC presented in this work considering the proposed Bidirectional NoC design [52]. A channel has three states: free, idle and waits. One end of a channel is configured as free state and data are transmitted from that end. Receiving end of a channel is in idle state. To change a state from idle to free, it goes through the intermediate state called wait state. BiNoC design is converted into Extended Time Automata for verification. State Graph Manipulator (SGM) [53] is used as model checking tool in all the experiments. Due to state space explosion, the work can not verify deadlock freedom. The model used in this work considers only one router and has not considered detailed model of the NoC components. The approach may not scale for a complete NoC. In a Ph.D. dissertation on formal verification of fault tolerant NoC [37], process-algebra is used for modeling. For all the experiments in this work a 2x2 NoC model is considered. Scalability is not achieved in this work. Deadlock verification fails due to state space explosion problem.

2.2.2 Formal Modeling using xMAS Primitives

There is another approach of modeling interconnection network using executable micro architectural specification (xMAS) [54, 55]. A richer set of micro architectural primitives are identified that allow describing complete system by composition methods. The xMAS primitives help to build models faster as models are now simply wiring diagrams at some level of abstraction. It is claimed that xMAS based models are closed to hardware implementation. Models designed using xMAS can be used for model checking as well as for dynamic validation and for performance modeling. Some of the basic xMAS primitives are queue, function, source, sink, fork, join, switch and merge. Source can generate packet infinitely and a sink can absorb them without delay. By applying a function in a channel, data can be modified. The xMAS primitives are used as basic design units to describe complete system by composition. They are useful as a modeling framework for validating existing micro architecture. These are recent works that usages xMAS as modeling framework in their works [33, 56–62].

Some existing verification works on communication network [33], [56], [63] use xMAS for modeling the system. S. Ray et al. present progress verification on a communication fabric by breaking end-to-end progress property of a virtual channel into localized progress property [33]. This paper considers a virtual channel which is designed using xMAS [33]. This work focuses on progress property. Progress means whenever there is a packet trying to enter the virtual channel, it will pass through the channel. In this work [33], end-to-end progress property is broken down into localized progress properties. Localized progress are more easily provable, and leads to a formal proof of overall progress. ABC verification engine is used for verification [64]. The authors conclude that some more study is needed to apply the same approach for progress verification in NoC. D. E. Holcomb et al. [56] present compositional performance verification of NoC. The overall latency in a routing path is calculated using xMAS model. The overall latency bound problem is divided into a number of smaller sub-problems, termed latency lemmas. Firstly, the network is modeled using xMAS primitives. Next, the network model is converted into stage graph. Each stage is associated with a worst case latency calculated from latency lemma. By considering latency in every stage, overall latency of a packet is calculated. This approach may fail in cyclic network. Worst case latency are only be considered in all stages which is too extreme. Our focus is not on performance verification on NoC. In [63], formal modeling and deadlock verification of NoC are demonstrated without considering any specific routing algorithm. The work scales up to 8x8 grid. Intuition behind this work is to find inactive channel in an architecture, that leads to deadlock. In [65], we verify progress property on a xMAS based NoC model using NuSMV [22]. Buffer is not considered in that work. Considering buffer, verification of this model using NuSMV does not scale even upto 2x2 NoC. Using this approach, deadlock verification of NoC is found to be infeasible due to state space exposition problem.

2.2.3 Modeling using FSM and CFSM

Finite State Machine (FSM) is a popular modeling formalism suitable to use with model checker. NuSMV internally composes all the FSMs present in the system model and verifies the truth value for any given specification with respect to that composed model [43]. In verification process, state space increases exponentially in case of a huge system like NoC. This is called state space explosion problem [21]. Besides formal verification purpose, FSM are being used for various other applications by the research community [66–73]. In this thesis, we have modeled the NoC using FSM first to check locally dependent properties like starvation. We have modeled NoC considering detailed components. Synchronization between NoC components is essential for proper functioning. For maintaining synchronization between NoC components we have used dedicated FSM in our FSM based NoC model.

Communicating finite state machine (CFSM) is another modeling formalism suitable for modeling a system with number of functional units executing synchronously in parallel [40]. The synchronization between the units in a system are maintained with help of sending and receiving messages between the functional units. Message communication between functional units take place with help of blocking message queues. There are existing works on verification of communication protocol and deadlock detection using CFSM. A communication protocol validation approach using CFSM model consisting of two machines are presented in [74–76]. Fair reachability [74, 75] is applicable to CFSM consisting of two machines. Gouda et al. proposed an deadlock detection algorithm on a system consisting of two CFSMs only [76]. A theorem for checking bounded communication between two CFSMs is presented in [77]. These works [74–76], have not demonstrated systems where more than two CFSMs are present.

In general, the deadlock is identified by detecting cyclic dependencies of resources in NoC. Since CFSM is suitable to maintain synchronization between parallel system and gives clear indication in case of deadlock, we have chosen CFSM for modeling NoC. In this thesis, we have shown that the cyclic dependencies for resources are equivalent to the identifying a global state with no further transitions in the CFSM models. Moreover, this work is the first attempt to model a large distributed and parallel system like NoC in detail using CFSM.

2.2.4 Traffic Modeling using Queuing Approach

There are several attempts to model traffic characteristics for accurate performance analysis. Queuing-based approaches are widely used for network performance analysis and they are mostly based on Poisson distribution. However, the issues with queuing-based approaches is that, such models cannot efficiently account for many of the traffic characteristics e.g., non-stationarity, self-similarity etc, which are vital for multicore designs. To address these limitations, Bogdan et al. [78, 79] propose statistical physics inspired approach to model the traffic dynamics in multi-core systems. The model presented in [79] can investigate buffer overflow probability in NoC. In another work [80], a mathematical router model for NoC performance analysis is presented. This model is primarily based on number of FIFO buffers connected with switches. Different parameters like buffer size, packet size, packet transmission rate, packet waiting time, channel bandwidth etc. are used in the model. This model reports throughput, buffer utilization and average latency per router in an NoC. Qian et al. [81] present a learning based support vector regression (SVR) model for analyzing the latency performance in NoC. In [82], Qian et al. propose an accurate and scalable latency model for NoC performance analysis. Channel waiting time is estimated using queuing-based approach. In prior queuing-based work [83], the traffic arrival model uses Poisson approximations. It is extended to a generalized exponential traffic arrival in work [82]. The model in [82] can handle bursty traffic and dependent arrival times with general service time distributions. However, our focus is not performance analysis of NoC. It may interesting to study if our NoC model can be utilised for performance analysis as well.

2.2.5 Run-time Deadlock Detection

Some existing works target run-time deadlock detection and recovery in NoC [84, 85]. Primarily, they are all based on a predetermined time-out values. Therefore, these techniques may give false-positive results. In another run-time deadlock detection work [86], a distributed and parallel transitive closure (TC) network is tightly coupled with the original NoC. A TC computation unit is connected with each router. All TC units are connected with its neighbouring TC units and share transitive closure computation information for deadlock detection. Our work is some what different. Our aim is to work on a deadlock detection approach without relying on predetermined time-out values. Our approach is to check resource dependency explicitly with help of formal method and report the exact deadlock scenario.

2.2.6 Challenges and Objective

Formal modeling and verification of a complete NoC challenging due to the presence of large number of functional units and the necessary interaction between all these components. The state space explosion problem is a major issue in the scalability of verification of a complete NoC model [21]. Detailed modeling of complete NoCs are not performed so far in the existing works as discussed in this section. The importance for detailed modeling of NoC is to keep the verification results consistent in real hardware as well. *Therefore, our objective in this thesis is to work on detailed modeling of NoC considering individual functional unit* *and their interaction in details.*

2.3 Routing Algorithms for NoC

NoC became an alternative to the bus technology with the evolution of System-on-Chip (SoC). Among the different NoC topologies, the Mesh and Torus topology have been a popular NoC topology for constructing massively parallel processors [87, 88]. In Mesh NoC, each router is connected with four or three or two routers in the topology. There is more connectivity towards the central region and less connectivity in the boundary routers. Torus is another topology where each router is connected with four other routers. In Torus NoC, the communicating distance between boundary routers are reduced by interconnecting the boundary routers of the Mesh NoC. Routing algorithm plays a key role in the functioning of an NoC. There are numbers of alternative paths between a source and destination routers in an NoC. Routing algorithm determines the path that a packet will follow to reach to its destination from its source. Efficiency of an NoC depends on the underlying routing in an NoC. High performance, load balancing, fault tolerance, freedom from deadlock are some of the parameters for a routing algorithm. Based on the adaptive nature that the routing algorithms facilitated to packets, routing algorithms are broadly classified into three categories: deterministic routing, partially adaptive routing and fully adaptive routing [7]. In the deterministic routing algorithms, paths are determined based on a predefined rule and packets have to pass through predefined paths. The routing process is simplified in deterministic routing. One drawback is, packets cannot use alternative paths to avoid congested channels along the predefined paths. Adaptive routing are capable of changing its path based on the congestion in the network channels. In the partially adaptive routing algorithms, they use the advantages from both the deterministic and fully adaptive routing. A fully adaptive routing exercises higher path diversity and less restriction regarding path in comparison to deterministic and partially adaptive routing.

In case of deterministic routing, it offers only one path for each source and destination router pair. XY-routing is a deterministic routing where a packet is routed first along the

X-dimension in a 2D Mesh NoC [89]. After X-distance is covered, the packet is routed along the Y-dimension until the packet reaches its final destination. XY routing is a minimal routing algorithm. The path between the source and destination used by this algorithm are always the same and is one of the shortest paths. XY-routing is deadlock free in Mesh NoC. Though XY routing can not handle faulty paths, it is very simple to implement and has lower overhead than implementing an adaptive routing. Deterministic routing helps in reducing router complexity and power consumption has a direct relation to the number of routers a packet traverse, i.e., the hop counts. The procedure of implementing XY-routing for Torus NoC is given in [3]. Two alternate paths are available for both X-distance and Y-distance in Torus NoC. One path is in clockwise direction and another path in in anti-clockwise direction. The shorter path is considered from the clockwise and anti-clockwise alternatives.

In partially adaptive routing some moves are pre-determined and some moves are determined based on the network congestion. Three partially adaptive routing for Mesh NoC are presented using Turn model in [1, 9]. The core idea of Turn model is to prohibit a few Turns to break all of the cycles so that deadlock can be avoided. Three partially adaptive routing algorithms are namely West-First, North-Last and Negative-First that are presented in [1]. For implementing the same algorithms in Torus NoC as well, Glass et al. [1] propose FirstHop routing approach by imposing certain restriction. Applying the traffic distribution helps in improving performance of a routing algorithm. An improved routing algorithm on traffic distribution namely Odd-even routing with a better traffic distribution is presented for Mesh NoC in [90]. Odd-even routing is based on Turn model concept. Odd-even routing restricts some Turns based on its location so that deadlock can be avoided. The basic idea of Odd-even routing is to restrict some Turns with respect to location so that cycle formation can be prevented. For example, EN and NW Turns could be the part of same cycle. Therefore, EN and NW are not permitted in the same column of a Mesh NoC. Similarly, ES and SW Turns are also not permitted in the same column.

Adaptive routing considers the network status, typically buffer occupancy when selecting between alternative routing paths [3]. An adaptive routing that make routing decisions based only upon the local network state might results in global imbalance. Therefore, a global network information is necessary for a good adaptive routing algorithm. An adaptive routing for Mesh NoC using dynamic programming (DP) is proposed in [91]. A DP network with routing mechanism and routing table updating strategies are presented. This DP network is integrated with the NoC architecture. The NoC with DP network function as a dual network. Another approach for designing adaptive routing algorithm applicable to both Mesh and Torus NoC is presented by Duato in [11]. Virtual Channels (VCs) are use in this approach. Each physical channel is shared between k ($k > 1$) VCs. VCs are categorised into two classes: deterministic class and adaptive class. Thus the network is divided into deterministic and adaptive networks. Deadlock handling policy need to apply while virtual channels are requested by group of packets packets. An adaptive routing algorithm using dedicated buffer for flow control is presented in [92]. The NoC router architecture becomes complex on embedding adaptive routing algorithms in the router. In many adaptive routing, virtual channels are used to avoid deadlock. Due to use of virtual channel and complex router architecture, power consumption increases in adaptive routing.

2.4 Deadlock Avoidance in a Routing Algorithm

Deadlock is a circumstance in NoC where a group of packets are unable to make any progress because they are holding and waiting on another in a cyclic manner to release buffer [3]. Occurrence of deadlock in an NoC has the potential to halt the complete system eventually. The deadlock avoidance in NoC has been drawing attention of researchers for a few decades [1, 2, 11–14, 29]. In this section we discuss the important works on deadlock avoidance in NoC context and figure out the possible research gap.

2.4.1 Up*/Down* and Turn Model Approach

Up*/Down* and Turn model are two deadlock avoidance approaches where certain restriction are imposed on movement of packets. For avoiding deadlock in any topology, the Up*/Down* routing approach uses spanning tree corresponding to the given network [10]. Due to the absence of cycle in a spanning tree, deadlock due to cyclic dependency between channels are eliminated. No additional resources are used for avoiding deadlock in this approach. A deadlock free dynamic reconfiguration network based on Up*/Down* routing is presented in [93]. Routing paths are highly restricted and most of the packets use non minimal paths in Up*/Down* routing approach [94].

Turn model theory [1, 9] is used for detecting cyclic dependency and designing deadlock free routing algorithms by avoiding those cycles. No additional resources are used for avoiding deadlock in this approach. In case of Torus NoC, it is not possible to create a minimal routing algorithm without using VCs [95]. According to Turn model, a packet is allowed to use a wraparound channels only at its first hop [1]. Thus, the scope of using wraparound channel is limited in this approach. *In our work, we propose a design approach for routing algorithm for Torus NoC that uses wraparound channels efficiently.* Another approach based on Turn model concept is presented in EbDa [14]. The author claims it to be scalable for arbitrarily large dimension. It partitions channels into disjoint sets without containing any cyclic dependency. Transitions between the partitions are allowed with a defined order. EbDa method is applicable for Torus NoC with help of VC for breaking the cyclic dependency across a ring.

2.4.2 FirstHop Routing for Avoiding Deadlock in Torus NoC

Torus NoC needs special care as the topology is more vulnerable towards deadlock due to the inherent cyclic paths via wraparound channels [95]. It is not so obvious to predict deadlock accurately in Torus NoC using the available Turns presented in Turn model [1] . For formation of dependency cycle between packets in a deadlock situation, one necessary criteria to fulfil is that the destination for each packet in that dependency cycle are at least two routers away from the source router. Because, if source and destination are only single hop away, i.e., next to each other, all such packets will get delivered eventually after being transmitted from the source routers as there is no chance for dependency with a third party. Based on the same principle, the FirstHop routing approach is proposed to break the dependency at certain point. In FirstHop approach, wraparound channels are allowed to use only from the boundary router. Since packets from rest of the routers cannot move to boundary for using a wraparound channel, the resource dependency is discontinued. Other approaches of avoiding deadlock in Torus includes the use VCs.

2.4.3 Dally's Approach with Virtual Channel

Dally et al. [2, 96] propose a theorem that a routing algorithm is deadlock free if there is no cycle in the channel dependency graph. A Torus routing chip is developed by using [96]. In their next work [2], constructing deadlock free routing algorithm for an arbitrary communication network using VCs are shown. The cycles in the channel dependency graph are removed by splitting physical channels into group of VCs [2]. For eliminating the cycles, VCs are ordered and routing algorithm is restricted to route packets in decreasing order of VCs. In date line approach [3], two classes of VCs are used. A packet is forced to use another VC class after crossing the date line. Date line is just a marking on a row and a column. Crossing that row or column, a separate VC class need to be used. Thus it prevents from forming a cyclic resource dependency.

2.4.4 Duato's Approach with Escape Path and Virtual Channel

Duato et al. [11, 27] present an approach of designing deadlock free adaptive routing using VCs. An approach of constructing adaptive routing function allowing cyclic dependencies between channels, provided that there are alternative paths without cyclic dependencies for forwarding a packet towards destination are presented [13, 27]. Xiang et al. [97] proposed an adaptive routing algorithm on Torus NoC using VCs and additional buffer in the input port. In their proposed algorithm, two VCs are required. One set of channels are used by a known deadlock free routing in a Mesh sub-network. The other set of channels are used for making the algorithm adaptive.

2.4.5 Bubble Flow Control with Dedicated Buffer

Bubble flow control is a flow control technique used for avoiding deadlock in Torus NoC [28, 29, 92, 98–100]. Packet injection or changing direction of a packet is only allowed if there are at least two empty buffer slots in the direction required by the packet. Therefore, at least one empty buffer slot always remains free in the ring. That free buffer acts as bubble guaranteeing that at least one packet is able to progress. Flit bubble flow control [30] works on the same principle. It maintains one free flit size buffer slot inside a ring to avoid deadlock. Forwarding a flit does not reduce overall free buffer in a ring since it leaves it previously occupied free slot. Only injection of packet into ring reduces free buffer amount. Injecting a packet or changing direction is allowed if the receiver's input port has one more free slot remains after receiving the packet.

2.4.6 Other Approaches

A survey on different approaches for designing routing algorithms is presented in [95]. In that work, deadlock avoidance approaches like dropping packets, numbering network resources for resource allocation in a specific order, putting restrictions on changes of routing directions for packets are presented. This work claims that for a k-ary n-cube topology with $k > 4$, it is impossible to construct a minimal deadlock free routing algorithm without using extra channels [95]. In [17], cyclic dependency is broken by applying in place swapping of packets between adjacent router. No extra buffer is used in this approach. Applying in place swap process periodically is one overhead in this approach and the router micro-architecture needs to be upgraded accordingly for this in place swap operation.

2.4.7 Challenges and Objectives

The inherent cyclic path in each row and column of a Torus NoC makes the topology deadlock prone. VCs or extra buffers need to used for developing a deadlock free routing algorithm in Torus NoC. Though, Up*/Down* routing approach avoid deadlock without using additional resources [10], routing paths are mostly non minimal or lengthy [94]. In FirstHop routing, a packet is allowed to use the wraparound channel only at its first hop so that cyclic path is discontinued for avoiding deadlock in Torus NoC [1]. Therefore, the leverage of wraparound channels are not utilised for most of the packets. To the best of our knowledge, there is no other approach available for avoiding deadlock in Torus NoC without additional buffer or virtual channels (VC). *The objective identified from this background study is to work on a deadlock avoidance approach for Torus NoC without using any additional buffer or VC and at the same time increase utility of wraparound channels.*

Detecting confirmed deadlock with an exact deadlock scenario is a challenging task due to the huge state space of NoC [41]. Popular NoC simulator like Booksim [24] and Gem5 [25] report a warning message for possible deadlock scenarios based on a predefined threshold value. It is very helpful to get the exact deadlock scenario after a deadlock is detected. This deadlock scenario would be helpful is formulating deadlock avoidance. *A formal model of NoC considering detailed NoC components would be helpful to detect confirmed deadlock along with deadlock scenarios. One objective of this thesis is to develop a deadlock detection framework using formal model of NoC considering NoC component in detail.*

2.5 Conclusions

In this chapter we survey the literature on formal modeling and verification of NoC and different deadlock avoidance approaches in NoC. We have identified research gaps in the verification of NoC using detailed level modeling. Detecting confirmed deadlock with deadlock scenarios are missing in the existing works. We have also identified lack of deadlock avoidance approaches that are applicable to Torus NoC without using any additional resources. There is a scope for developing deterministic deadlock free routing algorithms for Torus NoC. In this thesis we will address these problems.

3 Formal Modeling of NoC using FSM and Verification of Starvation using Model Checker

3.1 Introduction

Modeling NoC components in detail level is essential for formal verification to ensure that an NoC works correctly after manufacturing of the real hardware. Modeling NoC close to hardware functionality and scalability of the verification method are the two primary challenges in NoC verification. In most of the existing NoC verification works, detailed modeling of the complete NoC is system is missing [31–34,36,38,39,101]. We have considered NoC functional units in a comprehensive way in this thesis. It is convenient to implement

NoC using Finite State Machine (FSM) in a model checker like NuSMV [22]. Therefore, we have chosen FSM for modeling NoC so that the model can be easily encoded and various properties can be checked using a state-of-the-art model checker.

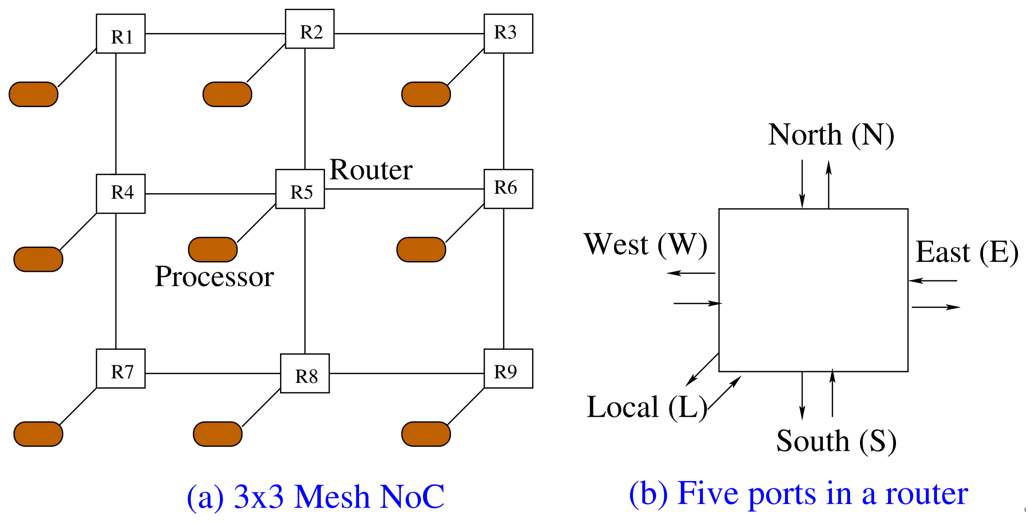


Figure 3.1: *A 3x3 Mesh NoC and five bidirectional ports in a router*

3.1.1 NoC Router Components

A 3x3 mesh NoC with routers and the connected processors is shown in Fig. 3.1(a). R1, R2, R3, etc., are the routers shown with square boxes. The brown oval shaped units represent the processors. Each router constitutes of buffers for storing packets, switch for computing the route for a packet and diverting it to desired output port and an arbiter to control the transmission of packets via an output port. We have considered NoC functional units or NoC components like buffer, switch and arbiter for modeling NoC. An NoC works with help of the collaborative efforts between all these units. *Buffers* are present at the input ports of a router. The five bidirectional ports in a router are shown in Fig. 3.1(b). They are used as temporary storage for a packet before being transmitted in the desired path. The route computation of a packet is performed by another functional unit called *switch*. After the route computation is performed, the packet requests for the expected output port. There may be multiple packets competing for the same output port at the same time. In each output port, an *arbiter* is present that resolves the conflict by picking one packet from the competing packets. The selected packet is transmitted to the next router.

In NoC context, the output port and its connection to the next router are considered as a resource of conflict. If multiple packets from different input ports in a router try to access that output port continuously, there should be fairness such that competing packets from all input ports get a fair chance to be transmitted. This is called starvation-freedom. Starvation-freedom ensures fairness in resource allocation for the packets from all input ports. In this chapter, we present verification of starvation-freedom using our FSM based NoC model.

3.1.2 Contributions

We have modeled an NoC using FSM, considering buffer, switch and arbiter as individual functional units. Synchronization between these functional units is maintained using dedicated FSMs. As an application of our model, we target verification of starvation-freedom considering fixed-priority and round-robin arbiters in this work. We encode our FSM based NoC models in NuSMV model checker [22] and give the starvation-freedom specification using Linear Temporal Logic (LTL) [43] for verification. The model checker reports if the given property is satisfied or not. Specifically, the contribution of our work is summarized as follows:

- Detailed modeling of NoC router components like buffer, switch, arbiter using FSMs are presented.
- Demonstrate the designing of fixed-priority and round-robin arbitration policies using FSMs for selecting a packet from more than one competing packets.
- For the correctness of the NoC model, verification of synchronization within a router, progress between NoC components and loss-less transfer of packets are performed.
- As an application of the FSM based model, verification of starvation-freedom for fixed-priority and round-robin arbitration are demonstrated.

- Verification time is reduced significantly by invoking parallel threads for individual routers.

The rest of this chapter is organized as follows. A brief description of FSM and the short form used for describing NoC model is presented in Section 3.2. Formal modeling of NoC using FSM is presented in Section 3.3. Verifying the correctness of the FSM model is described in Section 3.4. Application of the FSM model is described in Section 3.5. Experimental results are presented in Section 3.6. Finally, we conclude the chapter in Section 3.7.

3.2 Finite State Machine and the Naming Convention

In this section, we briefly describe Finite State Machine (FSM). The numbers of NoC routers and NoC components vary depending on the size of the NoC. We consider a 3x3 Mesh NoC as a reference NoC for the convenience in presenting our FSM based NoC Model. The same procedure is applied while modeling and implementing NoC of bigger sizes in the thesis.

3.2.1 Finite State Machine

The Finite State Machine (FSM) is being used by research community for modeling of transition systems in various applications [66–73]. Each FSM has a finite number of states and a transition takes place from one state to another state based on predefined transition rules. Deterministic Finite Automata (DFA) is a special class of FSM where each transition is a unique path from one particular state to another particular state with respect to a specific input or with respect to the truth value of a specific condition. A formal definition of a DFA is given below [102].

Definition 3.2.1. *A Deterministic Finite Automata is represented using a quintuple,* $\{Q,\ \Sigma,\ \delta,\ q_o,\ F\}$.
Here, Q *represents a finite set of states that constitute the automata,*
Σ *represents the input symbol that are used for state transition,*

δ *represents the transition function* $\delta : Q \text{ x } \Sigma \rightarrow Q$*,*

q_o *represents the initial state,* $q_o \in Q$*,*

F *represents the set of final sets,* $F \subseteq Q$*.*

In this thesis, corresponding to each NoC functional unit a FSM is modeled. A FSM state indicates current status of the corresponding NoC functional unit. Each FSM starts from an initial or starting state q_o, indicated with an incoming arrow. The current state of a FMS keeps on changing with respect to the movement of packet. Once the transmission of a packet is over, the FSMs corresponding to that transmission return to their initial states again. Therefore, we consider the initial state as the final state as well, i.e., $F = \{q_0\}$ in our FSM based NoC modeling. This process continues for the transmission of other packets. This is a continuous process and the states of FSMs keep changing.

There present numbers of FSMs in a complete NoC model. In an NoC, each of the functional unit interacts with some other NoC functional units for proper functioning of the system. In the same context, each of the FSM interacts with other FSMs in the NoC model. All the transitions in a FSM are controlled by the states of other FSMs with whom the FSM interacts and all transitions happen with a hand checking fashion. Transitions for the FSMs are described as $\delta : Q \text{ x } \Sigma \rightarrow Q$. In context of our FSM based NoC model, Q represents the states in the FSM under consideration and Σ represents the FSM states information for the complete NoC model. Alternately, Σ represents the global states information of the complete NoC that are used for defining state transitions δ in each individual FSM.

3.2.2 Short forms and the Naming Convention

We have used the Fig. 3.1(a) as a reference while describing FSM model for different NoC components. We consider the the South port of router R2 in Fig. 3.1(a) while demonstrating FSM model for buffer, switch and arbiter. The five ports present in a router are shown in Fig. 3.1(b). While describing buffer and switch at the South port of router R2, we are considering the movement of packet towards router R2 from router R5. While describing the arbiter at the South port of router R2, we are considering the movement of packet from

Table 3.1: *Short form used for describing transitions in FSMs*

Functional Unit	**Short Form**	**Functional Unit**	**Short Form**
Arbiter	Ar	Buffer	Bu
Priority	Pr	Return	Re
Switch	Sw	Sync	Sy

router R2 towards router R5. Every time we mention a router in describing the FSM model, that router corresponds to Fig. 3.1(a).

For accommodating space while describing the transitions in a FSM diagram, we use the short form[1] as shown in Table 3.1. For a buffer we use "Bu", for an arbiter we use "Ar", for a switch we use "Sw", etc. in the FSM diagrams in this chapter. Besides these short forms in Table 3.1, we use L, E, W, N and S for the Local port, East port, the West port, the North port and the South port, respectively. The Name of an NoC component is associated with the router name as a prefix and the port name as a postfix to the component name. For example, R2BufferS indicates the buffer associated the S port of router R2, R5ArbiterN indicates the arbiter present at the North port of router R5, etc. These component names are used with short form in the FSM diagrams. For example, R2BufS is used for R2BufferS, R5ArN is used for R5ArbiterN and so on as per the Table 3.1. Thus, the information about the associated router and associated port of NoC components are made self explanatory by using the naming convention for the NoC components.

3.3 Formal Modeling of NoC using FSM

In this section, we describe the FSM models of various components of the NoC and their synchronization. We have modeled buffer, switch and arbiter for an NoC router. Synchronization between them is maintained by using dedicated FSMs. Considering components from all NoC routers and maintaining the synchronization between them give a complete model for the NoC.

[1]Short form in Table. 3.1 are used only in FSM diagrams for representing state transitions for clarity of the figures. We use full forms in other places for better readability.

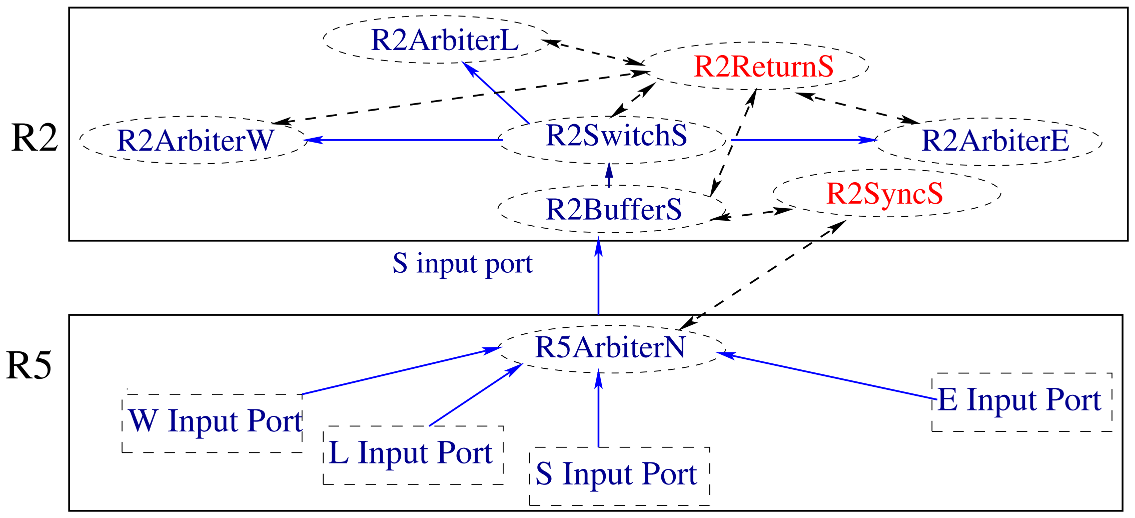


Figure 3.2: *Movement of packets from router R5 to R2, and synchronization using Return and Sync FSMs*

3.3.1 High-Level Overview of the Movement of Packets

We have shown the movement of packets from router R5 to router R2 (of Fig. 3.1) in the Fig. 3.2. The blue solid arrows in Fig. 3.2 show the possible paths for the movement of packets from router R5 to router R2. Movements of packets are shown from the North output port of router R5 to the South input port of router R2. An arbiter named R5ArbiterN is present at the North output port of router R5 as shown in the Fig. 3.2. The packets from four input ports, namely the Local, East, West and South, can transmit packet via the North output port of router R5. If packets from more than one input ports compete for the same output port at a time, the R5ArbiterN has to select only one packet at a time based on the arbitration policy. The arbiter transmits the selected packet to the router R2. The transmitted packet is stored in the South input port buffer of router R2. This buffer is named as R2BufferS as shown in Fig. 3.2. The route computation for the packet is performed at the R2SwitchS. As per the position of router R2 in the reference Fig. 3.1, there are three possible directions where the packet can move. These directions are towards the East port or towards the West port or towards the Local port. Based on the destination address of the packet, R2SwitchS determines the output port and requests the corresponding arbiter for transmission. If R2 is the destination router for the packet, it has already reached the destination router. In such case, the packet would be directed to the arbiter at the local port and will eventually be delivered to the local core. If R2 is not the destination router, the packet would move to the adjacent router in the East or West direction based on the routing decisions.

3.3.2 Synchronization between NoC Components

Once a packet is moved to the next router, the buffer storage in the previous router needs to be cleared. The switch and the arbiter are also ready for processing the next packet only after transmitting the current packet. Therefore, maintaining synchronization between buffer, switch and arbiter and between two adjacent routers are important for smooth functioning and lossless packet transmission in NoC routers. We maintain the synchronization between the buffer, switch and arbiter with help of two dedicated FSMs named *sync* and *return* in each input port. The synchronization between FSM models are maintained with the help of the handshaking principle.

3.3.2.1 Synchronization between two Routers

The synchronization between two routers is controlled by a *sync* FSM in our FSM based NoC model. A buffer that presents at the input port of an NoC router, accepts packets from an arbiter in the adjacent router. Before transmitting a packet, the arbiter has to ensure that the buffer in the next router is free. Moreover, the packet cannot be deleted from the previous router until it is safely transferred to the next router. The *sync* FSM maintains the synchronization between the arbiter and buffer between two routers as shown in Fig. 3.2. The detailed modeling of a *sync* FSM is described in the Subsection 3.3.3. The high level overview depicting the interaction of a *sync* FSM with other NoC functional units at the S input port of router R2 is shown in Fig. 3.2. The dotted double ended arrow connecting R5ArbiterN and R2BufferS via R2SyncS indicates synchronization between two routers.

3.3.2.2 Synchronization within Router Components

The synchronization between NoC components within a router are maintained using a dedicated FSM named *return* in our FSM based NoC model. After transmitting a packet from a router, all FSMs in that router that are involved in that transmission need to return to their initial states. The detailed model for a *return* FSM is described in the Subsection 3.3.5. The high level overview depicting the interaction of a *return* FSM with other NoC functional units at the S input port of router R2 is shown in Fig. 3.2. The dotted double-ended arrows connecting R2ReturnS with R2BufferS, R2SwitchS and all the arbiters in roueter R2 represents the synchronization between components in R2.

The paths for packets movements are shown with the blue solid arrows in Fig. 3.2. In a five-port router, we have five such synchronization FSMs corresponding to each port. In this work, we present an input buffer router model. Design of an output buffer router model is similar with minor changes in synchronization.

3.3.3 Modeling Buffer using FSM

Buffer is present at the input port of a router for storing an incoming packet until it is transmitted. Synchronization between two routers has to be taken care of before storing a packet into the buffer. The *sync* FSM maintains synchronization between the *buffer* and the adjacent router from which the buffer accepts packet. The basic idea here is that the *sync* will read the current state of the buffer and the arbiter in the adjacent router. It allows transfer of packets only if there is a slot free in the buffer and allows the buffer to change its state. Therefore, we present the FSM model for *sync* and *buffer* together.

3.3.3.1 FSM model of Sync

A *sync* FSM synchronized with the arbiter in the adjacent router. The FSM model for *sync* is shown in Fig. 3.3(b). We consider the *sync* FSM at the S input port of router R2, with reference to the 3x3 Mesh NoC in Fig. 3.1(a), for explaining the synchronization with buffer at the same input port. Following the naming convention, we name this *sync* FSM as R2SyncS. Here, R2 indicates the associated router name and S indicates the associated port name. The FSM model for R2SyncS and all transitions are shown in Fig. 3.3(b).

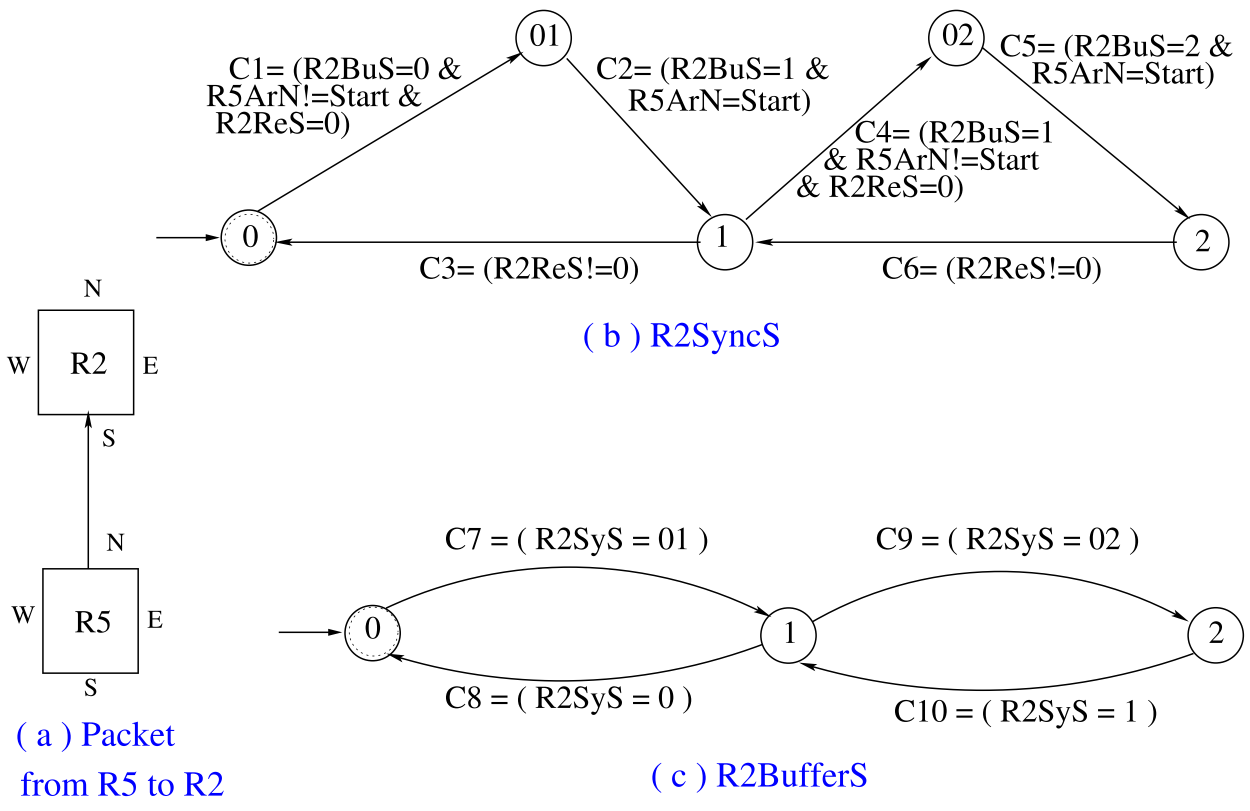

C1= (R2BuS=0 & R5ArN!=Start & R2ReS=0)
01
C2= (R2BuS=1 & R5ArN=Start)
02
C5= (R2BuS=2 & R5ArN=Start)
C4= (R2BuS=1 & R5ArN!=Start & R2ReS=0)
0
1
2
C3= (R2ReS!=0)
C6= (R2ReS!=0)
(b) R2SyncS
N
W
R2
E
S
N
W
R5
E
S
C7 = (R2SyS = 01)
C9 = (R2SyS = 02)
C8 = (R2SyS = 0)
C10 = (R2SyS = 1)
(a) Packet from R5 to R2
(c) R2BufferS

Figure 3.3: *Buffer and sync: (a) Packets from R5 to R2, (b) R2SyncS: Synchronizing between R2BufferS and R5ArbiterN, (c) R2BufferS: Buffer at S input port of R2*

Current state of *R2SyncS* = 0/ 1/ 2 means the R2BufferS contains zero/ one/ two packet(s). *R2SyncS* = 01/ 02 are the intermediate transition states used by the corresponding buffer to change its state. Initially R2SyncS is in state 0. The condition for the transition from (0 → 01) in Fig. 3.3(b) is **C1** = (R2BuS = 0 ∧ R5ArN ≠ Start ∧ R2ReS = 0). Here, (R2BuS = 0) means buffer is free as no packet is stored in R2BufferS. The second condition (R5ArN!=Start) means the R5ArbiterN is ready to transmit a packet. The third condition (R2ReS=0) means the South input port is ready to receive a packet for the transmission. If all three conditions are satisfied, the transition (0 → 01) takes place. The next transition (01 → 1) in Fig. 3.3(b) takes place if the R5ArbiterN of router R5 has returned to its initial state after transmitting the packet and the R2BufferS also stored the packet and updated its state. This is represented by the condition **C2** = (R2BuS = 1 ∧ R5ArN = Start). Two transitions are possible from the state 1 as shown in Fig. 3.3(b). If the transmission for the packet is over, i.e. **C3** = (R2ReS ≠ 0), the FSM returns to the initial state (1 → 0). Here, (R2ReS ≠ 0) means transmission of a packet is completed. On the other hand, if another packet has arrived before the transmission is completed, i.e. , **C4** = (R2BuS = 1 $\wedge$ R5ArN $\neq$ Start $\wedge$ R2ReS = 0), the FSM state changes with the transition (1 $\rightarrow$ 02). Here, (R2BuS = 1) indicates that one packet is stored in the buffer and it can accommodate another packet. We consider a two slot buffer in this example. In similar way, all other transitions in Fig. 3.3(b) take place.

3.3.3.2 FSM model of Buffer

We consider a buffer (R2BufferS) that has the capacity for storing two packets in the presented model. All the transition in a buffer is controlled by the status of the corresponding *Sync* FSM. R2BufferS uses the current states of R2SyncS FSM for its transitions. The FSM models and transitions for R2BufferS are shown in Fig. 3.3(c). The condition **C7** = (R2SyS = 01) in Fig. 3.3(c) indicates that a packet is transmitted from the adjacent router. R2BufferS stores that packet and changes its state (0 $\rightarrow$ 1). The condition **C8** = (R2SyS = 0) indicates that the transmission is over and buffer has to return to initial state. The condition **C3** = (R2SyncS = 02) indicates that another packet has arrived when the buffer is already storing one packet. Therefore, the second packet is accommodated and the transition (1 $\rightarrow$ 2) takes place Fig. 3.3(c). In this way, the state of a *buffer* is controlled by a *sync* FSM.

3.3.4 FSM Model of Switch

A switch accepts packets from the buffer and diverts it to the desired output port for transmitting into the next router. We consider the R2SwitchS at S port of router R2 for explaining the design. The FSM representation of switch R2SwitchS is shown in Fig. 3.4. It remains in its initial state Wt (Wait) until no packet arrives in the buffer. When a packet arrives and the condition **C1** = (R2BuS $\neq$ 0 $\wedge$ R2ReS = 0) is satisfied, the FSM changes its state to C (Wait $\rightarrow$ C). Here, (R2BuS $\neq$ 0) in **C1** indicates the buffer is not empty. At least one packet is stored in the buffer which needs to be transmitted towards its destination. The condition (R2ReS = 0) in **C1** means the input port is ready for processing a packet.

The state C stands for compute where routing decision is taken.

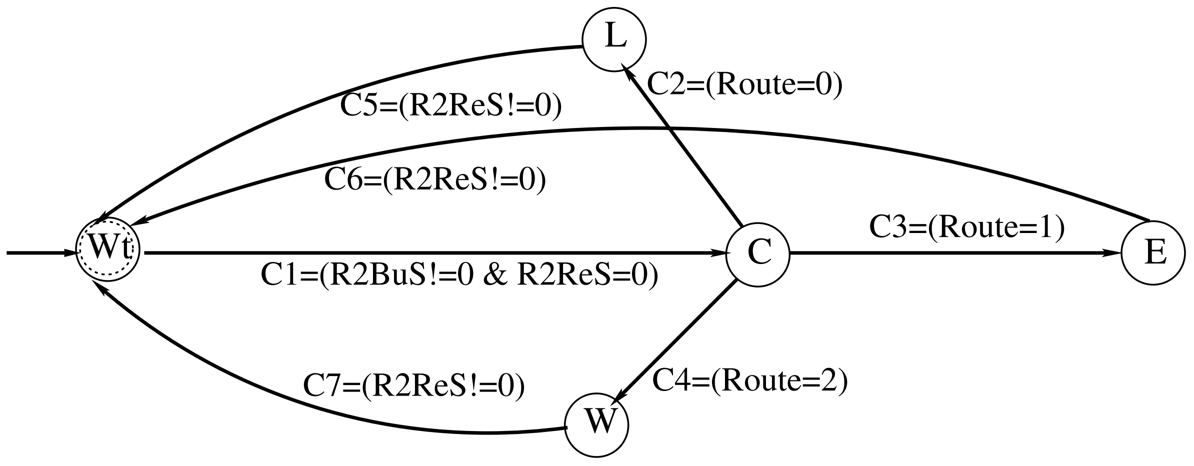


Figure 3.4: *R2SwitchS: Switch at South Port of R2*

All possible routing directions of a packet is possible. One routing direction is selected based on the given routing algorithm. At the state *C*, a variable named *Route* is considered. It takes any values from 0, 1 and 2, indicating the direction towards L, E and W ports, respectively. We have not considered any specific routing algorithm here. Considering a routing algorithm demands for packet information as well. Instead, all possible paths would be considered in the verification by taking one path at a time. In simulation approach, the routing direction of a packet is decided by invoking a routing algorithm. Since this work does not intend to develop a simulation framework, specific routing algorithm is not considered. For implementing a formal model based simulation framework using the proposed FSM model, routing algorithm will be invoked when the FSM reaches the state *C*. We have presented this approach in our next work [41]. Based on the routing direction, R2SwitchS reaches the appropriate state. Once the packet is transmitted, the condition C5(/C6/C7) becomes True and the FSM returns back to the initial state *Wt*. Here, **C5** = **C6** = **C7** = (R2ReS $\neq$ 0). It means the packet is transmitted and all the respective FSMs need to return to their respective initial states. Until then R2ReturnS would not changes its state and the condition (R2ReS $\neq$ 0) remains True.

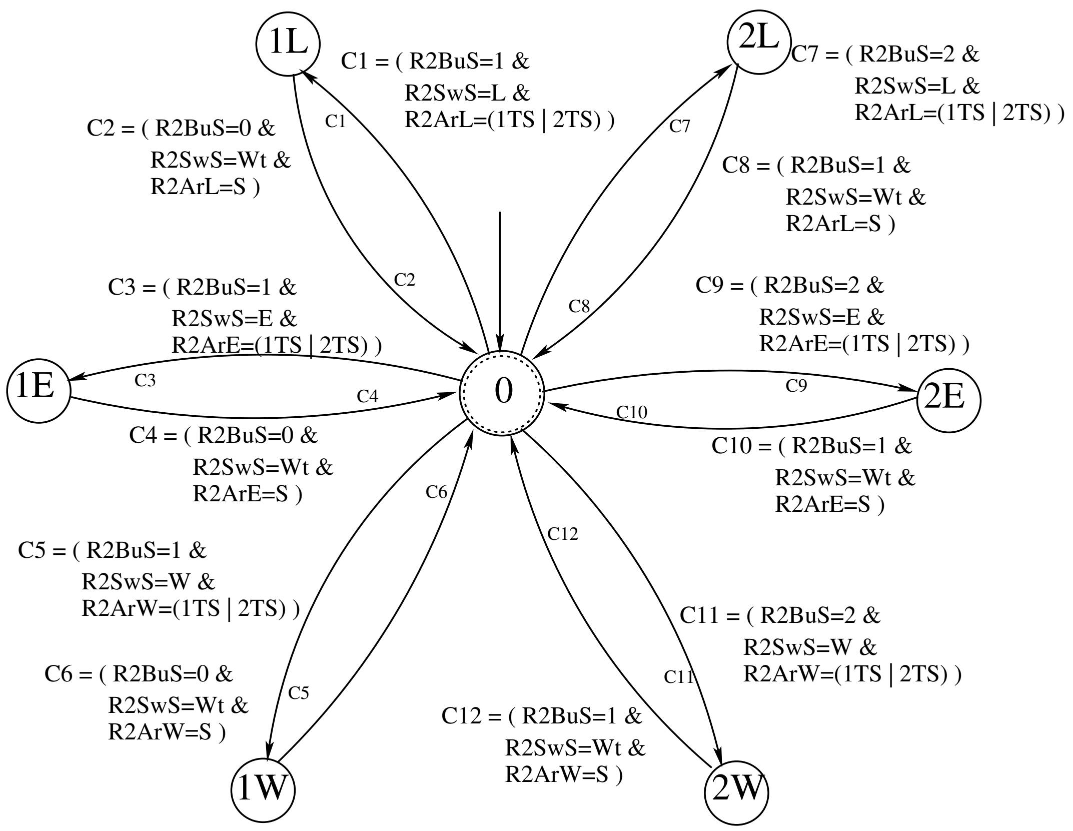

1L
2L
1E
2E
1W
2W
0
C1 = (R2BuS=1 & R2SwS=L & R2ArL=(1TS | 2TS))
C2 = (R2BuS=0 & R2SwS=Wt & R2ArL=S)
C3 = (R2BuS=1 & R2SwS=E & R2ArE=(1TS | 2TS))
C4 = (R2BuS=0 & R2SwS=Wt & R2ArE=S)
C5 = (R2BuS=1 & R2SwS=W & R2ArW=(1TS | 2TS))
C6 = (R2BuS=0 & R2SwS=Wt & R2ArW=S)
C7 = (R2BuS=2 & R2SwS=L & R2ArL=(1TS | 2TS))
C8 = (R2BuS=1 & R2SwS=Wt & R2ArL=S)
C9 = (R2BuS=2 & R2SwS=E & R2ArE=(1TS | 2TS))
C10 = (R2BuS=1 & R2SwS=Wt & R2ArE=S)
C11 = (R2BuS=2 & R2SwS=W & R2ArW=(1TS | 2TS))
C12 = (R2BuS=1 & R2SwS=Wt & R2ArW=S)
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12

Figure 3.5: *R2ReturnS: Synchronization between S port Buffer, S port Switch and Arbiters at L, E and W ports*

3.3.5 FSM Model of Return

After transmitting a packet, the corresponding *buffer*, *switch* and *arbiter* return to their initial states by checking the status of the *return* FSM. If the current state of the *return* FSM is other than 0, it indicates that the packet is transmitted and corresponding FSMs involved in that transmission must return to their initial states. Once corresponding FSMs return to their initial state, the state of the *return* FSM changes to initial state 0. The *return* FSM R2ReturnS at the S input port of router R2 is shown in Fig. 3.5.

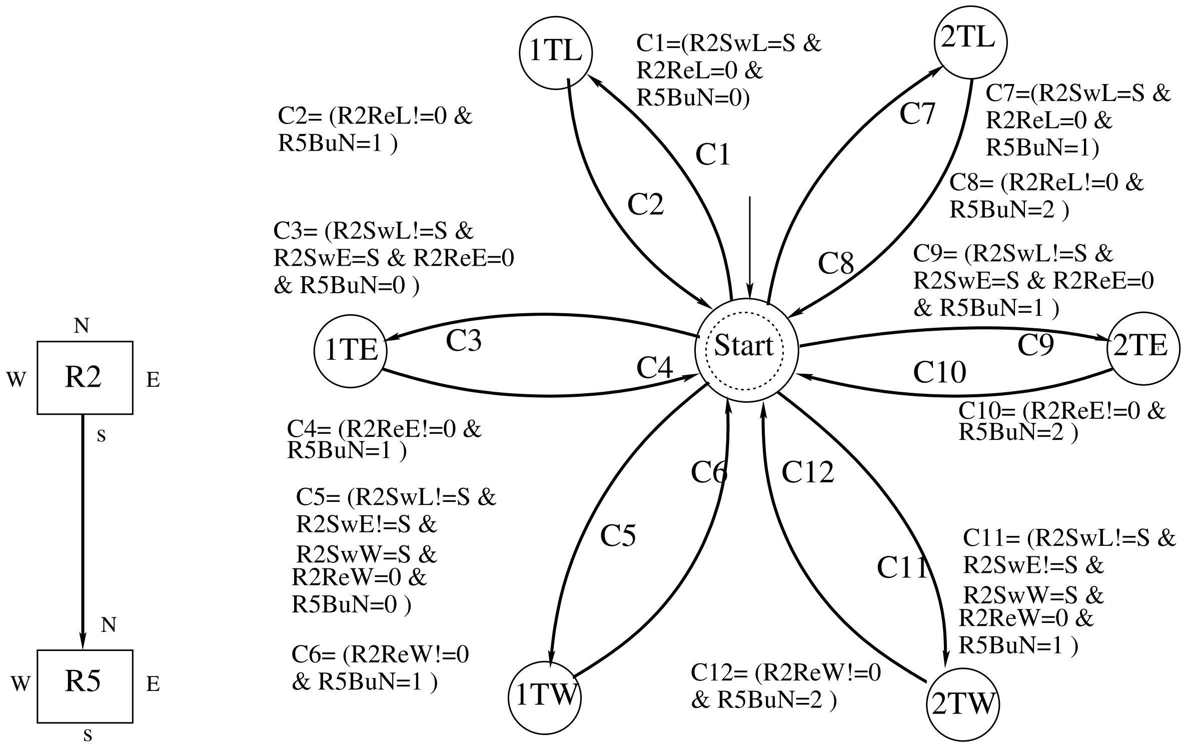
The initial state of R2ReturnS is 0. If R2BufferS contains one(/two) packet(/packets) and a packet is transmitted via E output port and the condition C3(/C9) is satisfied, the state of the R2ReturnS changes to $1E(/2E)$. If the condition **C3** $=$ ((R2BuS $= 1) \wedge$ (R2SwS $=$ East) $\wedge$ (R2ArE $=$ (1TS $\vee$ 2TS)) is satisfied the transition ($0 \rightarrow$ 1E) takes place. The meaning of **C3** is that the packet at R2BuS is destined toward the East output port and it has won the arbitration at R2ArE. Here, (R2BuS $= 1$) means the buffer contains one packet, (R2SwS = East) means the packet is heading toward the East output port and (R2ArE = (1TS ∨ 2TS) means the packet from the South input port has won the arbitration at the East output port. If the condition **C4** = ((R2BuS = 0) ∧ (R2SwS = Wait) ∧ (R2ArE = Start)) is satisfied the transition (1E → 0) takes place. The meaning for (R2BuS = 0) is the buffer return to initial state, (R2SwS = Wait) means R2SwitchS return to its initial states and (R2ArE = Start) means R2ArbiterE returns to its initial state. Therefore, if **C4** is satisfied the transition (1E → 0) takes place, i.e., R2ReturnS returns to initial state. After all the corresponding FSMs related to this transmission return to their initial states, the condition C4 is satisfied and the R2ReturnS also returns to its initial state. For the condition **C9** buffer is considered as (R2BuS = 2). Other conditions are same as that of the condition **C3**. Similarly, for the condition **C10**, the sate of the buffer becomes (R2BuS = 1) from the state (R2BuS = 2). Other conditions are same as that of the condition **C4**. In similar way, all other transitions for R2ReturnS takes place as shown in Fig. 3.5.

3.3.6 Approach for Designing Virtual Channels

Blocking problem of one packet by another packet is resolved with the help of virtual channels (VCs) where a set of VCs share the same physical channel [41, 103]. In VC, a *buffer* is restructured into separate smaller *buffers*. Corresponding to each VC there is a need for separate *buffer* (smaller *buffers*) with corresponding *sync*, *switch* and *return* FSM. An *arbiter* resolves conflict for physical channel at an output port which is shared by a set of VCs. If the number of VCs increases, the states in an *arbiter* also increase. For the sake of simplicity, in this work we present *arbiter* with single VC only.

3.3.7 FSM Model of an Arbiter

Multiple packets from different input ports compete for the same output port at the same time. These conflicts are resolved by an arbiter with the help of an arbitration policy like round-robin, weighted round-robin, first-come-first-serve, fixed-priority, etc.. In this work, we design fixed-priority and round-robin policies in the arbiter.


N
W
R2
E
S
N
W
R5
E
S
1TL
2TL
1TE
Start
2TE
1TW
2TW
C1=(R2SwL=S & R2ReL=0 & R5BuN=0)
C2= (R2ReL!=0 & R5BuN=1)
C3= (R2SwL!=S & R2SwE=S & R2ReE=0 & R5BuN=0)
C4= (R2ReE!=0 & R5BuN=1)
C5= (R2SwL!=S & R2SwE!=S & R2SwW=S & R2ReW=0 & R5BuN=0)
C6= (R2ReW!=0 & R5BuN=1)
C7=(R2SwL=S & R2ReL=0 & R5BuN=1)
C8= (R2ReL!=0 & R5BuN=2)
C9= (R2SwL!=S & R2SwE=S & R2ReE=0 & R5BuN=1)
C10= (R2ReE!=0 & R5BuN=2)
C11= (R2SwL!=S & R2SwE!=S & R2SwW=S & R2ReW=0 & R5BuN=1)
C12= (R2ReW!=0 & R5BuN=2)
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12

(a) Packet from R2 to R5

(b) Fixed–priority: R2ArbiterS

Figure 3.6: *R2ArbiterS: Fixed-priority arbiter at S port of router R2*

3.3.7.1 FSM Model of Fixed-priority Arbiter

In the fixed-priority arbiter, priority is fixed. We choose the priority sequence of Local > East > West > North > South. It means, packets from the L input port has the highest priority and the packet from the S input port has the lowest priority. The FSM representation for the fixed-priority arbiter, (R2ArbiterS), present at the S port of a R2 is shown in Fig. 3.6. It transmits packet to the buffer R5BufferN in the adjacent router R5. Initially, the arbiter is in initial state *Start*. If the buffer R5BufferN in the adjacent router R5 contains zero(/one) packet and the condition C1(/C7) is satisfied, a packet from L input port wins arbitration at R2ArbiterS. The arbiter changes its state to *1TL(/2TL)* from the initial state *Start*. Here the condition **C1** = ((R2SwL = S) $\wedge$ (R2ReL = 0) $\wedge$ (R5BuN = 0)). The condition (R2SwL = S) indicates a packet at the Local input port is destined towards the South output port. The condition (R2ReL = 0) indicates the Local input port is ready for processing a packet and the (R5BuN = 0) indicates the buffer in the adjacent router R5 contains no packet and is free to receive packets. If **C1** is satisfied, the transition (Start $\rightarrow$ 1TL) takes place in Fig. 3.6. It means that the R5BufferN in the next router is free and the Local input port wins the arbitration. Once the transmission is over and the R5BufferN receives the packet and the condition C2(/C8) is satisfied, the arbiter returns to its initial state *Start* and the transition (1TL $\rightarrow$ Start) takes place. Here, the condition **C2** = ((R2ReL $\neq$ 0) $\wedge$ (R5BuN = 1)). The condition (R5BuN = 1) indicates the packet is transferred to the next router R5. The condition (R2ReL $\neq$ 0) indicates that all the FSMs related to this transmission have to return to their respective initial states as the transmission is over. The transitions due to condition C7 and C8 takes place in similar way. The only difference is the buffer in the adjacent router contains one packet and it can store another packet. All the other transitions for R2ArbiterS are carried out in a similar way as shown in the Fig. 3.6.

3.3.7.2 FSM Model of Round-robin Arbiter

In round-robin arbiter, the priority is not fixed but keeps on updating dynamically. After transmitting a packet from one port, the priority of that port is set to lowest. It helps in getting a fair chance for the packets from other ports. For simplicity, we have designed round-robin priority generator and round-robin arbiter in two separate FSMs.

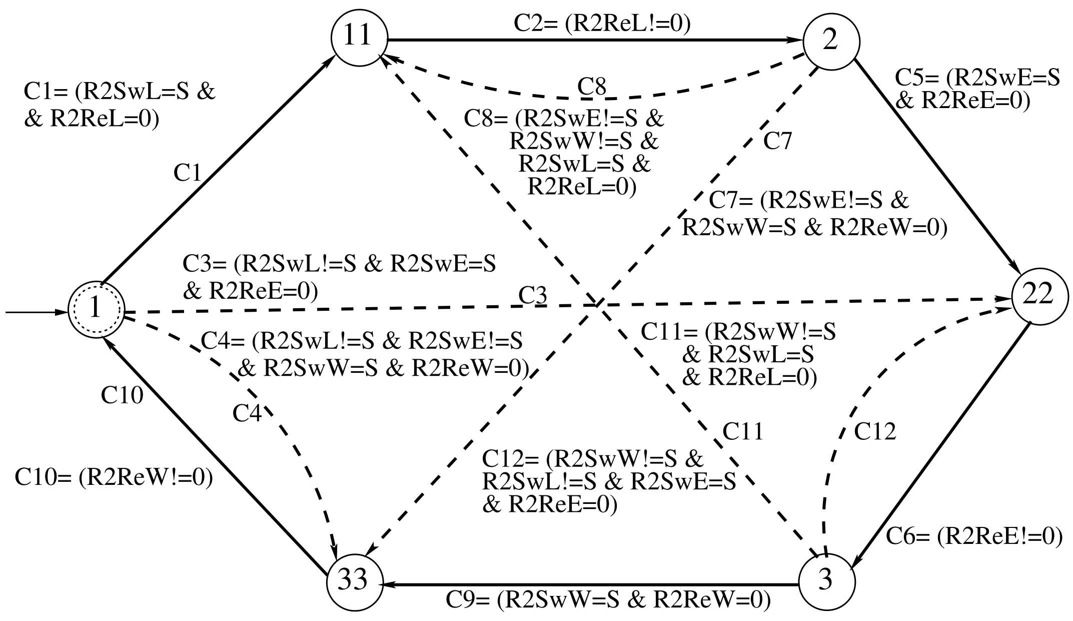
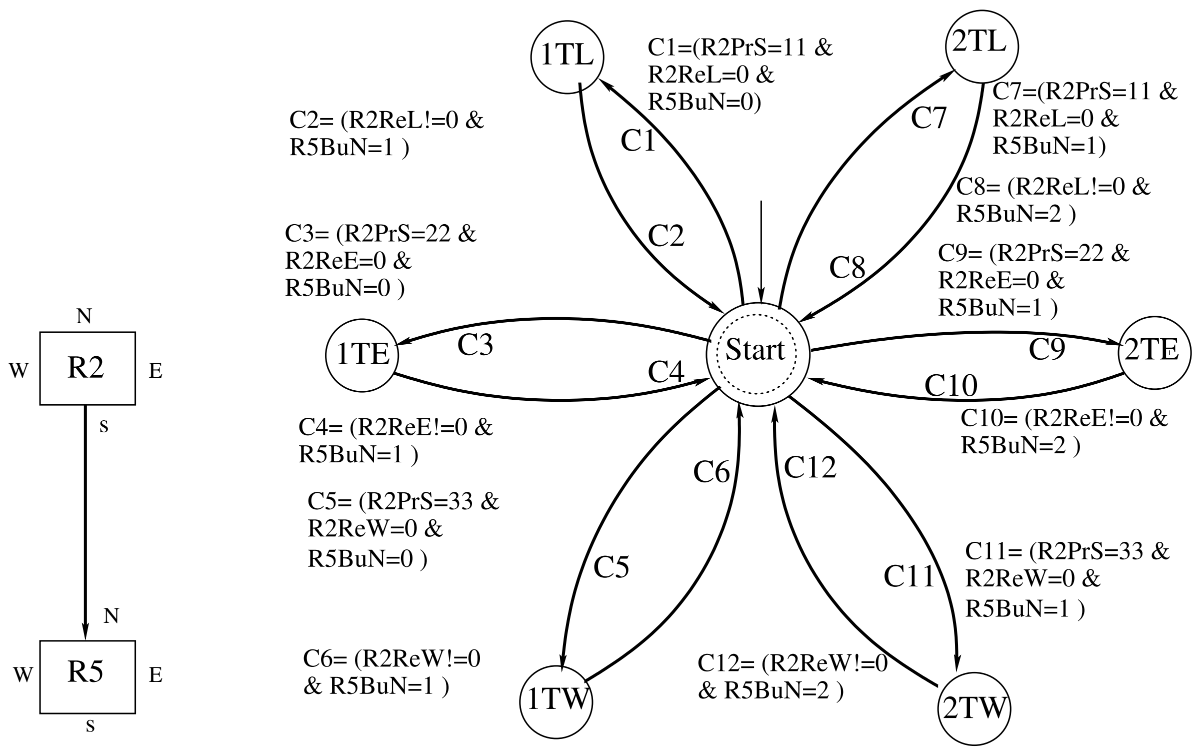

11
2
1
22
33
3
C2= (R2ReL!=0)
C1= (R2SwL=S & & R2ReL=0)
C1
C5= (R2SwE=S & R2ReE=0)
C8
C8= (R2SwE!=S & R2SwW!=S & R2SwL=S & R2ReL=0)
C7
C7= (R2SwE!=S & R2SwW=S & R2ReW=0)
C3= (R2SwL!=S & R2SwE=S & R2ReE=0)
C3
C4= (R2SwL!=S & R2SwE!=S & R2SwW=S & R2ReW=0)
C4
C10
C10= (R2ReW!=0)
C11= (R2SwW!=S & R2SwL=S & R2ReL=0)
C11
C12
C12= (R2SwW!=S & R2SwL!=S & R2SwE=S & R2ReE=0)
C6= (R2ReE!=0)
C9= (R2SwW=S & R2ReW=0)

Figure 3.7: *R2PriorityS (Round-robin priority generator at the S output port of router R2)*

Round-robin priority generator: A round-robin priority generator is shown in Fig. 3.7. Each state represents a priority value. Initially, it is in the initial state *1*. The priority values *1, 2, 3* indicate the next preferable ports to win arbitration are L, E and W port, respectively. If current state is *1* and there is packet from L input port, the state changes to *11*. The priority *11* indicates that L input port wins the arbitration. Similarly, the priority *22 (/33)* indicates that a packet presents at the E (/W) input port wins the arbitration. Transitions in round-robin priority generator set priority to a packet corresponding to an input port in round-robin fashion as shown in Fig. 3.7. Let the current state is *1* and there is no packet from L input port. Let there is a packet competing from E input port and the condition C3 is satisfied. The East input port wins arbitration and the priority changes from state *1* to state *22* in such case. Here, the condition **C3** = $((\text{R2SwL} \neq \text{S}) \wedge (\text{R2SwE} = \text{S}) \wedge (\text{R2ReE} = 0))$. The condition $(\text{R2SwL} \neq \text{S})$ means that there is no competing packet from the Local port towards the South output port and the condition $(\text{R2SwE} = \text{S})$ means that there is a packet from the East input port towards the South output port. The condition $(\text{R2ReE} = 0)$ says that the East input port is ready for processing a packet. If **C3** is satisfied, the transition $(1 \rightarrow 22)$ takes place. It indicates that there is a packet from the East port to the South port and no packet is from the Local port to the South port is competing. Therefore, priority changes to 22 in favour of the East port. Similarly, if there is no packet from the Local and East input ports towards the South output port and at the same time, there is a packet from the West input port towards the South output port, then the condition C4 is satisfied. The priority changes to 33 in favour of the West input port. The other transitions in the priority generator of Fig. 3.7 can be explained in the similar fashion.

Round-robin Arbiter: The round-robin arbiter uses the priority set by the corresponding round-robin priority generator. The FSM states and transitions of round-robin arbiter R2ArbiterS are shown in Fig. 3.8(b). In round-robin arbiter, we use priority information generated by a priority generator as shown in Fig. 3.7. The priority values (*11, 22, 33*) indicate the packet from the input port that wins the arbitration. Therefore, the state of the



(a) Packet from R2 to R5

(b) Round–robin: R2ArbiterS

Figure 3.8: *R2ArbiterS (Round-robin arbiter at South port of Router R2)*

switch FSMs are not explicitly considered in the transitions for round-robin arbiter as they are considered during determining the priority. The FSMs for round-robin arbiter is similar with the FSM for fixed-priority arbiter in the remaining other aspects.. If the priority is set to *11*, a packet from L port gets a chance for transmission when buffer in the next router is free (condition C1 or C7 in Fig. 3.8). The new state of the R2ArbiterS becomes *1TL* or *2TL*. Here, the condition **C1** $= ((\text{R2PrS} = 11) \wedge (\text{R2ReL} = 0) \wedge (\text{R5BuN} = 0))$. The condition (R2PrS $= 11$) indicates that the priority is set for the packet from the Local input port. The condition (R2ReL $= 0$) indicates that the Local input port is ready and (R5BuN $= 0$) indicates that the corresponding buffer in the adjacent router R5 is empty. Therefore, the transition (Start $\rightarrow$ 1TL) takes place with an indication that the Local input port wins the arbitration. On satisfying the condition **C2** $= ((\text{R2ReL} \neq 0) \wedge (\text{R5BuN} = 1))$, the transition (1TL $\rightarrow$ Start) takes place. It means the R2ArbiterS returns to its initial state after transmitting a packet from the Local port. Here, the condition (R5BuN $= 1$) means that the buffer in the adjacent router R5 is updated from (R5BuN $= 0$) after receiving the packet. The condition (R2ReL $\neq 0$) means all the FSMs corresponding to this transmission has to return to their respective initial states. The other condition regarding the transfer of packet from the local port **C7** = (R2PrS = 11 $\wedge$ R2ReL = 0 $\wedge$ R5BuN = 1) is similar to **C1**. Only difference is, in case of **C1** the adjacent router buffer was empty and in case of **C7** the adjacent router buffer contains one packet and has capacity for storing one more packet. On satisfying **C7**, the transition (Start $\rightarrow$ 2TL) takes place. Here, the state 2TL means R5BufferN in the next router already contains one packet and can store one more packet. The state 1TL means R5BufferN in the next router contains no packet. In similar way, all other transitions in the round-robin arbiter in Fig. 3.8 are carried out.

3.4 Correctness of the Model

We have verified progress and synchronization between functional units for ensuring the correctness of our FSM based NoC model. Progress in a communication network ensures liveness of the system [33]. That means a system component should not stuck in a state and each component of the system is functioning. Satisfying progress in the proposed NoC model implies that current state of an FSM does not stuck permanently in a state, i.e., the state of an FSM keeps on changing provided that there presents a packet as input for transmission. In NoC context, simply satisfying progress property locally does not guarantee deadlock freedom. Satisfying progress ensures only the correctness of the system. For deadlock-freedom, global deadlock needs to be avoided in consideration with a specific routing algorithm.

3.4.1 Progress in Router Components

In this subsection we have presented the LTL specifications for the progress in the router components: *buffer*, *switch* and *arbiter*.

3.4.1.1 Progress in a Buffer

The change of states in a *buffer* is controlled by the state of the *sync* FSM. Initially both the *buffer* and *sync* FSMs are at their initial state 0. If the *sync* FSM changes its state to 01 with the transition ($0 \rightarrow 01$) in Fig. 3.3(b), the corresponding *buffer* also needs to change its state with the transition ($0 \rightarrow 1$) in Fig. 3.3(c) in the next cycle. Formally this property is written as, "If the current state of *sync* FSM is 01, the next state of the corresponding *buffer* changes to 1 in the next cycle". The Linear Temporal Logic (LTL) for this property is, `G((R2SyncS = 01)` $\Longrightarrow$ `X(R2BufferS = 1))`. Similarly, LTL specification for the progress property when another packet is stored in the buffer is, `G((R2SyncS = 02)` $\Longrightarrow$ `X(R2BufferS = 2))`. Satisfying these properties ensures the correctness of the buffer model.

3.4.1.2 Progress in a Switch

If there present at least one packet in a buffer, the corresponding switch should compute routing direction for transmitting the packet. Formally this property is written as, "If the input buffer is non-empty and the switch is currently in waiting(Wt) state, the switch state will eventually change to compute(C) state". The specification using LTL for this property is, `G(((R2BufferS != 0)` $\wedge$ `(R2SwitchS = Wt)` $\Longrightarrow$ `F((R2SwitchS = C))`. Similarly, "If switch is currently in compute(C) state, the switch state will eventually change to L or E or W based on the routing direction". The LTL representation is, `G((R2SwitchS = C)` $\Longrightarrow$ `F(R2SwitchS = L` $\vee$ `R2SwitchS = E` $\vee$ `R2SwitchS = W))`.

3.4.1.3 Progress in a Fixed-priority and Round-robin Arbiters

When a packet from an input port intends to get transmitted via a particular output port, the corresponding arbiter in that output port must acts to select the packet for transmission. Formally this property is written as, "If an arbiter is in initial state (Start) and there is at least one packet requesting that output port, the state of the arbiter will eventually change". The LTL representation is, `G((R2ArbiterS=Start)` $\wedge$ `((R2SwitchL = S)` $\vee$ `(R2SwitchE = S)` $\vee$ `(R2SwitchW = S))` $\Longrightarrow$ `F(R2ArbiterS != Start))`. In both the fixed-priority arbiter and round-robin arbiter, the progress does not ensure the starvation freedom. Since states are same for the fixed-priority arbiter in Fig. 3.6 and the round-robin arbiter in Fig. 3.8, same LTL specification is used for both the arbiters.

3.4.2 Synchronization within a Router

The synchronization within a router is controlled by a *return* FSM present at each input port. Initial state of *return* FSM is 0. If ($Return$!=0), it indicates that a packet is selected for transmission and the corresponding FSMs involved in that transmission must return to their respective initial states. Once the corresponding FSMs return to their initial states, the state of *return* FSM changes to ($Return$=0). A new transmission from that port starts only after that point. Correct execution of these transitions in order indicates proper synchronization inside a router. One synchronization property is, "When an input port packet is selected for transmission by an arbiter, the state of the *return* FSM corresponding to that input port changes from its initial state". Its LTL representation is, `G((R2ReturnE = 0)` $\wedge$ `(R2BufferE = 1)` $\wedge$ `(R2SwitchE = S)` $\wedge$ `(R2ArbiterS = (1TE` $\vee$ `2TE))` $\implies$ `X(R2ReturnE = 1S))`. After transmitting the packet from E input port via S output port in router R2, the E input port is not ready for a new packet until the corresponding *buffer*, *switch* and *arbiter* return to their initial states. After all these FSMs return to their initial states, the *return* FSM returns to its initial state ($R2ReturnE$=0) again, as shown in Fig. 3.5. Processing of new packets are enabled only when ($R2ReturnE$=0). This synchronization property can be expressed as, "After transmitting a packet from an input port if the corresponding buffer sets its buffer slot as free, corresponding switch and arbiter return to their initial states then the corresponding return FSM also returns to its initial state". In LTL, `G(((R2ReturnE=1S)` $\wedge$ `(R2BufferE = 0)` $\wedge$ `(R2SwitchE = Wt)` $\wedge$ `(R2ArbiterS = Start)` $\implies$ `X((R2ReturnE = 0))`. In similar way, synchronization between FSMs within a router for all ports can be represented using LTL properties for verification.

3.4.3 Correctness of a Priority Generator

The priority generated by a priority generator for a round-robin arbiter has to be checked if the priorities are generated correctly. Since no priority is used for the fixed-priority arbiter we have not checked this for the fixed-priority arbiter.

For proper functioning of the round-robin arbiter, the priority needs to be set eventually for each input port by the corresponding round-robin priority generator in Fig. 3.7. For example, in router R2, packets from L input port (priority *1*, *11*), E input port (priority *2*, *22*) and W input port (priority *3*, *33*) compete for S output port. If current state of the priority generator is *1* and a packet from W input port competes for S output port, eventually the priority should be set to *33* so that the packet from W input port gets a chance for transmission. This is represented in LTL as, `G((R2PriorityS = 1)` $\wedge$ `(R2SwitchW = S)` $\wedge$ `(R2ReturnW = 0)` $\implies$ `F(R2PriorityS = 33))`. Satisfying this LTL specification indicates correct functioning of the round-robin priority generator. Similarly, in router R5, if a packet from S input port is competing for the N output port, the priority for S input port competing for the N output port must set to *44* eventually. To verify this, its LTL specification is represented as, `G((R5PriorityN = 1 )` $\wedge$ `(R5SwitchS = N)` $\wedge$ `(R5ReturnS = 0)` $\implies$ `F(R5PriorityN = 44))`. Verification results for all such LTL specifications must be True for assuring the correctness in the design of round-robin priority generator.

3.5 Application of the Model

As applications of the presented FSM based NoC models, verification of starvation-freedom and transfer of packets across routers are presented in this section. We also discuss the challenges of extensive state space in a complete NoC and our approach to verify overall NoC considering the NoC in part-by-part.

3.5.1 Verification of Starvation-freedom

Starvation-freedom is defined as fairness in resource allocation between competing agents. Starvation at an output port of an NoC depends upon the underlying arbitration logic. As already described, two arbitration policies, fixed-priority and round-robin policies are considered in this work for arbitration. There may be more than one packets from different input ports that want to exit through the same output channel. If only one input port packet keeps on getting preference, the packets from other ports have to wait indefinitely and they suffer from starvation. Starvation-freedom in NoC context can be expressed as, "If a packet from an input port is competing for an output port, eventually the output port arbiter has to select that input port for transmitting the packet". For example, "If there is a packet from W input port intending to exit through S output port in router 2, the packet should get a chance to exit through S output port in future." The LTL representation of the same is, `G((R2ArbiterS=start)` $\wedge$ `(R2SwitchW = S)` $\implies$ `F((R2ArbiterS = 1TW)` $\vee$ `(R2ArbiterS = 2TW))`. By satisfying this property, it indicates that every packets from W input port intending S output port will be transmitted in future. If this property fails, it indicates that there are some possible scenarios where packets from W input port to S output port never gets transmitted. In router R1 of Fig. 3.1, R1ArbiterE accepts packets from L and S input ports. The LTL representation of starvation-freedom for S port at R1ArbiterE is, `G(((R1ArbiterE = Start)` $\wedge$ `(R1SwitchS = E))` $\implies$ `F((R1ArbiterE = 1TS)` $\vee$ `(R1ArbiterE = 2TS))`. It means, globally when the FSM R1ArbiterE is in state *Start* and FSM R1SwitchS is in state *E*, eventually the state of R1ArbiterE will be *1TS* or *2TS*. The arbiter state *1TS* or *2TS* indicates that the arbiter has selected a packet from S input port. By satisfying this property, the starvation-freedom for packets from S input port is ensured at R1ArbiterE. In a similar way, the starvation-freedom for any input port at a given output port is represented using LTL properties.

3.5.2 Verification of Transfer of Packets

The correct synchronization between two routers implies the transfer of packets between two routers in a proper way. A desirable property for a buffer is, "If a buffer is empty and the corresponding arbiter in the adjacent router has selected a packet for transmission, the state of the buffer changes eventually to store the packet." The LTL representation of the same is, `G((R5ArbiterN != Start)` $\wedge$ `(R2BufferS = 0)` $\wedge$ `(R2ReturnS = 0)` $\implies$ `F(R2BufferS = 1))`. The same property can be expressed as, "An arbiter is ready for transmitting a packet and the corresponding buffer in the next router is free for accommodating a packet, eventually the packet is stored in the next router buffer". Thus, the LTL representation for transfer of packets between two routers are presented for each pairs of connected routers in an NoC.

3.5.3 Verification of Overall NoC

Considering all the routers in a complete NoC along with the respective router components results in extensive state space. In this section we discuss the state space for our FSM based NoC model by considering the number of FSMs needed for the complete NoC model. We have also presented the active window concept where a part of NoC is considered instead of considering the complete NoC at a time.

3.5.3.1 Number of FSMs in an NoC

The number of active ports in an NoC router depends upon its position. All ports may not be active in an NoC router. Considering the 3x3 Mesh NoC in Fig. 3.1(a), for all the corner routers namely R1, R3, R7 and R9 has three ports active. Two ports connect its two neighbour and another port is for its local core. There present 4 such corner routers in a Mesh NoC of any size. The router R2 in Fig. 3.1(a) has three neighbours. Therefore, router R2 has four active ports, three ports connecting each neighbour and another port connecting the local core. We get (N-2)*4 such routers having four active ports in a NxN Mesh NoC. The router R5 in Fig. 3.1(a) has four neighbours. Therefore, all the five ports

Table 3.2: *Number of FSMs in an NoC with Fixed-priority (FP) and Round-robin (RR) arbiter*

NoC	**Number of FSMs considering Fixed-priority (FP) arbiter**			**Total FSMs**
	Active ports = 3 $a = (R_3 * FP_3)$	**Active ports = 4** $b = (R_4 * FP_4)$	**Active ports = 5** $c = (R_5 * FP_5)$	$(a+b+c)$
2x2	4 * 15 = 60	0 * 20 = 0	0 * 25 = 0	60
3x3	4 * 15 = 60	4 * 20 = 80	1 * 25 = 25	165
4x4	4 * 15 = 60	8 * 20 = 160	4 * 25 = 100	320
5x5	4 * 15 = 60	12 * 20 = 240	9 * 25 = 225	525
6x6	4 * 15 = 60	16 * 20 = 320	16 * 25 = 400	780
7x7	4 * 15 = 60	20 * 20 = 400	25 * 25 = 625	1085
8x8	4 * 15 = 60	24 * 20 = 480	36 * 25 = 900	1440
NoC	**Number of FSMs considering Round-robin (RR) arbiter**			**Total FSMs**
	Active ports = 3 $a = (R_3 * RR_3)$	**Active ports = 4** $b = (R_4 * RR_4)$	**Active ports = 5** $c = (R_5 * RR_5)$	$(a+b+c)$
2x2	4 * 18 = 72	0 * 24 = 0	0 * 30 = 0	72
3x3	4 * 18 = 72	4 * 24 = 96	1 * 30 = 30	198
4x4	4 * 18 = 72	8 * 24 = 192	4 * 30 = 120	384
5x5	4 * 18 = 72	12 * 24 = 288	9 * 30 = 270	630
6x6	4 * 18 = 72	16 * 24 = 384	16 * 30 = 480	936
7x7	4 * 18 = 72	20 * 24 = 480	25 * 30 = 750	1302
8x8	4 * 18 = 72	24 * 24 = 576	36 * 30 = 1080	1728

active for router R5. There are (N-2)*(N-2) such routers with five active ports in a N*N Mesh NoC.

In our FSM model for a router with fixed-priority arbiter we need 5 FSMs corresponding to each port of that router. These five FSMs are namely switch, buffer, sync, return and fixed-priority arbiter. If a router with fixed-priority arbiter has three active port, we need 15 ($3 * 5 = 15$) such FSMs to model the router. Therefore, we consider $FP_3 = 15$ in Table 3.2. Similarly, for a router having fixed-priority arbiter with 4 active ports, $FP_4 = 20$ $(4 * 5)$ and with 5 active ports, $FP_5 = 25$ $(5 * 5)$. These values are used in Table 3.2 for calculating the number of FSMs in a Mesh NoC model. If we consider round-robin arbiter in the router, one additional FSM is needed for priority generation. Therefore, for a router with round-robin arbiter we need 6 FSMs corresponding to each port of that router. If a router with round-robin arbiter has three active port, we need 18 ($3 * 6 = 18$) such FSMs to model the router. Therefore, we consider $RR_3 = 18$ in Table 3.2. Similarly, for a router having round-robin arbiter with 4 active ports, $RR_4 = 24$ ($4 * 6$) and with 5 active ports, $RR_5 = 30$ ($5 * 6$). The total number of FSMs needed while designing a complete NoC using fixed-priority arbiter and round-robin arbiter is shown in Table 3.2. The number of FSMs needed for modeling a 8x8 Mesh NoC is 1440 considering FP arbiter. This number is even more while considering round-robin arbiter. For 8x8 Mesh NoC we need 1728 FSMs. The number of states in each FSM is multiplied while a complete NoC is encoded using a model checker. It gives the total state space for a complete NoC. State space for a complete NoC is a too high for achieving scalabilty using model checker. State space for even for a 2x2 NoC is $2^{138.24}$ while using fixed-priority arbiter and is $2^{169.2}$ while using round-robin Arbiter. These numbers explode with the increase of NoC sizes. Therefore, we have considered the active windows next for each NoC router instead of considering a complete NoC at a time.

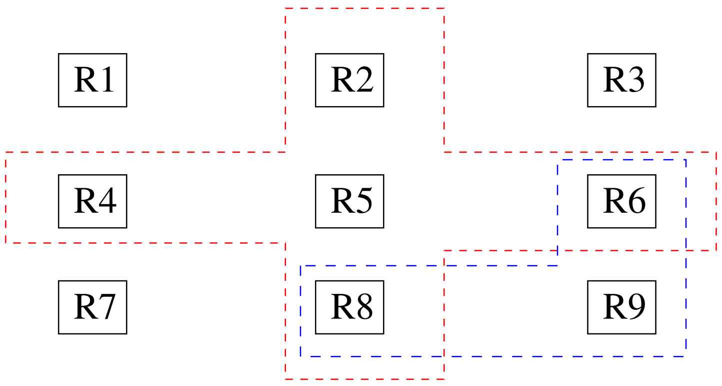


Figure 3.9: *Partitioning NoC: Active Windows for the Router R5 and R9*

3.5.3.2 Active Windows

Due to the state space explosion problem a complete NoC system along with its detailed components cannot be encoded with a state-of-the-art model checker. Therefore, we partition the NoC into active regions and perform verification for starvation and transfer of packets in each active region. Router R9 interact with its two neighbours R6 and R8. Therefore, we consider router {R6, R8, R9} at a time as the active window for R9, as shown in Fig. 3.9. Similarly, active window for R5 is {R2, R4, R5, R6, R8} and for R8 is {R5, R7, R8, R9} and so on. The communication between two routers are present in a active window. Therefore, properties involving two routers can be checked using active windows. All these active windows can be executed using parallel threads to save verification time. To verify global properties like deadlock and livelock, we need to consider the complete NoC at a time. Even active window would not be able to facilitate that. Therefore, global properties like deadlock and livelock can not be verified due to state space explosion even by using active windows.

3.6 Experimental Results and Analysis

We have created all FSM models manually from 2x2 to 8x8 Mesh NoC. The FSM models are encoded in NuSMV [22] for verification. Individually executing each router for verification of progress, synchronization, proper priority generation for round-robin arbiter, transfer of packets and starvation are time consuming operations. In this section, experimental results and analysis for all the experiments are presented. In all the experiments, an Intel Xenon(R) 2.10GHz X 32 processor, 64 GB RAM machine is used.

3.6.1 Verification of Progress, Synchronization and Priority Generation within a Router

In this experiment, we have used all routers in an NoC individually and execute them both serially and in parallel threads. We have verified the progress in NoC router components, synchronization within a router and correctness in priority generation using the LTL specifications as described in Section 3.4. Experimental results show that synchronization properties within a router and progress within a router are satisfied to be True for all routers. The LTL specification for priority generation for a round-robin arbiter also satisfied in all the experiments.

Table 3.3: *Verification of progress, synchronization and priority with fixed-priority (FP) and round-robin (RR) arbiter (A) within individual routers*

Mesh	Serial Exe. (H:M:S)		Parallel Exe. (H:M:S)		Seep Up	
NoC	FP A.	RR A.	FP A.	RR A.	FP A.	RR A.
2x2	0.0.2	0.0.8	0.0.2	0.0.7	1x	1.1x
3x3	1.20.02	3.38.16	1.10.09	2.26.34	1.1x	1.3x
4x4	4.0.17	11.36.23	1.17.09	2.35.16	3.1x	4.5x
5x5	8.02.11	23.54.51	1.26.07	2.41.02	5.6x	8.9x
6x6	13.20.31	40.32.56	1.36.27	2.58.21	8.3x	13.7x
7x7	20.12.03	61.30.42	2.05.03	3.57.06	9.6x	15.6x
8x8	28.02.16	86.48.02	2.38.56	4.49.27	10.6x	18.0x

3.6.1.1 Runtime Improvement with Parallel Execution considering Individual Router

The execution time for the experiments is shown in Table 3.3. State-space increases due to the implementation of dynamic priority in round-robin arbiter as compared to fixed-priority arbiter. Therefore, verification time for NoC with a round-robin arbiter increases significantly than that of NoC with a fixed-priority arbiter. Table 3.3 shows that verification time increases with the increase of NoC sizes as well. In case of larger NoC, the number of routers increases. Therefore, verification time increases with the increase of NoC grid size. The verification time needed for both the serial and parallel execution along with speed ups are shown in Table 3.3. The speed up is calculated as the ratio of serial execution time to the parallel execution time. The experimental results show that using parallel threads speed ups the verification process for individual router up to 18x times over its equivalent serial execution for bigger NoCs.

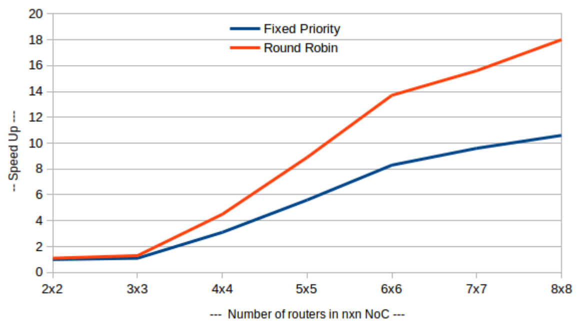


Figure 3.10: *Speed up with the increase in the number of routers*

We put the speed up trend for parallel execution of individual routers in Fig. 3.10. Though, theoretically we could expect the speedup proportional to the router numbers i.e., n^2 for an nxn NoC, practically, we are getting lesser than that. Even the speed up trend is not the same for both the Fixed Priority and Round Robin arbiter. Besides the number of routers, other factors like thread overhead, operating system details, and memory requirements for the verification also affect the speed up. The speed up trend continues until the memory is not saturated.

3.6.2 Verification of Transfer of Packets and Starvation Freedom considering Active Windows

For verifying the transfer of packets, we need to consider an arbiter and the corresponding buffer in the adjacent router, i.e., the connected port of adjacent router. Similarly, for verification of starvation, we need to consider the arbiter which communicates with the buffer of adjacent router. For considering the communication with an adjacent router, we have considered active windows in this experiment. Example for active windows are shown in Fig. 3.9. We have used active window corresponding to each router in an NoC. All active windows are executed using parallel threads for saving the verification time.

Successful transfer of packets indicates correct synchronization between an arbiter and a buffer in the adjacent router as well. The LTL specification that are used for the verification of transfer of packets are presented in Section 3.5. All the specifications for transfer of packets corresponding to each pair of arbiter and connected buffer are verified to be True in all the experiments. We have verified starvation freedom as well in this experiment. The verification results of starvation-freedom for router R5 in a 3x3 Mesh NoC (Fig. 3.1) are shown in Table. 3.4. We get similar results for all the other routers up to 8x8 Mesh NoC. Some ports are inactive (W and N ports in R1 of Fig. 3.1) and starvation-freedom is not applicable (NA) for some other ports. Such ports in router R5 with respect to different arbiters are denoted as “not applicable (NA)” in Table. 3.4. In router R5 (Fig. 3.1), packets from L, W, N and S input ports compete for E output port (R5ArbiterE). Incoming packets from E input ports do not compete for E output port. Therefore, starvation-freedom at R5ArbiterE from E input port is not applicable and is denoted as NA in Table 3.4. For packets from L, W, N and S input ports, whether starvation-freedom specification is satisfied or not is denoted by T (True) or F (False).

Table 3.4: *Starvation-freedom for fixed-priority (FP) and round-robin (RR) arbiters*

Packets from input ports	**Starvation-freedom at the output port of an Arbiter**									
	R5ArL		**R5ArE**		**R5ArW**		**R5ArN**		**R5ArS**	
	FP	**RR**	**FP**	**RR**	**FP**	**RR**	**FP**	**RR**	**FP**	**RR**
L	NA	NA	T	T	T	T	T	T	T	T
E	T	T	NA	NA	T	T	T	T	T	T
W	T	T	T	T	NA	NA	F	F	F	T
N	F	T	F	T	F	T	NA	NA	F	T
S	F	T	F	T	F	T	F	F	NA	NA

3.6.2.1 Analysis of the Findings on Starvation Freedom

For fixed priority arbiter, it seems obvious that starvation-freedom is satisfied only for highest priority port. On the other hand, experimental results in Table. 3.4 shows that *starvation-freedom is satisfied for both highest and second highest priority ports.* We consider the priority order as L > E > W > N > S, i.e., packets from L and S input ports have the highest and the lowest priority, respectively. At the N output port of router R5 (R5ArbiterN), Table. 3.4 shows that W and S input port suffer from starvation whereas L and E input ports are free from starvation. The reason behind these findings are analysed from the the debug trace. Debug trace shows that there are continuous packets from L, E, W, S input ports competing for the N output port (R5ArbiterN). The packet from the L input port (highest priority port) is transferred to the next router. There present other consecutive packets from the L port. After transmitting a packet from an input port there involves some delay in performing synchronization with the buffer (clearing buffer) and switch using the return FSM, that are present in the same router. On the other hand, the packet from the E input port (next highest priority port) is ready and is waiting for the same N output port.

Therefore, the packet from E input port gets transmitted. The synchronization latency in the highest priority port is the reason for satisfaction of starvation-freedom property for the second-highest priority port as well in fixed-priority arbiter. If synchronization latency can be nullified in a model, packets only from the highest priority port, i.e., the Local port in this case, get chance for transmission. We believe that such delay would present in the actual hardware as well unless it is nullified using extra resources like extra buffers. By the time the E input port packet is transferred, the L input port is synchronized and a packet from L input port gets a chance for the next transmission. In this way, packets only from L and E input port get transmitted alternatively though there are continuous packets from L, E, W and S ports. Packets from W and S input ports suffer from starvation. On the other hand, round-robin arbiter is a dynamic arbitration policy. The priority is dynamically changing by the priority FSM shown in Fig. 3.7. Therefore, no input port suffers from starvation in round-robin arbiter as shown in Table. 3.4. Experimental results reinforce that our design and implementation are correct.

Table 3.5: *Verification time for transfer of packets and starvation freedom considering fixed-priority (FP) and round-robin (RR) arbiter (A) in Active Windows*

NoC	Time (H:M:S) for Fixed-priority (FP) and Round-robin (RR) Arbiter					
	Serial execution (H:M:S)		Parallel execution (H:M:S)		Seep Up	
	FP A.	RR A.	FP A.	RR A.	FP A.	RR A.
2x2	0.0.4	0.0.56	0.0.2	0.0.24	2x	2.3x
3x3	7.33.04	70.02.56	6.23.09	38.03.21	1.2x	1.8x
4x4	26.12.04	234.08.56	10.01.13	53.50.09	2.6x	4.4x
5x5	57.57.04	492.18.56	10.21.05	90.45.12	5.6x	5.4x
6x6	102.08.04	844.32.56	11.09.17	152.03.51	9.1x	5.5x
7x7	158.45.04	1290.50.56	13.10.03	226.01.07	12x	5.7x
8x8	227.48.04	1836.57.56	16.58.10	315.12.32	13.4x	5.8x

3.6.2.2 Runtime Improvement with Parallel Execution for the Active Windows

The execution time for transfer of packets and starvation verification using active window is shown in Table. 3.5. Similar to the previous experiments considering individual routers, NoC with a round-robin arbitration policy takes more time than that of NoC with a fixed-priority arbitration policy. Experimental results in Table 3.5 shows that verification time increases with the increase of NoC sizes as well. In case of smaller NoC, the number of routers are less. Moreover, corner routers and boundary routers have less number of active ports. For example, the router R1 in Fig. 3.1 has three active ports (L, E and S). But for the middle router like R5, all five ports are active. Larger NoC has more number of such routers where all ports are active. Therefore, verification time increases with the increase of NoC grid size. *In case of parallel execution, execution time is significantly saved for NoC of higher grid size. Therefore, the parallelization of the verification process is very effective in the verification of starvation. As shown in Table 3.5, running starvation verification for fixed-priority arbiter using parallel threads speed ups the verification process up to 13.4x times over serial execution for bigger NoCs.* This runtime improvement for round-robin arbiter using parallel thread is 5.8x times over serial execution. The improvement is not significant for round-robin arbiter after a 5x5 NoC. Because the round-robin arbiter demands more state space.

3.7 Conclusion

In this chapter, we have presented formal modeling NoC components using FSM by considering NoC components in detail. Synchronization between different functional units is taken care for error-free functioning of the overall system. We have verified the correctness of the model by verifying progress property and synchronization between NoC components. Transfer of packets between routers is also verified. Verification of starvation-freedom using fixed-priority policy and round-robin policy are demonstrated in this chapter. Due to state space explosion problem, it is practically not feasible to verify globally dependent properties like deadlock using a detailed NoC model with help of model checker. Since formal verification is a time consuming procedure, we have used thread level parallelism for verifying routers individually and verifying active windows corresponding to each router in an NoC in our experiments. The properties that are internal to a router are verified considering individual router. Active windows are used for verifying properties that involve communication between two routers. Experimental results show a significant improvement in verification time for thread level parallelism over its equivalent serial execution.

4

Formal Modeling of NoC using CFSM and Developing a Simulation Framework for Deadlock Detection

4.1 Introduction

A deadlock scenario in an NoC involves resource dependency that might spread across the complete NoC. Therefore, formal model of the complete NoC is needed to be considered to detect deadlock. Whereas, it is not feasible to verify a complete NoC with detailed modeling due to unmanageable state space. Encoding of a huge system like the complete NoC is not supported by the state-of-the-art model checkers due to the state space explosion problem. On the other hand, while considering the state-of-the-art NoC simulators, they give only warning message about possible deadlock, which does not guarantee a real deadlock situation. Therefore, a simulation framework on a formally modelled NoC which can detect confirm deadlock is promising. Here, formal model ensures correctness in detecting deadlock and the simulation on that formal model detects deadlock specific to an application.

Maintaining synchronization between all NoC components during simulation is a difficult task. A large number of synchronization FSMs are needed to be implemented explicitly for this purpose. Therefore, our FSM based model presented in the previous chapter may not be suitable for developing formal simulator. In contrast, in Communicating Finite State Machines (CFSMs), synchronization between functional units are maintained automatically with help of message passing. No dedicated CFSMs for synchronization are required. Therefore, we use CFSMs for developing formal simulator of NoCs. A formal modeling of NoC using CFSM is presented first. We have automated the CFSM model generation for NoC with Mesh and Torus topologies. We have then presented the CFSM based formal simulator.

4.2 Contributions

Contributions of this chapter are classified into two categories:
a) Formal modeling of NoC using CFSM and automation of NoC model generation.
b) Developing a simulation framework using the generated NoC model to be used for application specific deadlock detection.

4.2.1 Formal Modeling of NoCs using CFSM

The formal method for performance analysis and study of the reachability of packets in NoCs is a computationally complex task. So, scalable formal models of the NoC components which support parallel execution are needed. Moreover, analysis of properties like deadlock, starvation, livelock require detailed level modeling. CFSM is one such formal model, proposed by Daniel Brand and Pitro Zafiropulo from IBM Research Laboratory in 1983 [40]. This model can be used to represent distributed computing, parallel processing, computer network, and communication protocols. In such systems, a huge number of functional units work in parallel. The CFSM models these units as processes and the communication between them are captured by sequence of sending and receiving operations.

The same approach can be applied to NoC, where router components are the processes and communication between them takes place using the packets. Therefore, in this work, we use CFSM for modeling the NoC components. To the best of our knowledge, detailed formal modeling of NoCs is not explored enough in the literature (discussed in the Chapter 2). There are some abstract level models of NoCs in the literature [80], [31], [50] that do not capture the detailed behaviour of the NoC architecture. Specifically, the contributions of this work on modeling NoCs are as follows:

- This work demonstrates an approach for formal modeling of Mesh and Torus topologies using CFSMs. A similar approach can be followed for designing other NoC topologies as well.
- We formulate the naming convention of each component of the CFSM model so that each component is uniquely identifiable.
- It is error-prone to create formal models of NoC manually, which requires a large number of CFSMs for its representation. Therefore, a programming interface is developed to automatically generate the formal CFSM models of Mesh and Torus based NoCs of any size.
- The condition required for deadlock in CFSM based model of NoCs is formally captured in Theorem 4.5.2. Moreover, we formally show the bounded communication of our CFSM models using Lemma 4.6.1 and Lemma 4.6.2.
- The proposed CFSM based model of NoCs is generic in terms of routing algorithms, i.e., any routing algorithm can be symbolically simulated on it.

4.2.2 Development of CFSM based Simulation Framework

We further develop a formal simulator based on our CFSM model. As an application of our CFSM based simulator, we focus on detecting deadlock of a routing algorithm for a given application in the next phase of the work. Several techniques on deadlock avoidance like restriction on packets from taking certain turns [9], increasing the buffer size and the number of virtual channels [103] etc. have been proposed. Most of these schemes require additional overhead in terms of system resources, area, and power consumption. Further, such avoidance techniques are designed to prevent deadlocks for a given routing algorithm. A routing algorithm may have deadlock for an NoC topology. However, the same routing algorithm may not reach in deadlock state for a given set of applications in the same NoC topology. If deadlock avoidance techniques can be designed to work only for a given set of applications the NoC currently executes, system resource overhead can be minimized. To elaborate, if application specific deadlock detector is designed, it can be used to enable or disable deadlock avoidance scheme based on the requirement. In this work, we aim at the development of an application specific deadlock detector for NoC using our CFSM based models.

Detection of deadlock in an NoC is a challenging task, due to the prohibitive state space required to model the NoC and the routing algorithms therein. Booksim2.0 [24] is a cycle accurate NoC simulator that simulates the NoC components including routers in parallel. This simulator uses a threshold number of cycles to determine deadlock. If the number of cycles exceeds the threshold and yet packets remain to be delivered, simulation is aborted giving warning message for possible deadlock. Gem5 [25] is another widely used full system simulator that uses a similar philosophy for deadlock detection. It internally uses Garnet [26], an interconnection network model, for NoC simulation. It may be noted that since cyclic dependency for the resource is not checked in Booksim or in Gem5, it does not give a guarantee that the scenario, in fact, is a deadlock. The threshold value may overshoot due to other circumstances like failure, livelock or starvation.

The second contribution of this work is an application specific simulation framework using our CFSM based NoC models. Our framework accurately detects the occurrence of a deadlock situation for a given routing algorithm and an input traffic sequence. Specifically, the contributions of the second part of the work are as follows:

- A CFSM based simulation framework is developed to detect confirmed deadlock sce-

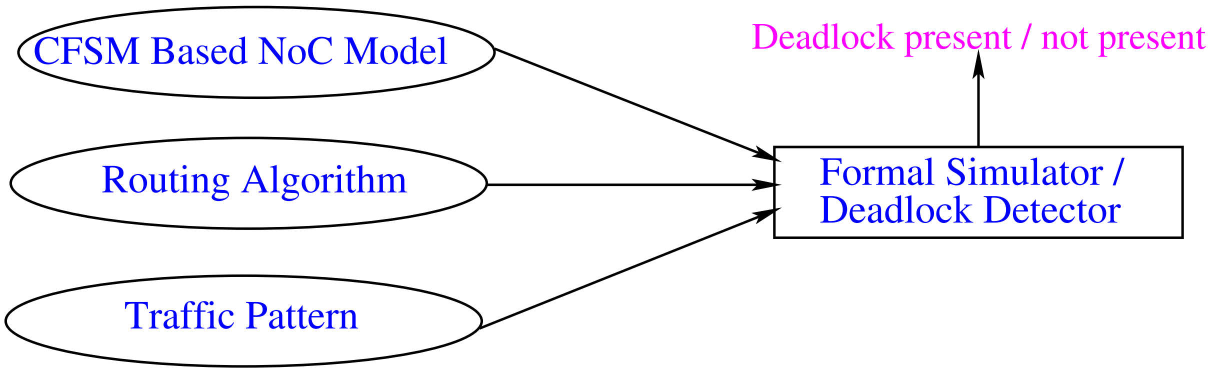

CFSM Based NoC Model
Routing Algorithm
Traffic Pattern
Formal Simulator / Deadlock Detector
Deadlock present / not present

Figure 4.1: *Deadlock Detection using CFSM based NoC Model*

nario in an NoC on a given traffic pattern for a given routing algorithm. The overview of the simulator is given in Fig. 4.1.

- We have tested our framework for three routing algorithms namely, XY routing (static) and Dynamic-XY routing [8] (adaptive) and a modified version of West-First routing (adaptive) [9] on Mesh and Torus NoCs. The experimental results are presented for various network sizes and for various traffic patterns. Our CFSM based framework identifies false-positive deadlock warnings of the Booksim simulator.

The rest of the chapter is organized as follows. Section 4.3 covers the background theory of CFSMs. Section 4.4 demonstrates the modeling of NoC using CFSM. Our proposed approach to NoC deadlock detection and representation of cyclic dependencies using CFSM are described in Section 4.5. Automated NoC model generation and computational feasibility of our models are described in Section 4.6. Experimental results and their analysis are presented in Section 4.7. Finally, we conclude the chapter in Section 5.7.

4.3 Background of Communicating Finite State Machine based Modeling

A communicating finite state machine (CFSM) is an automata with directed labelled graph having two types of edges, namely sending and receiving edges [75]. Each node in a CFSM represents the current state of the CFSM and has at least one outgoing edge. For each message, there is a sending edge where the message is generated and there is a corresponding receiving edge where the message is consumed. For a message m, the sending edge (message generation) and the receiving edge (message consumption) are labelled as $-m$ and $+m$, respectively. The $-m$ and $+m$ messages are part of two different CFSMs, i.e., a message generated by one CFSM is consumed by another CFSM. If each outgoing edge of a CFSM has distinct label, then the CFSM is called *deterministic*; otherwise, the CFSM is called *nondeterministic* [77].

Each component of the communicating system is represented as a process with the help of a finite state machine. All the processes can communicate with each other by passing messages. To store a message temporarily before being received by a process, a message queue is used between each pair of communicating processes. The processes are synchronized by blocking read of the message queues. Unless an expected message is generated by a process and is stored in a specific message queue, the receiving process cannot make a transition by receiving that expected message. For each pair of communicating processes, massage sending and massage receiving take place with the help of two dedicated message queues. Message queues, that are virtually present between a pair of communicating CFSMs, are different from the buffer, that are used to store packets in NoC. When a message m is transmitted from a CFSM Px at a state Gen, there should be an transition labeled $-m$ from the state Gen (for generating). Similarly, when a message m is received by an another CFSM Py at a state Rcv, there should be an transition labeled $+m$ from the state Rcv (for receiving). If a CFSM is in a *receiving state*, it cannot make progress until it receives desired message from the corresponding CFSM.

Example 4.1. Fig. 4.2 shows an example of three communicating processes, namely P1, P2 and P3 with their initial states $Gen1$, $Rcv2$ and $Rcv3$, respectively. P1 sends a message to P3 via P2. On receiving the message, P3 sends back an acknowledgment to P1 via P2. In this example, all the messages with prefix "$(+/-)a$" act as acknowledgments. P1 generates a message $-m1$ to indicate a specific event. This message is received as $+m1$ by P2 in the state $Rcv2$. This event triggers the generation of another message $-m2$ at state $Tr2$

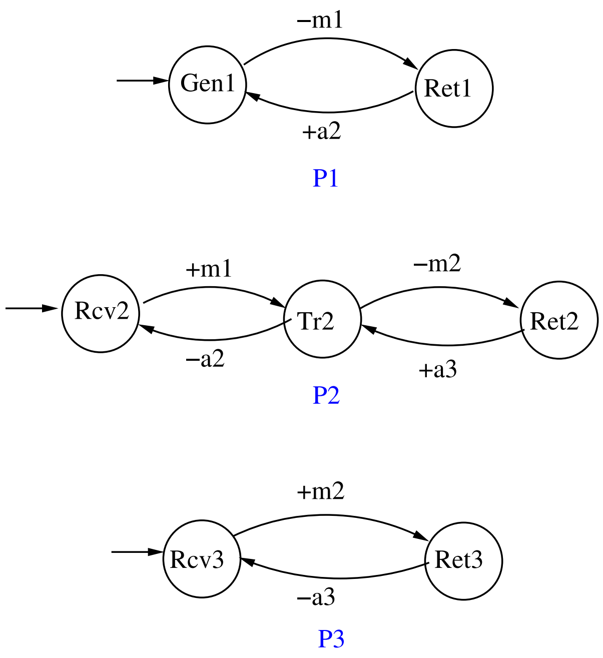


Figure 4.2: *Three CFSM processes communicating with each other*

(transmit) of P2. This message is received by P3 as $+m2$ in the state $Rcv3$. P3 generates a message $-a3$ making a transition from $Ret3$ (return) to its initial state $Rcv3$. This message is consumed at P2 in the state $Ret2$ and makes a transition from the state $Ret2$ to $Tr2$. P2 then generates message $-a2$ and makes a transition from its current state $Tr2$ to its initial state $Rcv2$. By consuming $+a2$ in the state $Ret1$, P1 makes a transition to its initial state $Gen1$. After these cycles of messages P1, P2 and P3 again go back to their respective initial states.

The formal definition of a CFSM is presented as follows [40].

Definition 4.3.1. *A communication system in CFSM is represented using a quadruple, $\{(S_i)_{i=1}^n, (I_i)_{i=1}^n, (M_{ij})_{i,j=1}^n, succ\}$. Here, n is a positive integer representing the number of processes. S_i represents the set of states of process P_i. I_i is a member of S_i that represents the initial state of process P_i. $(M_{ij})_{i,j=1}^n$ are n^2 disjoint finite sets. M_{ij} represents the messages that can be sent from P_i to P_j. M_{ii} is empty for all P_i. The succ is a partial function mapping for each P_i and P_j, $S_i \times M_{ij} \rightarrow S_i$ and $S_i \times M_{ji} \rightarrow S_i$. The $succ(s,x)$ indicates the next state after the transition when the process transmits or receives message x in the state s, i.e., if $t = succ(s,x)$, the current state s of P_i is changed to t, after it* *transmits (/receives) a message x to (/from) the process P_j.*

Example 4.2. We represent the CFSM in Fig. 4.2 using the formal notation of the CFSM definition where $n = 3$, $S_1 = \{Gen1,\ Ret1\}$, $S_2 = \{Rcv2,\ Tr2,\ Ret2\}$, $S_3 = \{Rcv3,\ Ret3\}$, $I_1 = Gen1,\ I_2 = Rcv2,\ I_3 = Rcv3$, $M_{12} = \{m1\},\ M_{21} = \{a2\}$, $M_{23} = \{m2\},\ M_{32} = \{a3\}$, $succ(Gen1,\ m1) = Ret1,\ \ succ(Tr2,\ a2) = Rcv2$ and so on.

An execution of a CFSM based system is represented as sequence of global states. The formal definition of global state in a CFSM execution is follows [40].

Definition 4.3.2. *A global state is a pair [S, Q]. Here, S is an n-tuple of states* $\{s_1, s_2, s_3, \ldots, s_n\}$*, where* s_i *is a state of process* P_i*. Q is an* n^2*-tuple* $\{q_{11}, q_{12},\ \ldots, q_{1n}, q_{21}, q_{22},\ \ldots, q_{2n}, \ldots, q_{n1}, q_{n2},\ \ldots, q_{nn}\}$*, where each* q_{ij} *is a sequence of messages from* M_{ij}*, i.e.,* $q_{ij} \subseteq M_{ij}$*.*

The sequence of messages q_{ij} in the channel from process P_i to process P_j are stored in a message queue Q_{ij}. Process P_j can consume messages from P_i present in the message queue Q_{ij}, in first-in-first-out (FIFO) basis.

Example 4.3. With this notation, the initial global state of the system in Fig. 4.2 is expressed as $G_0 = [\ Gen1,\ Rcv2,\ Rcv3,\ Q_{12} = \{\phi\},\ Q_{21} = \{\phi\},\ Q_{23} = \{\phi\},\ Q_{32} = \{\phi\}\ ]$. After $m1$ is generated by P1, the new global state is expressed as $G_1 = [\ Ret1,\ Rcv2,\ Rcv3,\ Q_{12} = \{m1\},\ Q_{21} = \{\phi\},\ Q_{23} = \{\phi\},\ Q_{32} = \{\phi\}\ ]$. In this way, we express the execution of a system in terms of a sequence of global states.

4.4 Formal Modeling of NoC using CFSM

In this section, we model major NoC components like buffer, switch, arbiter and scheduler using CFSMs. For identifying different CFSMs and different messages by unique names in our CFSM based model, we first present the *naming convention* of each component in this section.

4.4.1 Naming Convention

The CFSM model presented in this section are with reference to the functional unit of the 2x2 Mesh NoC shown in Fig. 4.3(a). We consider a unique number for each router. To save space and for reading convenience, we are describing the names of functional units using short forms. For example, the North port of router R3 in Fig. 4.3(a) is indicated as N port of R3. The arbiter present at the South port of the router R1 is denoted by ArbiterS1. Similarly, the buffer present at the North port of the router R3 is indicated by BufferN3. We use the similar naming convention for describing our CFSM based NoC model.

There are large number of messages on a CFSM based model of an NoC. To keep the message names unique and meaningful, we follow a naming convention. Formation of all types of messages used in our CFSM based NoC model are presented with examples in Table 4.1. In this table, three symbols namely Rn, Pn and Bn are used to represent message formation structure. Here, Rn indicates the router number, Pn indicates the port name (L, E, W, N, S), and Bn indicates the buffer slot number. Each message ends with router number (Rn) preceded with port name (Pn), i.e., $PnRn$. It indicates the associated port name and associated router number of the message. We maintain unique name and unique number for each CFSM while generating the CFSM model. Even though we have not followed strict naming convention for the CFSM states, we can distinguish each CFSM state uniquely with help of unique CFSM name and CFSM number. Purpose and meaning of different message types are given in short in Table 4.1. More detailed information are elaborated while describing different CFSM models.

4.4.2 Modeling Buffer

Buffer is used as a temporary storage for a packet at the input port of a router. We present CFSM models for one buffer with single slot and another buffer with three slots in this subsection.

Table 4.1: *Naming convention for Messages in CFSM*

Prefix	**Formation**	**Example**	**Meaning**
A (Arbiter)	*PnRn*	AS1	Indication of packet transmission from South port arbiter at router 1 to the next router.
B (Buffer)	*PnRn*	BN3	North port buffer of router 3 is free to receive a packet. This is indicated to the connected adjacent router.
B (Buffer)	*BnPnRn*	B2N3	Indicates that a packet presents at the North port 2nd buffer slot of Router 3 is moving to the next buffer slot.
BT (Buffer Transmit)	*PnRn*	BTN3	North port buffer of router 3 requests connected switch for start processing a packet for transmission.
comp (Compute)	*PnRn*	compN3	Request for route computation (RC). North port switch of router 3 requests corresponding RC unit.
C (Compute)	*PnPnRn*	CEN3	RC result is determined. North port RC unit of router 3 calculates output port to be E (East) and inform the switch.
SS (Switch Scheduler)	*PnPnRn*	SSNE3	Message to scheduler. North port switch of router 3 requests East output port scheduler of the same router to generate priority.
S (Switch)	*PnPnRn*	SNE3	Message to arbiter. North port switch of router 3 requests East port arbiter of the same router for transmitting a packet from North input port.
priority (0, 1, 2, 3)	*PnRn*	0S1	South port scheduler of router 1 sends priority value 0 to South port arbiter of the same router.
ret (Return)	*PnRn*	retN3	North port switch of router 3 requests the last buffer slot at North port to return to its initial state.
ret (Return)	*PnPnRn*	retSE1	South port Arbiter of router 1 requests the switch at East port of router 1 to return to its initial state.
retB (Return Buffer)	*BnPnRn*	retB3N3	A packet is transferred from 2nd buffer slot to 3rd buffer slot of North port of router 3. The 3rd buffer slot inform the 2nd buffer slot at the same port to return to its initial state using this return message.
retComp (Return Compute)	*PnRn*	retCompN3	North port switch at router 3 requests route computation unit at North port to return to its initial state.

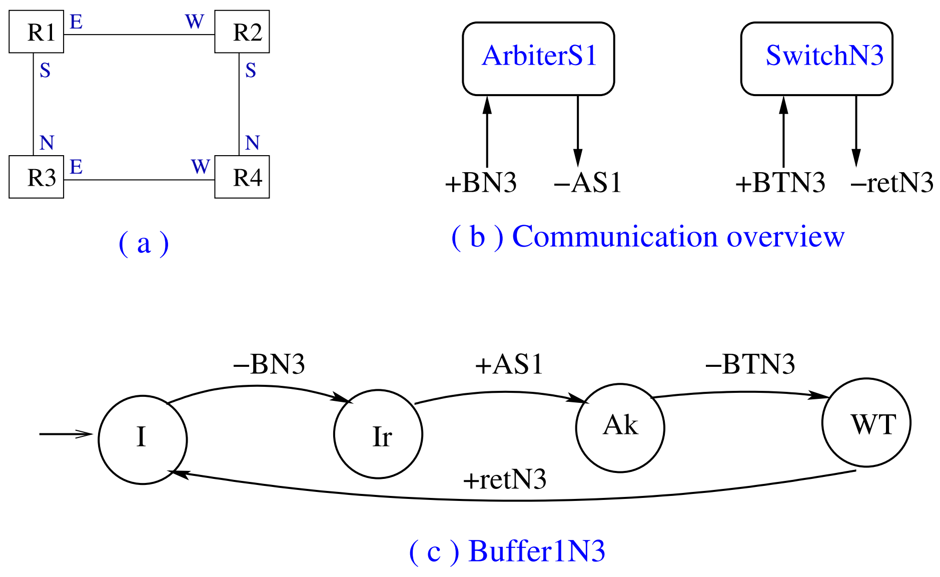


Figure 4.3: *Buffer with single slot: (a)* 2x2 *NoC as a reference for modeling (b) Overall communication for buffer at the North port of router R3 (c) CFSM model for buffer with capacity=1*

4.4.2.1 Buffer with single slot

A single slot buffer (Buffer1N3) is shown in Fig. 4.3(c). It is present at the North port of router R3, shown in the 2x2 NoC in Fig. 4.3(a). The communication overview of Buffer1N3 is shown in Fig. 4.3(b). The Buffer1N3 is connected from the arbiter at the South port of router R1 (ArbiterS1) and receives packet from that arbiter. The buffer communicates with the North port switch, i.e., SwitchN3, for route computation. If Buffer1N3 is free, it sends $-BN3$ message to ArbiterS1 indicating that buffer is free for receiving a packet. If router R1 needs to send a packet to router R3, ArbiterS1 sends $-AS1$ message to Buffer1N3 (Fig. 4.3(b)). This message is consumed as $+AS1$ by Buffer1N3 and the new state becomes Ak (acknowledgment), as shown in Fig. 4.3(c). From the state Ak, it generates message $-BTN3$ towards SwitchN3, to process the packet in switch for transmission. The new state of Buffer1N3 becomes WT, where it waits for SwitchN3 until it does the necessary operation for the packet and sends a message $+retN3$.

4.4.2.2 Buffer with more than one slots

For modeling buffer with more than one slots we need CFSMs corresponding to each buffer slot. Transferring packet from one slot to another slot are modeled using CFSMs.

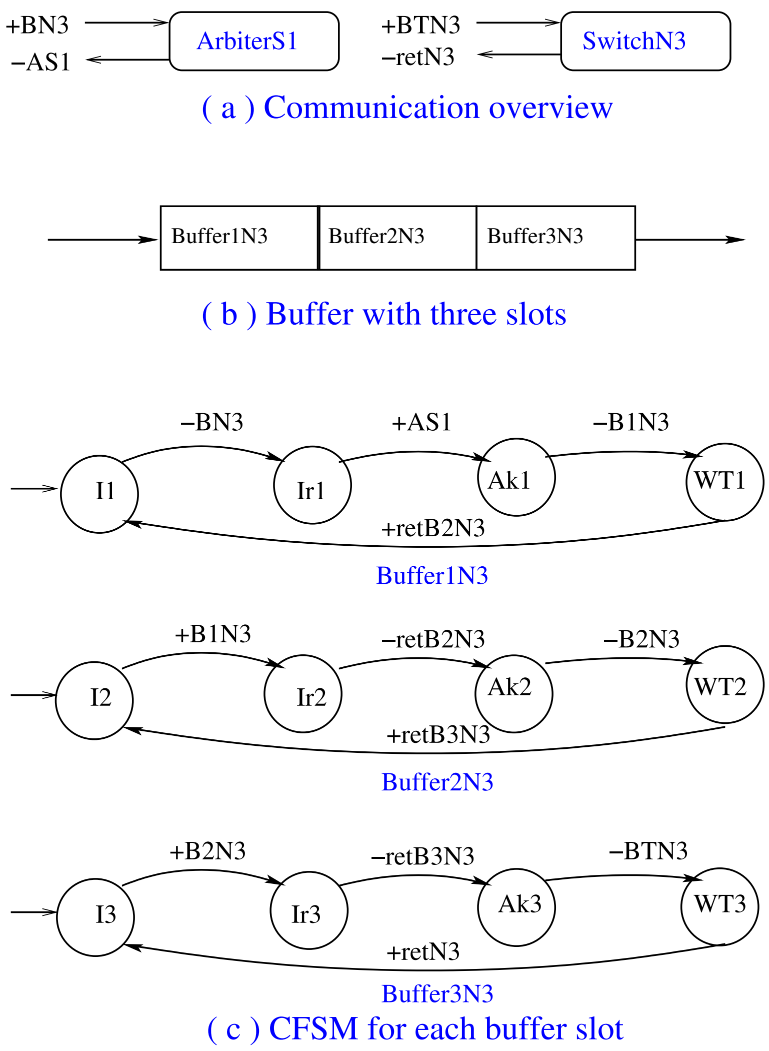


Figure 4.4: *Buffer with three slots: (a) Overall communication for buffer at the North port of router R3 (b) Three buffer slots as a FIFO (c) CFSM model for buffer with three slots*

Communication of a buffer with its corresponding NoC components and a three slots buffer are shown in Fig. 4.4(a) and Fig. 4.4(b), respectively. Modeling of the buffer with three slots, at the North port of router R3, is shown in Fig. 4.4(c). Three CFSMs, namely Buffer1N3, Buffer2N3 and Buffer3N3, are used for this purpose. Each CFSM in Fig. 4.4(c) represents a buffer slot from the Fig. 4.4(b). Since the buffer is a FIFO, a packet is first placed at the first slot from the adjacent router (ArbiterS1) and is transmitted to the switch of that port (SwitchN3) from the third slot. The packet is moved to slot three through the intermediate buffer slot, i.e., second slot in this case. In the CFSM model of Fig. 4.4(c), at first, a packet enters Buffer1N3. It sends the necessary synchronization message to ArbiterS1 and forwards the packet to Buffer2N3, as shown in Fig. 4.4(c). If Buffer2N3 is free (in state

$I2$), it sends acknowledgment $-retB2N3$ to Buffer1N3 and passes the packet to Buffer3N3 (by sending message $-B2N3$). In case, Buffer2N3 is not free (not in state $I2$), that means it is holding some other packet and has to wait for processing of a new packet. If Buffer3N3 is free and gets a new packet (indicated by message $+B2N3$), it requests SwitchN3 to process the packet by sending $-BTN3$ message and waits in the state $WT3$. All the transitions in a buffer for storing and forwarding packets are self explanatory from Fig. 4.4(c). Using the same approach, we can model a buffer of capacity n with help of n such CFSMs.

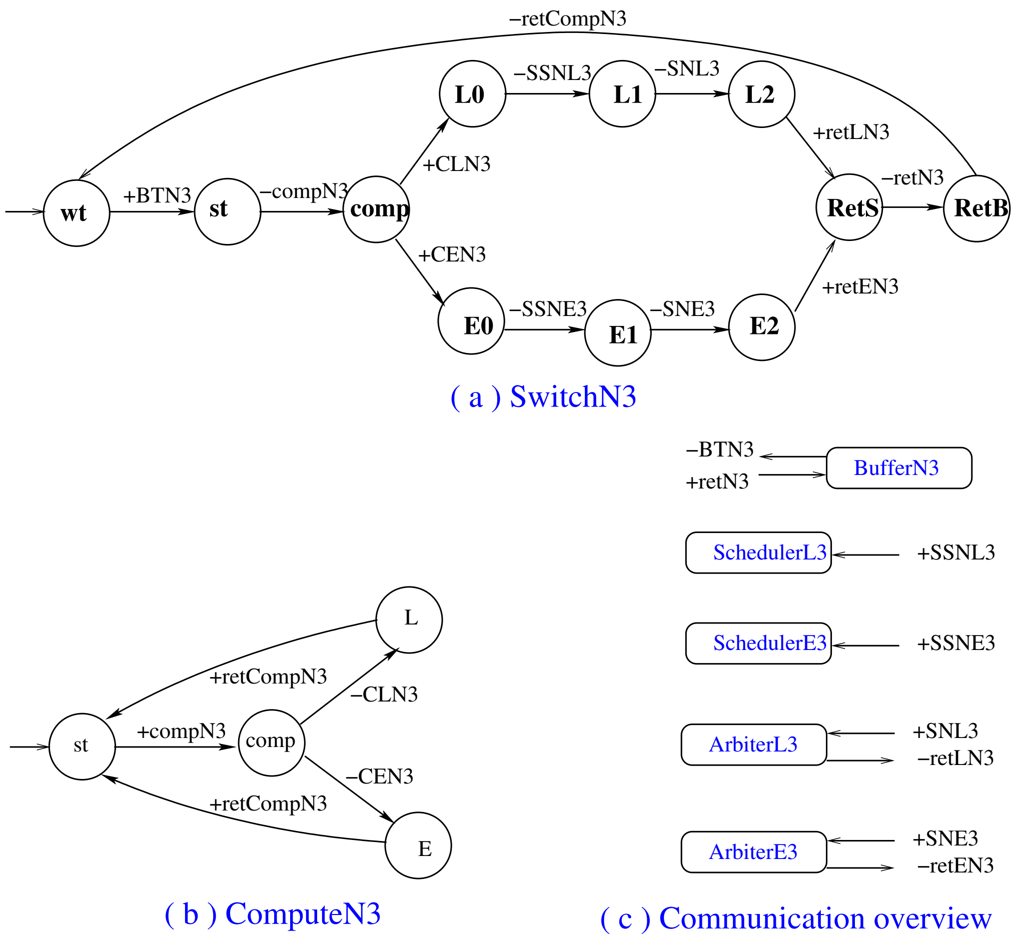


Figure 4.5: *CFSM model for a switch: (a) Switch at the North port of router R3 (b) Route computation (c) Communication overview for SwitchN3 with other NoC components*

4.4.3 Modeling Switch and Route Computation

A switch considers a packet stored in the input port buffer for route computation. Route computation decides the output port through which the packet has to transmit. If the packet has already reached its destination router it is deliver to the local port. Modeling of a switch, a route computation unit and the overall communication with a switch is shown in Fig. 4.5(a), Fig. 4.5(b) and Fig. 4.5(c), respectively. A packet is directed to a switch from a buffer at the input port of a router. Switch gets the routing information for the packet from another CFSM (compute machine) and forwards the packet to the arbiter present at the intended output port. The CFSM representation of SwitchN3 is shown in Fig. 4.5(a). Buffer1N3 sends $-BTN3$ message to the SwitchN3 as shown in Fig. 4.3. By consuming $+BTN3$ message, the switch moves from waiting state (wt) to starting state (st) and starts processing the packet. SwitchN3 sends $-compN3$ message to ComputeN3, where routing algorithm is implemented. ComputeN3 is shown in Fig. 4.5(b). Routing algorithm determines the direction of output port through which data packet should traverse to reach the destination router. The route computation is performed in compute machine and route information is passed to switch by sending $-CLN3$ or $-CEN3$ message. If the packet reaches its destination router, $-CLN3$ message is sent to SwitchN3. If the routing algorithm under consideration decides the packet has to move towards the East, $-CEN3$ message is transmitted to SwitchN3. After consuming $+CEN3$, a message $-SSNE3$ is sent to SchedulerE3 requesting scheduler at the East port of router R3 to set priority for the packet. Another message $-SNE3$ is sent towards ArbiterE3 requesting the East output port for transmission. Once output port is granted, SwitchN3 receives an acknowledgment message $+retEN3$ from ArbiterE3. After consuming this message, SwitchN3 sends acknowledgment message $-retN3$ towards Buffer1N3 of Fig. 4.3. Finally, SwitchN3 returns to its initial state wt by sending $-retCompN3$ message towards ComputeN3. By consuming this message, ComputeN3 also returns to its initial state st. SwitchN3 and ComputeN3 wait for a new packet in their initial states wt and st, respectively.

Referring to the 2x2 NoC in Fig. 4.3(a), it may be noted that the North port switch of router R3 (SwitchN3) can send packets only in two ways: either towards the local port of router R3 or towards router R4 via the East output port. In generic case, for a router with four neighbours in a 3x3 NoC or NoC of higher sizes, all the ports will be active and the packet can move in four directions. In such case, the *comp* state of the switch will have four outgoing edges or four outgoing transitions. Thus, the number of outgoing edges for a switch depends upon the number of neighbours or number of active ports. A router can have two or three or four neighbours depending upon its position in a Mesh or Torus NoC. In our NoC model, we modelled all such types of routers. For sake of simplicity in representation, we present CFSM model with reference to a 2x2 NoC shown in Fig. 4.3(a).

4.4.4 Modeling Arbiter and Scheduler

An arbiter is present at the output port of a router and resolves the conflict if multiple packets are requesting the same output port at the same time. Switch sends a packet towards an appropriate output port based on the decision of the routing algorithm. At each output port, an arbiter is present. Multiple packets may compete for the same output port at the same time. These conflicts are resolved by an arbiter with the help of a scheduling policy, like round-robin policy, first-come-first-serve policy, priority based policy, etc.. In this work, we use round-robin policy. The CFSM representations of ArbiterS1 along with SchedulerS1 (round-robin scheduler) at the South port of router R1 (refer Fig. 4.3(a)) is shown in Fig. 4.6(a) and Fig. 4.6(b), respectively. The communication overview for ArbiterS1 is shown in Fig. 4.6(c). ArbiterS1 accepts packets from SwitchL1 and SwitchE1 based on the priority set by SchedulerS1. The two states L and E in ArbiterS1 are corresponding to incoming packets from SwitchL1 and SwitchE1, respectively. In SchedulerS1, state named as '0' indicates priority 0 and a packet from SwitchL1 gets preference. State named as '1' indicates priority 1 and a packet from SwitchE1 gets preference. Before setting priority, the scheduler goes through some intermediate states (00, 11, 01 and 10) as shown in Fig. 4.6.

4.4.4.1 Changing of priority in round-robin fashion

Here we describe how the priority keeps on changing so that packets from both the switches, i.e., SwitchL1 and SwitchE1, get chance in round-robin fashion in Fig. 4.6(b). Initially, ArbiterS1 and SchedulerS1 are in their initial states L and 0, respectively. State 0 in

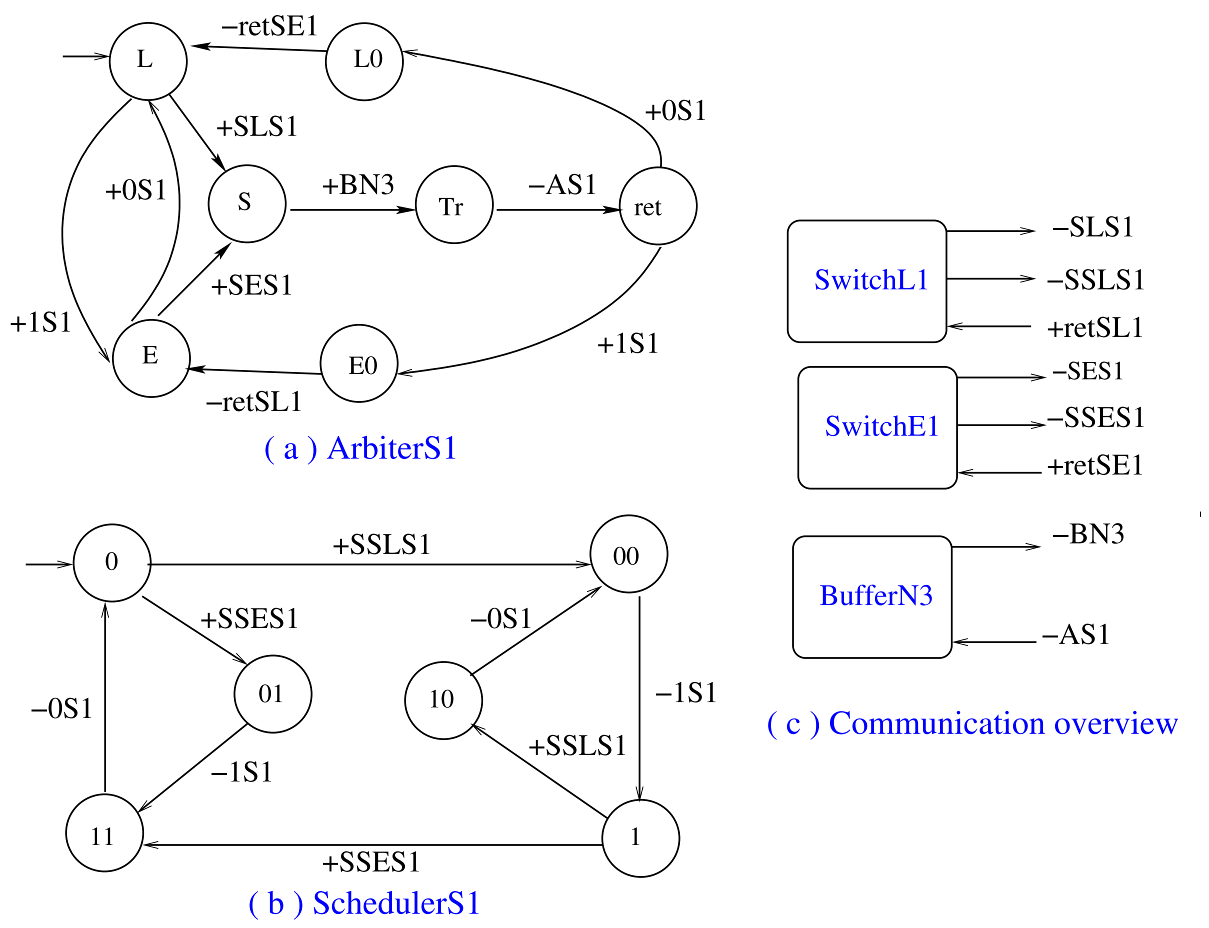


Figure 4.6: *Arbiter and Scheduler at the South Port of Router R1: (a) CFSM model for ArbiterS1, (b) CFSM model for SchedulerS1, (c) Communication overview for ArbiterS1*

scheduler means, packet from SwitchL1 gets first preference in the arbiter. Similarly, state 1 in scheduler means packet from SwitchE1 gets first preference in the arbiter. Therefore, if the current state of the scheduler is 0 and both $+SSLS1$ and $+SSES1$ messages are available for consumption, $+SSLS1$ is consumed and the scheduler moves to state 00. A node in a CFSM has computation power to decide as like deciding the routing path for a switch. In deterministic CFSM, using of same message for transition into two different states are not permitted. Since each outgoing edge of the CFSM has distinct label, the CFSM is *deterministic* [77]. The message $+SSLS1$ is generated by SwitchL1 to make the priority keep on changing so that packets from other ports get chance for transmission. SchedulerS1 generates $-1S1$ for ArbiterS1 and moves to state 1. At this point, one observation from the switch in Fig. 4.5(a) is that $+SSLS1$ and $+SLS1$ are generated consecutively by a switch. Therefore, $+SSLS1$ is available for consumption by SchedulerS1 means $+SLS1$ is also available for consumption by the corresponding ArbiterS1 in the next cycle. The current state of ArbiterS1 is L. Since $+SLS1$ is available for consumption, ArbiterS1 consumes this message and moves to the state S.

4.4.4.2 Transmitting the packet from current router to the next router

The transfer of packets to the next router takes place if the buffer in the next router is free. If the corresponding buffer in the next router is full, the arbiter has to wait till the buffer becomes free. Our CFSM based model of the arbiter takes care of all such situations. ArbiterS1 waits for the buffer (Buffer1N3) in the next router at the state S if the buffer is full. Buffer1N3 of router R3 (Fig. 4.3(a)) generates $-BN3$, if it has free slot to receive a packet. At this state, ArbiterS1 consumes $+BN3$ and moves to the state Tr as shown in Fig. 4.6(a). After coming to the state Tr, ArbiterS1 of router R1 generates $-AS1$ message towards Buffer1N3 of router R3 and moves to the state ret. The message $-AS1$ indicates that ArbiterS1 sends a packet towards Buffer1N3. Another message $+1S1$ is already waiting for consumption by ArbiterS1. By consuming this message, it moves to the state $E0$. ArbiterS1 generates $-retSL1$ from the state $E0$, and returns to state E. At this state, packet from SwitchE1 gets higher priority in ArbiterS1. Thus, packets from all input ports get a chance for transmission in a round robin fashion.

4.4.5 Modeling Virtual Channel

In an NoC router with Virtual Channel (VC), a single physical channel is shared by a number of VCs. Adding virtual channels do not increase bandwidth to the physical channel. A separate buffer is allocated for each virtual channel [103]. Organization of VCs in an input port, as described in [103], is shown in Fig. 4.7. In this diagram, four packets are present in a buffer. The buffer, shown in Fig. 4.7(a), is associated with a single VC. The alphabet in each buffer slot indicates the output port, the packet is intended to move. In this mode, one packet may block rest of the packets behind it. For the example in Fig. 4.7(a), if the

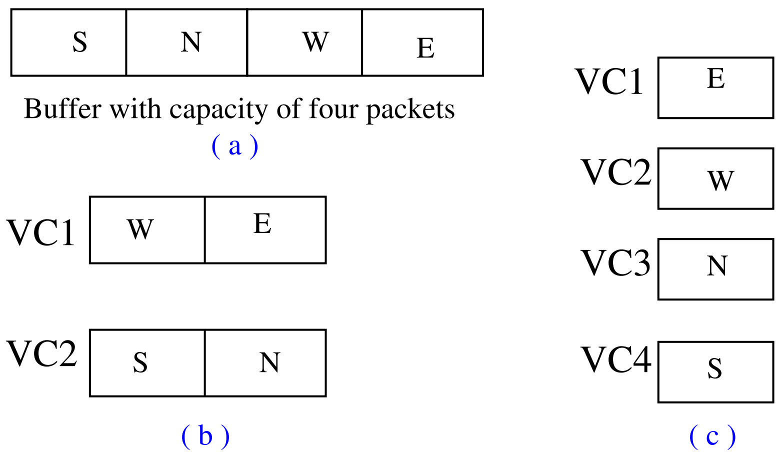


Figure 4.7: *Virtual channel: (a) Buffer with four slots, (b) Buffer restructuring for two virtual channels, (c) Buffer restructuring for four virtual channels*

first packet is blocked, other subsequent packets in the same buffer are also blocked. For implementing two VCs, the same buffer of capacity four packets is organized as two separate buffer each with a capacity for two packets, as shown in Fig. 4.7(b). In this arrangement one more packet gets chance to move towards North port. Similarly, it can be organized as four VCs by splitting into four separate buffers each with capacity one, as shown in Fig. 4.7(c). In this case, all four packets get a chance to compete for their desired output ports. Thus VC reduces the chance of blocking one packet by another packet.

If there are two VCs, i.e., VC1 and VC2, in an input port, we have to store the buffering information and routing information for both VCs distinctly. Therefore we need separate Buffer (Fig. 4.3), Switch and Compute (Fig. 4.5) for each VC. Thus, more numbers of CFSMs are added into the CFSM based NoC model for handling VCs. The increase of CFSMs in NoC model with the increase of VCs is reflected in Table 4.2. For keeping all the CFSMs unique, we need minor modifications, to the NoC models described in the Section 4.4. Suppose the number of VCs is two. Therefore we need two Buffer1N3 shown in Fig. 4.3(b). For distinguishing the CFSMs, we append VC number (_V) to the CFSM name. Thus we get two buffers with name Buffer1N3_1 and Buffer1N3_2, if there are two VCs. Similarly, we need to distinguish each message with its corresponding VC. To do so, each message in Buffer1N3_1 and Buffer1N3_2 are modified by appending VC number _1 and _2 , respectively. In general, by adding VC number (_V) to the corresponding machine name

Table 4.2: *Number of CFSMs in Mesh NoC*

NoC	VC Size = 1			VC Size = 2		
Size	#VC=1	#VC=2	#VC=3	#VC=1	#VC=2	#VC=3
2x2	60	96	132	68	112	156
3x3	165	264	363	189	312	435
4x4	320	512	704	368	608	848
5x5	525	840	1155	605	1000	1395
6x6	780	1248	1716	900	1488	2076
7x7	1085	1736	2387	1253	2072	2891
8x8	1440	2304	3168	1664	2752	3840

and each message of the corresponding Buffer, Switch and Compute, we keep all CFSMs unique. Arbiter and Scheduler (Fig. 4.6) control the physical link between routers. Since adding virtual channels do not increase the physical link from one router to another router, the number of Arbiters and Schedulers are not required to be increased with increase of VCs. Arbiter and Scheduler identify different VCs by looking at the VC number (_V) attached after each message. They send the corresponding message to the appropriate CFSM by appending the correct VC number (_V).

4.5 Proposed Scheme for Deadlock Detection

In an NoC, when more than one packet is competing for the same port, there involves waiting time for gaining access to the buffer in the desired port. If the waiting time is finite, we term this as *delayed reception* in CFSM representation. In this section, we present Lemma 4.5.1 on delayed reception. We also define deadlock situation in CFSM with respect to NoC. In this section, we also show that our deadlock definition actually represents a situation of cyclic dependency using Theorem 4.5.2.

4.5.1 Delayed Reception

For a system modeled using CFSM, if a message $+m1$ arrives at a process P1 whose current state is S_m but the system model does not specify any transition for consuming message $+m1$ in state S_m, such a situation is called *unspecified reception* [40]. Unspecified reception is considered as design error. On the other hand, for a complex system, it may be possible that P1 is waiting for another message $+m2$ from another process P2, receiving which, P1 makes a transition to state S_n. At the state S_n, transition for consumption of the message $+m1$ is defined. We are considering this as delayed reception instead of considering it as design error. The behavior of NoC allows delay in receiving a message. To represent this situation, in our CFSM based NoC model, certain scenario appears where a message gets consumed only after the consumption of other messages with a finite delay. If the state S_n is reachable from the state S_m with a number of finite and feasible transitions, eventually the message $+m1$ will be consumed by process P_1. This observation is stated as Lemma 4.5.1.

Lemma 4.5.1. *Let a message x from a CFSM P_i arrives at a CFSM P_j, when P_j is in state S_m. Even though no transition is specified for receiving x in state S_m, it is possible to reach a future state S_n (at P_j) from the state S_m in finite time, where transition for receiving x at P_j is specified. This situation is termed as **delayed reception**.*

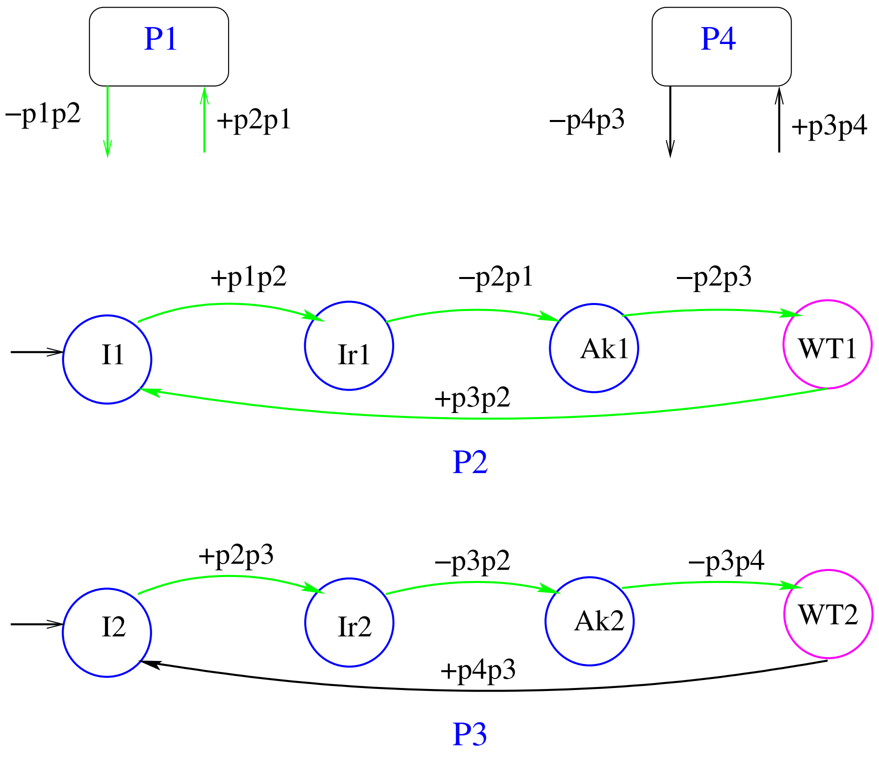


Figure 4.8: *An example of delayed reception*

Example 4.4. Let assume that processes that P1 and P4 in Fig. 4.8 have many states and transitions. All transitions of P1 and P4 are not shown to keep the example simple. The blue states and green edges indicate the visited states and edges, respectively. The magenta states are the last visited states in this example. The black edges indicate the edges not visited yet. P2 receives a message $+p1p2$ from P1 and moves from initial state $I1$ to intermediate state $Ir1$. To acknowledge the consumption of $+p1p2$, it makes another transition by generating the message $-p2p1$ towards P1. P2 makes a transition from state $Ak1$ (acknowledged state) to waiting state $WT1$ by generating the message $-p2p3$ towards P3. P3 consumes this message and acknowledges it by sending -p3p2 towards P2. Consuming this, P2 returns to its initial state $I1$ and P3 is in acknowledged state $Ak2$. P3 generates a message $-p3p4$ towards P4 and moves to state $WT2$ and waits for acknowledgment message $+p4p3$ from P4. Let us assume, P4 is busy in doing other operations and there is a delay in consuming $+p3p4$ and returning of $-p4p3$ towards P3. By this time, similar transitions are repeated in P1 and P2. Let P2 be in state $WT1$ by transmitting message $-p2p3$ towards P3. P3 is in state $WT2$ and cannot consume $+p2p3$ (from P2) until $+p4p3$ has arrived (from P4). This situation is an example of delayed reception where transition is possible with a delay (i.e., once $+p4p3$ is received and P3 moves to initial state $I2$).

The important point is to be noted that the transition is possible at process P_j from state S_m after a finite time period in case of delayed reception. However, no transition is possible within finite time period in case of a deadlock situation. We define the deadlock in CFSMs next.

4.5.2 Representation of Deadlock in NoC using CFSM

A transition in a CFSM based system is possible iff a message generation or message consumption is possible. Gouda et al. [77] describe deadlock as a reachable global state $[S, Q]$. A global state $[S, Q]$ is said to be a deadlock state iff *i*) all states $\in S$, are in receiving state, and *ii*) all message queues $\in Q$, are empty. Since all states are in receiving state, a transition due to the generation of a new message is not possible. A transition due to the consumption of a new message is also not possible due to the absence of any message in the message queue. *In a system, modeled using CFSM, if neither message generation nor message consumption is possible, it indicates no transition is possible in the system.* Therefore, no further transition is possible from a deadlock state of a system. The deadlock description in [77] has not considered the possibility of delayed reception. In this work, we have defined the deadlock in a CFSM based network model where delayed reception is permitted.

Definition 4.5.1. *Consider a CFSM model of a system with n number of processes, where delayed reception is permitted. A reachable global state $[S, Q]$ is said to be a deadlock state iff i) All states $\in S$, are in receiving state and ii) Either all message queues are empty or some other message queues are not empty but no further transition is possible.*

The condition, "All message queues are empty"- is already mentioned in the deadlock definition in [77]. The other case, where some message queues are not empty becomes applicable due to presence of delayed reception. Let us consider each process P_i, where $i = 1, 2, 3, \ \ldots, n$ and the current state of P_i is s_i. The state s_i is a receiving state and waiting for message from some process P_j, i.e., from message queue Q_{ji}. The message queues from where P_i receives messages are Q_{ki}, where $k = 1, 2, 3, \ \ldots, j, \ \ldots, n$ and $j \neq i$. If Q_{ji} is empty even though some other message queues in Q_{ki} are not empty, the transition from the current state s_i of process P_i cannot consume any message from any other message queue Q_{ki}. In such a situation, no transition is possible from any process P_i and the global state $[S, Q]$ is a deadlock state of the CFSMs.

Example 4.5. We have illustrated deadlock in a CFSM model with delayed reception in Fig. 4.9. The color of the states and the edges have the same meaning as mentioned in the previous example in Fig. 4.8. There are six processes in this example. Initially, all processes are in the respective initial states, represented by the global state G_0. A possible sequence of global state transitions for these CFSMs are shown in Fig. 4.10. Suppose P1 transmits message $-p1p3$ towards P3. At the same moment, P4 also transmits message $-p4p6$ towards P6. The new global state is G_1. After consuming these messages by P3 and P6, global state changes to G_2. In this way, there is a possible sequence of global state

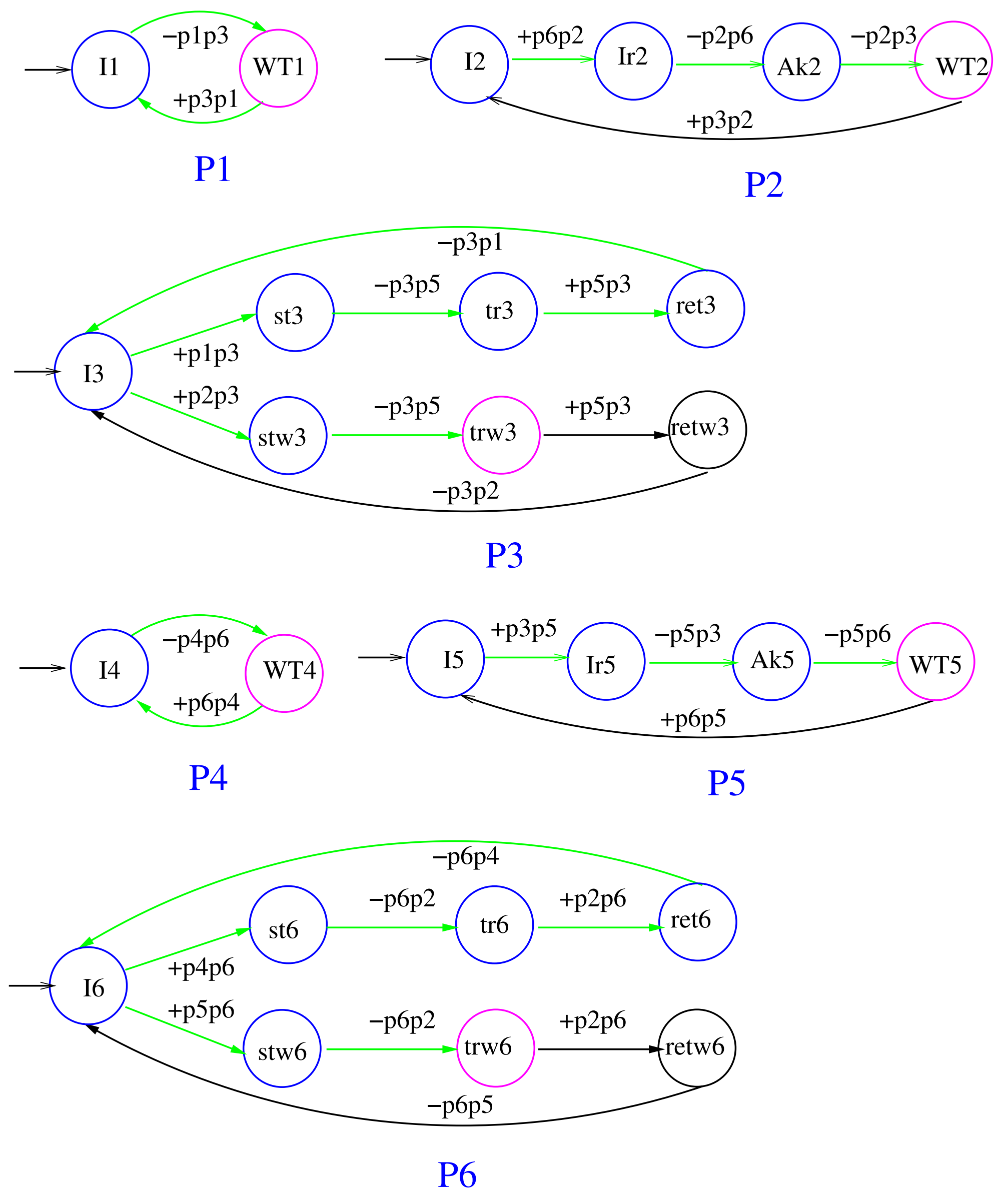


Figure 4.9: *CFSMs with Cyclic Dependency*

transitions that are illustrated in Fig. 4.10. Finally, the system reaches a global state G_f: $[WT1, WT2, trw3, WT4, WT5, trw6,\ Q_{13} = \{p1p3\},\ Q_{35} = \{p3p5\},\ Q_{46} = \{p4p6\},\ Q_{62} = \{p6p2\}\ ]$. From the global state G_f, all the processes are in receiving states and no further transition is possible although some of the message queues are not empty. If we observe carefully, it reveals that processes P2, P3, P5 and P6 are waiting for each other and they form a cycle of dependency. The dependency graph is shown in Fig. 4.11. In this graph, the current state of each CFSM and the message for which the CFSM is waiting is shown. There is no further transition possible after reaching global state G_f, due to cyclic dependency

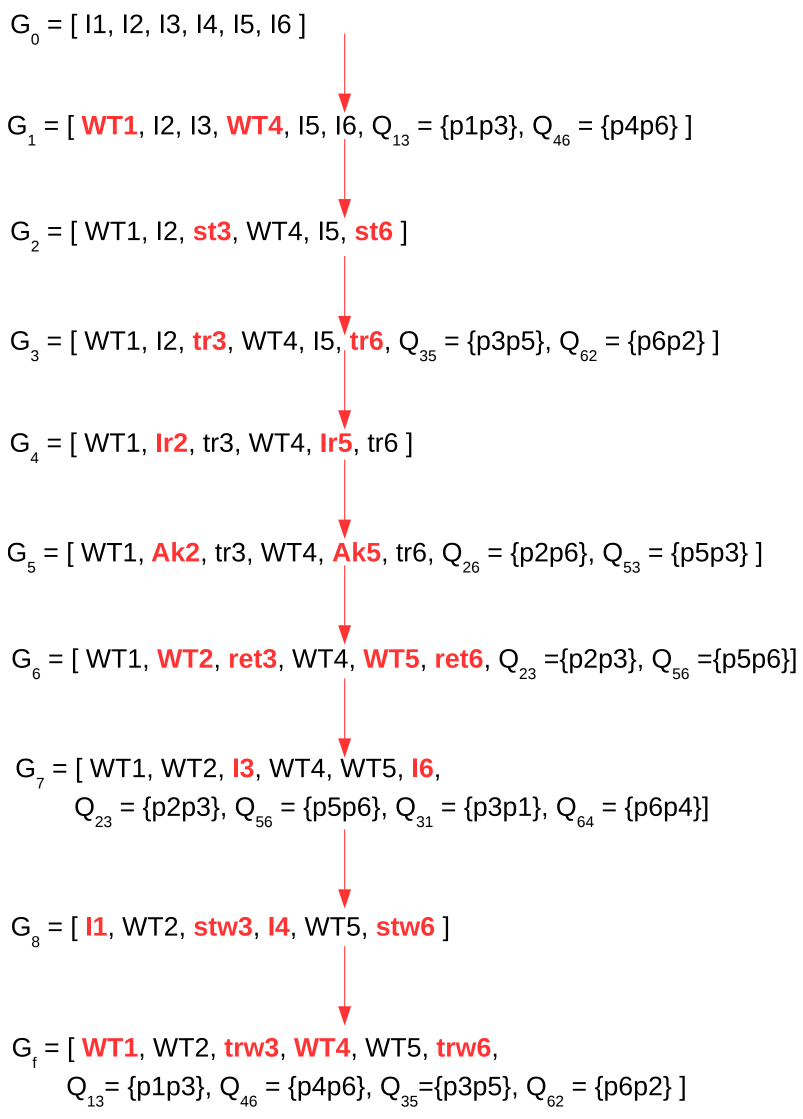

G0 = [I1, I2, I3, I4, I5, I6]
G1 = [WT1, I2, I3, WT4, I5, I6, Q13 = {p1p3}, Q46 = {p4p6}]
G2 = [WT1, I2, st3, WT4, I5, st6]
G3 = [WT1, I2, tr3, WT4, I5, tr6, Q35 = {p3p5}, Q62 = {p6p2}]
G4 = [WT1, Ir2, tr3, WT4, Ir5, tr6]
G5 = [WT1, Ak2, tr3, WT4, Ak5, tr6, Q26 = {p2p6}, Q53 = {p5p3}]
G6 = [WT1, WT2, ret3, WT4, WT5, ret6, Q23 ={p2p3}, Q56 ={p5p6}]
G7 = [WT1, WT2, I3, WT4, WT5, I6, Q23 = {p2p3}, Q56 = {p5p6}, Q31 = {p3p1}, Q64 = {p6p4}]
G8 = [I1, WT2, stw3, I4, WT5, stw6]
Gf = [WT1, WT2, trw3, WT4, WT5, trw6, Q13= {p1p3}, Q46 = {p4p6}, Q35={p3p5}, Q62 = {p6p2}]

Figure 4.10: *Global State Transitions of CFSMs in Fig. 4.9*

even if some messages ($p1p3$, $p3p5$, $p4p6$, $p6p2$) are present in the message queues. This is a deadlock situation defined in Definition 4.5.1, where message queues are not empty. This behavior can be exploited for deadlock detection of NoC. We put forth this behavior in our proposed Theorem 4.5.2.

Theorem 4.5.2. *Consider a communication system with n CFSMs, where delayed reception is present but no unspecified reception is present. If all CFSMs are in receiving states and no transition is possible in those CFSMs, it is a deadlock situation due to cyclic dependencies on waiting for messages amongst CFSMs to proceed with further transitions.*

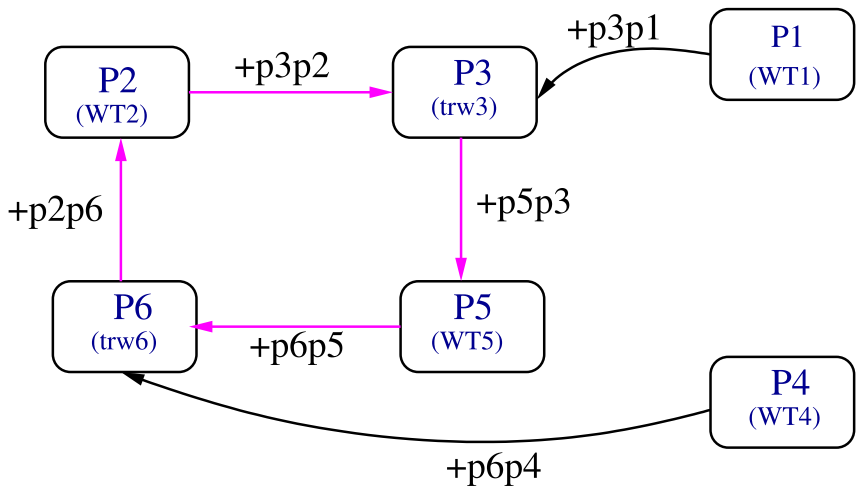


Figure 4.11: *Cyclic Dependency Graph for the CFSMs in Fig. 4.9*

Proof. Let us consider a CFSM based communication system with n number of processes $P_1, P_2, P_3, \ldots, P_n$, with no unspecified reception or design error. After some transitions, the system reaches a global state G_f. Let there be no further transition possible from G_f and all CFSMs are in receiving states. Let $G_f = [s_1, s_2, \ldots, s_n, Q]$ and current states of $P_1, P_2, P_3, \ldots, P_n$ are $s_1, s_2, \ldots, s_n$, respectively. It is given that all CFSMs are in receiving states. Therefore, they cannot generate any message and no transition is possible due to generation of new messages. The set of messages that are waiting at their specific message queues to be consumed are represented by Q. There are two possible scenarios in Q. i) All message queues are empty, i.e., $Q = \phi$. If there is no message in the message queues, there is no chance of consuming a message. or ii) There are some messages present in the message queue. For each process P_i, $i = 1, 2, 3, \ldots, n$, is waiting for message from some process P_j, i.e., from message queue q_{ji}. Since it is given that no transition is possible, any message consumption is also not possible. Therefore, the message queue q_{ji} must be empty, even though other message queues from where P_i can receive message may not be empty.

In both the cases, no transition is possible from any process and the global state G_f is in a deadlock state. Now we have to prove that the deadlock is due to the presence of a cyclic dependency in the system. We are proving this by the method of contradiction. Let us assume, there is no cyclic dependency for messages in the system. Since all processes are in receiving state and no transition is possible, each process is waiting for a specific message from one specific machine. Let us assume, P_1 is waiting for a message from P_2. Similarly, P_2 is waiting for a process P_3 and P_3 is waiting for a process P_4. If P_4 is waiting for P_1 or P_2 or P_3, it creates a cyclic dependency. Therefore, P_4 must be waiting for some process, other than P_1, P_2 and P_3, say P_5. In similar fashion, the waiting chain continues and reaches a situation where P_{n-1} is waiting for P_n. It is given that, no transition is possible from P_n as well. It must be waiting for a message from another processes P_x. We assume that there is no cyclic dependency in the system. If $P_x \in \{P_1, P_2, P_3, \ \ldots, P_{n-1}\}$, it creates a dependency cycle. Therefore, to satisfy our assumption, it must be true that, $P_x \notin \{P_1, P_2, P_3, \ \ldots, P_{n-1}\}$. But there is no other process left in the system. Therefore, $P_x \in \{P_1, P_2, P_3, \ \ldots, P_{n-1}\}$. Thus, P_x completes a cycle where processes are waiting for each other. Thus we reach a contradiction of our assumption that there is no cyclic dependency present in the system. Hence, we conclude that, if all CFSMs are in receiving states and no transition is possible in those CFSMs, there are cyclic dependencies on waiting for messages amongst CFSMs that create a deadlock. $\square$

4.5.3 Deadlock Detection Framework

For detecting deadlock, our CFSM based NoC models need to be analysed with respect to different routing algorithms and traffic patterns. In this subsection, we describe our application specific framework and methodology used for this purpose.

Our framework accepts packets corresponding to an application and determines if it is possible to deliver all the packets to its destination. An application is given as a sequence of packets. For each packet, the source and destination router and the time stamp of its generation are mentioned. If a deadlock situation is detected, the program exits reporting deadlock along with the packets that are stuck in cyclic dependency. Let us consider, we have n number of CFSMs in a Mesh NoC. Specific message queues are maintained between each pair of communicating CFSMs. Synchronization between the CFSMs are maintained by blocking read of message queues. Initially, all such message queues are empty and all the CFSMs are in their initial states. The current state of each machine and messages present in each message queues in the CFSM system are represented using global state, as defined in Section 4.3. Global state of the system is updated after generation or consumption of each message during each transition by any CFSM, by changing CFSM state and by addition or deletion of a message from the specific message queue. We are using a term *iteration* in our working methodology. *By one iteration, it means generation or consumption of all possible messages in parallel by all the CFSMs from their current states and updating the global state accordingly. The concept of one iteration in our CFSM based system is identical with a global clock tick of a hardware system where many units execute in parallel.*

Before starting each iteration, the input traffic sample is checked, if there is any packet left which is not delivered. If all the packets from the input traffic pattern are delivered, the program exits gracefully reporting delivery for all packets and no deadlock is detected. On the other hand, if some packets in the input traffic sample, are not yet delivered, the iterations continue. Each time a message is generated, it is added to the exact message queue and is updated in the global state. If a message is consumed, it is deleted from the message queue and is updated in the global state. Our framework has the flexibility of embedding different routing algorithms. ComputeN3, shown in Fig. 4.5(b), invokes routing algorithm when the state of ComputeN3 is *comp*. Packet information like source address, destination address, its current router location, its delivery status are maintained globally. Suppose some packets are not delivered yet and a new iteration is started. In that new iteration, let there be no generation or consumption of new message is possible in any of the CFSMs. It indicates that all CFSMs are in receiving states and no transition is possible from the current global state of the NoC model. According to Theorem 4.5.2, it infers that some packets are waiting for each other in cyclic dependency and not able to progress. Deadlock is reported in such cases and the program terminates.

4.6 Automation of CFSM Model Generation

An NoC may contain hundreds of routers along with the associated core. The number of CFSMs used in this work for modeling Mesh NoCs of different sizes are shown in Table 4.2.

Table 4.3: *Number of states on different CFSM model*

Active Ports	No of states on different CFSM model				
	Buffer (1 slot)	Switch	Route Copmute	Arbiter	Scheduler
3	4	11	4	7	6
4	4	14	5	9	12
5	4	17	6	11	20

The number of states for different NoC components are shown the Table 4.3. Manually creating thousands of such CFSMs is a time consuming and error prone job. Therefore, we develop an automated NoC model generator. In a NxN Mesh NoC, each row and each column contains N number of routers. Our CFSM based NoC model generator takes the value of N and the number of available buffer slots in the input port and the number of virtual channels as inputs from the user. Based on that inputs, our program determines the number of active ports for each router and generates the appropriate CFSMs for each component in the router. The CFSMs of same functionality consist of different number of states, based on its position in NxN Mesh and the number of active ports in that position. The Table 4.3 shows different number of states in different functional units. Our program generates a text file containing all the CFSMs represented in matrix format. Comments are added for each CFSM for better readability of the generated model. Name of the functional units (Buffer, Switch, Arbiter, etc.), port name, router number, etc. are added in the comments. The message names across different CFSMs are unique to avoid confusion and to avoid processing of the wrong messages by a program. Therefore, the naming convention described in the Table 4.1 is followed in our automated model generation. The NoC model described in this work is generic. Any routing algorithm can be embedded in our simulation framework. It may be noted that a routing algorithm is invoked from the state *comp* of the ComputeN3 in Fig. 4.5(b). We associate a unique number and a unique name to the CFSMs generated by our application.

4.6.1 Bounded Communication

A CFSM model is said to be *bounded* if length of each message queue is finite during all of its possible transitions [77]. It is important that the CFSM model of a system is bounded. In case, communications are not bounded in a design, checking of deadlock and progress properties are undecidable in that CFSM model [77]. A communication system with n functional units, represented with n CFSMs, is said to be *bounded* by a non negative finite number K iff for each reachable global states, the length l_{ij} of the each message queue Q_{ij}, $i = 1, 2, 3, \ldots, n$ and $j = 1, 2, 3, \ldots, n$, is bounded by K, i.e., $l_{ij} \leq K$.

A communicating machine is called *alternating* iff each of its sending edges is followed by receiving edges only. Gouda et al. [77] proposed a theorem for boundedness on alternating communicating machines. According to that theorem, the communication of any network with two alternating machines (CFSMs) is bounded by two. In that work, a communication network consisting of only two machines is considered. Motivated from this work [77], we put forth Lemma 4.6.1 and Lemma 4.6.2, applicable to any number of CFSMs. These two lemmas are applicable to our model as well for analyzing bounded communication.

Lemma 4.6.1. *In a communication system with finite number of CFSMs, a specific communication between two machines $M1$ and $M2$ is said to be alternating iff there are no consecutive sending edges where messages are sent from machine $M1$ to $M2$ or from machine $M2$ to $M1$. Such communication is bounded as the concerned message queues are bounded.*

Lemma 4.6.2. *In a communication system, let a machine M_0 be communicating with n machines. If all the communication between $\{(M_0, M_1), (M_0, M_2), ..., (M_0, M_n)\}$ are bounded, we term the communication of machine M_0 as bounded communication.*

A self loop in a CFSM contributes to infinite message queue size and unbounded communication. There is no self loop present in the CFSM model described in Section 4.4. Most of the CFSMs presented in this work exhibit alternating communication with the other CFSMs with whom it is communicating. All such CFSMs are bounded. For a communication that is not alternating in our design, deducing the maxmimum possible size of message queue by careful analysis of the transitions gives a finite message queue size only. By applying these two lemmas and with minute analysis of the transitions, we found that all our CFSM model are bounded.

4.6.2 Complexity of the CFSM Model

The complexity of our framework grows with the modeling parameters like NoC size, buffer size and number of VCs. The number of CFSMs in an NoC model depends on the total number of active ports in the NoC model. Let us consider a NxN Mesh NoC. Each corner router has three active ports. Number of such routers are four. Each boundary routers, other than the four corner routers, has four active ports. The number of such routers are $(N-2)\times 4$. Each non boundary router has five active ports. The number of such routers are $(N-2)\times(N-2)$. Therefore, total number of active ports $A = (4\times 3)+(N-2)\times 4\times 4+(N-2)\times(N-2)\times 5 = 5N^2-4N$. For each active port, five CFSMs are associated with, if buffer capacity $B = 1$. For A number of active ports with buffer size $B = 1$, the total number of CFSMs will be $A\times 5$. If buffer size $B > 1$, we can calculate the additional number of CFSMs for additional buffer size as $(A - N \times N) \times (B - 1)$. Therefore, the total number of CFSMs for $N \times N$ NoC with buffer size B is $A \times 5 + (A - N \times N) \times (B - 1)$. Putting the value of A, after simplification we find that total number of CFSMs $= 21N^2 - 16N + 4BN^2 - 4BN$. Therefore, the complexity of our CFSM model is $O(BN^2)$. Adding VCs to our NoC model results in increasing the number of CFSMs in the model. In an $N \times N$ NoC let the buffer capacity be B and number of VCs be V. Calculating in similar manner the required number of CFSMs is found to be $10N^2 + 11VN^2 + 4BVN^2 - 8N - 8VN - 4BVN$. Therefore, the complexity of our CFSM model with VCs is $O(BVN^2)$.

4.7 Experimental Results and Analysis

In this section, we put forth experimental results and analysis of deadlock detection, considering three routing algorithms on Mesh and Torus NoCs. We consider both randomly generated traffic patterns and synthetic traffic patterns in our experiments. We also compare our deadlock detection results with the results of Booksim simulator. Justification for the presence of deadlock in adaptive routing algorithm is also presented in this section.

We have implemented three minimal path routing algorithms, XY-routing (deterministic) [8], dynamic-XY routing (adaptive) [8] and modified West-First routing (adaptive), for Mesh and Torus NoCs. In XY-algorithm for Mesh NoC, a packet first moves towards X-direction. After the distance in X-direction is covered, it starts moving in Y-direction. In case of XY-algorithm for Torus, to traverse in X-direction (or Y-direction), the packet may traverse in clockwise or anti-clockwise direction to reach the destination. Out of these two directions, XY-routing in Torus chooses the shortest path as elaborated in [103]. In case of dynamic-XY routing, a packet moves on X-direction or Y-direction towards destination, based on the stress value of the current router's neighbours towards the destination. Stress value indicates the number of packets present in the buffer of a router. If more packets are present in the adjacent router's buffer in X-direction and fewer packets are present in the adjacent router's buffer in Y-direction, a packet traverses in Y-direction towards the destination. This flexibility is not provided in XY-routing. West-First routing is described in [9], where two turns are restricted. In modified West-First routing we have restricted only one turn (North-West turn) to make it deadlock prone as per Turn model as shown in Fig. 4.16. If the destination of a packet is towards North-West direction, first the packet has to traverse in West direction first to avoid North-West turn. For the rest of the directions, it is similar to dynamic-XY routing.

For all the experiments, a single Intel Core i5 3.20GHz, 8GB RAM machine is used. We apply randomly generated traffic patterns for Experiments I, II, III and IV. In Experiment I, the impact on the execution time with the increase in NoC size (Mesh and Torus) for XY-routing is analysed. Deadlock is detected in Torus NoC for some cases and we analysed the reason behind it. In Experiment II, the impact on the execution time with the increase of packets (in input traffic pattern) for XY-routing is analysed for Mesh and Torus NoCs. Deadlock detection using dynamic XY routing and modified West-First routing are analysed in the Experiment III. In Experiment IV, we analyse deadlock avoidance by increasing the buffer size. In the Experiment V, we compare our deadlock detection results with Booksim simulator by applying synthetic traffic patterns extracted from the Booksim simulator.

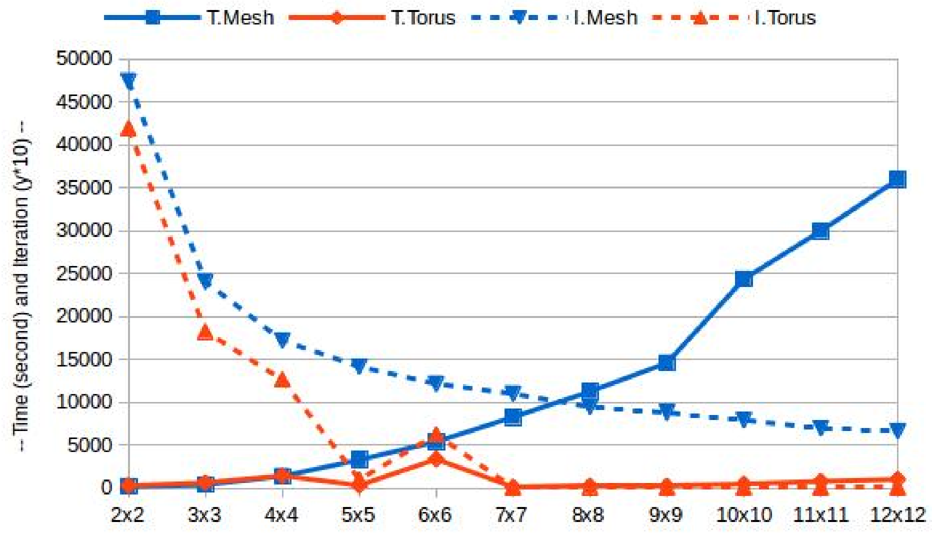


Figure 4.12: *Experiment I: XY Algorithm on Mesh and Torus NoCs with traffic pattern of 100000 packets. Run-time increases with the increase of NoC size if there is no deadlock. Deadlock is detected in Torus NoC if NoC size is bigger than* 4x4.

4.7.1 Experiment I: XY-routing on Mesh and Torus of Different Sizes

In Experiment I, a randomly generated sample of 100000 packets is taken as input for Mesh and Torus NoCs with buffer size of two. The number of packets is the same in all the samples, only the source and destination addresses and the time stamps are different according to the size of NoC. The XY algorithm is considered for both the cases. With the increase in number of routers in NoC, the number of CFSMs required for modeling also increases. Therefore, execution time is higher for verifying larger NoCs even if the same number of packets are given as input. In Fig. 4.12, the execution time is given in seconds by the run-time graph (solid lines) for Mesh and Torus NoCs. In Mesh NoC, no deadlock has been found and the process terminated after all packets get delivered. On the other hand, *deadlock is detected by our framework in Torus NoC after it exceeds the size of* 4x4. There are cyclic paths in each row and each column of a Torus NoC due to the wraparound channels connecting the routers at the opposite end. Deadlock occurs because of the presence of cyclic paths. In case of 2x2, 3x3 and 4x4 Torus, the added external communication link is only utilised by packets with the source and destination addresses in the exact opposite boundaries. Other packets do not use the wraparound channels, as it is not helpful in reducing the hop count (number of routers in the path). Therefore, no deadlock situation arises for 2x2, 3x3 and 4x4 Torus NoC. From 5x5 Torus, not only the packets with source and destination pair in the exact opposite boundaries, but also other types of packets (source or destination addresses not in boundary line) utilize the wraparound channels. Therefore, there is a chance for cyclic dependency and deadlock after 5x5 Torus.

In Experiment I, we have also shown the number of iterations taken by each execution. Number of iterations are normalized in the graph of Fig. 4.12 by dividing them with 10 for representation convenience. By one iteration, we mean generation or consumption of all possible messages by all the CFSMs from their current states at the present moment as described in Subsection 4.5.3. *For processing the same number of input packets, with increase in NoC size, the number of iterations decrease.* For bigger NoC, in a single iteration, more number of transitions happens in parallel and more number of packets move. For example, 100000 packets are distributed across 4 routers in a 2x2 NoC. Whereas, the same number of packets are distributed across 144 routers in a 12x12 NoC. Therefore, in a real scenario, 144 routers should take less time to process 100000 packets than the time taken by 4 routers. This is reflected by the iteration graph for Mesh NoC (dotted line) in Fig. 4.12. For 2x2, 3x3 and 4x4 NoC, no deadlock occurs for both Torus and Mesh. For these three NoCs, Torus takes less number of iterations as compared to Mesh. It indicates that Torus is more efficient than Mesh in terms of delivering packets. But one drawback is that deadlock is possible even with XY-routing in higher dimensions in this topology due to the presence of cyclic path.

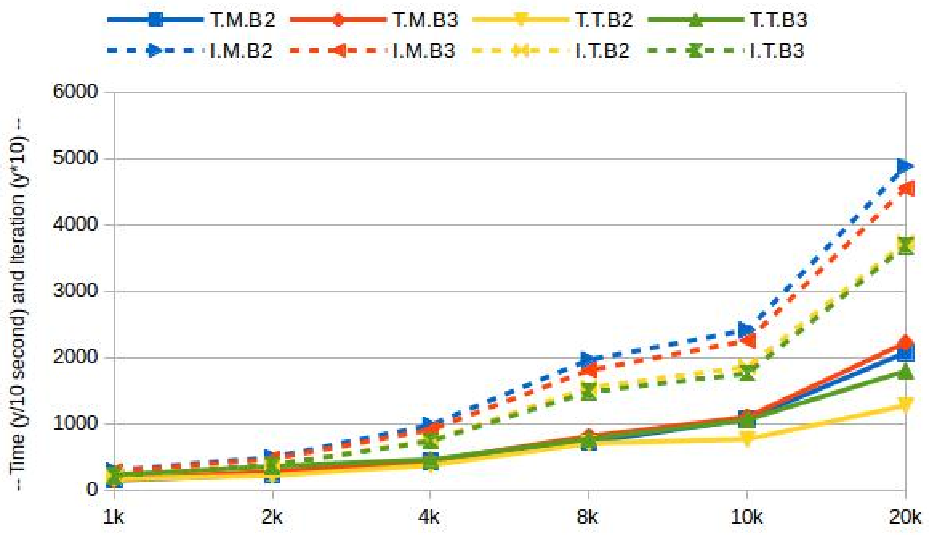


Figure 4.13: *Experiment II: XY Algorithm on a* 3x3 *NoC (Mesh and Torus) applying traffic patterns of different sizes. (T.M.B2: Time taken by Mesh with buffer size 2, T.T.B2: Time taken by Torus with buffer size 2, I.M.B2: Iterations for Mesh with buffer size 2, I.T.B2: Iterations for Torus with buffer size 2 etc.)*

4.7.2 Experiment II: XY-routing on Mesh and Torus with Different Traffic Size

In Experiment II, we apply different numbers of packets on a 3x3 Mesh and Torus NoCs with XY-routing. There is no deadlock and all packets get delivered at their destination. The execution time in solid lines and the number of iterations in dotted lines in Mesh and Torus NoC for buffer sizes 2 and 3 are shown in Fig. 4.13. In Fig. 4.13, T.M.B2 indicates the time taken by Mesh with buffer size two. Similarly, I.T.B3 indicates the number of iterations in Torus for buffer size 3. The iterations in the graph are normalized by dividing them with 10 for representation convenience. Iteration graph (dotted) shows that Torus is performing better in delivering packets in less number of iterations than that of Mesh. We also observe the change in behavior by varying the buffer size from two to three. NoC with buffer size of three takes fewer iterations than buffer size of two. It implies the better performance for NoC with buffer size of three as compared to buffer size two. By increasing numbers of packets, both execution time and number of iterations increase.

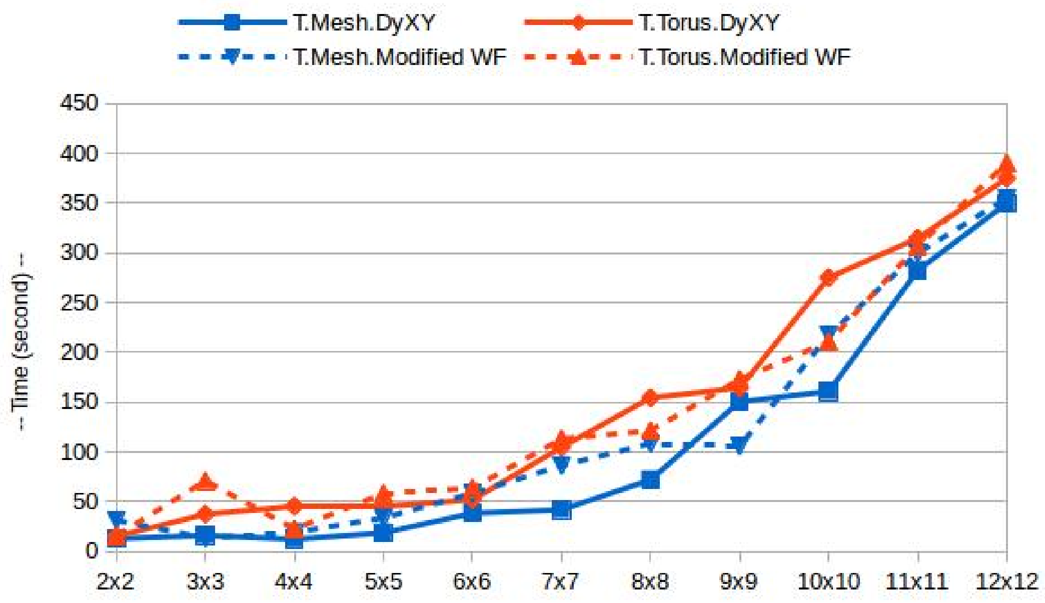


Figure 4.14: *Experiment III: Dynamic XY Algorithm and Modified West-First Algorithm (NW Turn Restricted) on traffic patterns of 100000 packets. Deadlock detection time depends on the pattern of the input traffic.*

4.7.3 Experiment III: Deadlock Detection on Dynamic XY-routing and Modified West-First routing

Experiment III reflects the behavior of dynamic XY (adaptive) and modified West-First (adaptive) routing in both Mesh and Torus NoCs. Samples of randomly generated 100000 packets are applied to different NoC with buffer size one. *In all the cases, deadlock is detected.* Using solid line (dynamic XY) and dotted line (modified West-First), the deadlock detection time is depicted in the graph in Fig. 4.14. The graph indicates that modified West-First algorithm survives more time in some traffic patterns without stuck into deadlock and in some other cases, dynamic XY survives longer without facing deadlock. The time required for detecting deadlock depends on the traffic pattern of the input. Since Torus topology uses shorter path to deliver packets, it survives more time without deadlock than that of Mesh NoC for the same traffic pattern. Once deadlock is detected, the simulation is terminated reporting deadlock. It may be noted in Fig. 4.14 that our method is able to detect deadlocks within 400 seconds in all the cases. All the deadlocks detected by our method are real deadlocks.

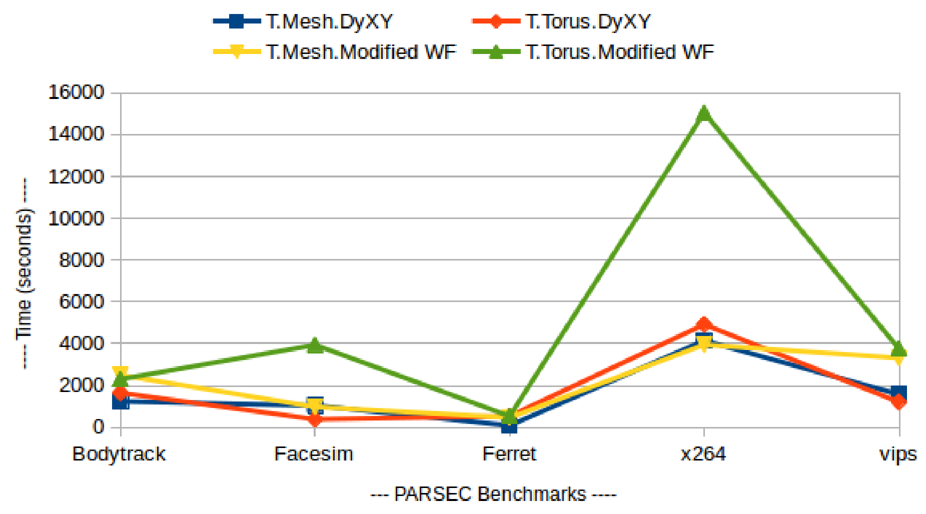


Figure 4.15: *Deadlock Detection: Dynamic XY Algorithm and Modified West-First Algorithm (NW Turn Restricted) using PARSEC benchmarks in an 8x8 Mesh NoC.*

We have performed similar experiments using traffic patterns from the PARSEC (Princeton Application Repository for Shared-Memory Computers) benchmark suite [104]. PARSEC benchmark suites are used for the research on multiprocessor system architecture. We have used five workloads from PARSEC benchmark suites that are related to diverse application domain. They are namely Bodytrack, Facesim, Ferret, x264 and Vips. Bodytrack workload is related to computer vision and artificial intelligence application domain, Facesim is related to animation application, Ferret is related to similarity search application, and both x264 and Vips are related to media file processing applications. Using PARSEC benchmark suites with the Gem5 [25] simulator, we have extracted the corresponding traffic patterns. The traffic pattern corresponding to these different workloads are simulated in an 8x8 Mesh NoC using both Dynamic XY Algorithm and Modified West-First Algorithm. Experimental results are plotted in Fig. 4.15. Deadlocks are detected for all the cases. Experimental results show that deadlock detection time for PARSEC benchmark suites is higher than that of randomly generated traffic shown in Fig. 4.14. The traffic patterns from PARSEC benchmark suites sustain more time without involving in deadlock. Experimental results show that deadlock detection time for PARSEC benchmark suites are higher than that of randomly generated traffic shown in Fig. 4.14. The traffic patterns from PARSEC benchmark suites sustain more time without involving in deadlock.

Table 4.4: *Experiment IV: Dynamic XY (DyXY) and Modified West-First (MWF) Algorithm in 5x5 NoC (Mesh and Torus) for 1600 packets. (D: Deadlock, T: Time in Second, I: Iterations)*

#Buffer	Mesh NoC						Torus NoC					
	DyXY			MWF			DyXY			MWF		
	D	**T**	**I**	**D**	**T**	**I**	**D**	**T**	**I**	**D**	**T**	**I**
1	Y	31	283	Y	27	534	Y	49	1219	Y	60	1483
2	Y	55	607	Y	33	816	N	73	1699	N	71	1662
3	Y	54	1367	Y	47	1120	N	93	1506	N	91	1521
4	N	114	2995	Y	403	794	N	97	1428	N	96	1464
5	N	116	2534	N	102	2031	N	106	1481	N	105	1451

4.7.4 Experiment IV: Deadlock Avoidance with Increasing Buffer Size

In Experiment IV, the relation of deadlock with buffer size is analysed. Buffer size is one major factor on which occurrence of a deadlock in a given traffic pattern depends. By increasing buffer size, some deadlock scenarios can be avoided in a given traffic pattern. Considering dynamic XY routing and modified West-First routing on a 5x5 NoC (with different buffer sizes), we have applied traffic sample of 1600 randomly generated packets in Mesh and Torus NoC. We have considered moderate size packet sample (1600 packet) so that for each algorithm we get deadlock with buffer size one in all cases. Table 4.4 shows the verification results of this experiment. Deadlock is detected for all the cases when buffer capacity is one. After increasing buffer capacity to two, the deadlock disappears for both the algorithms in Torus NoC. Deadlock still appears in Mesh NoC even after increasing the buffer size to three. With buffer size four, the deadlock disappears for dynamic XY algorithm (Mesh). Modified West-First is still faces deadlock for this traffic pattern in Mesh NoC when buffer size is four. Finally, with buffer size five, the deadlock disappears for modified West-First in Mesh NoC. Experimental results indicate that for this traffic pattern, the optimum buffer capacity to avoid deadlock in case of dynamic XY routing in Mesh and Torus NoC is four and two, respectively.

Table 4.5: *Experiment V: Booksim result on applying Uniform (U) and Tornado (T) traffic pattern with different injection rate (0.05 and 0.08).*

NoC	**Booksim results**							
	Deadlock (W = Warning, N = No)				**Booksim Time (second)**			
	U .05	**U .08**	**T .05**	**T .08**	**U .05**	**U .08**	**T .05**	**T .08**
2x2	W	W	N	N	7	4	31	52
3x3	W	W	W	W	5	5	37	8
4x4	W	W	W	W	10	11	10	11
5x5	W	W	W	W	17	20	18	19
6x6	W	W	W	W	26	30	28	31
7x7	W	W	W	W	39	34	40	43
8x8	W	W	W	W	54	58	51	55
9x9	W	W	W	W	69	75	65	72
10x10	W	W	W	W	99	101	83	86
11x11	W	W	W	W	139	122	109	113
12x12	W	W	W	W	169	153	141	165

Table 4.6: *Experiment V: CFSM results on applying Uniform (U) and Tornado (T) traffic pattern with different injection rate (0.05 and 0.08).*

NoC	**CFSM results**							
	Deadlock (Y = Confirmed, N = No)				**CFSM Time (second)**			
	U .05	**U .08**	**T .05**	**T .08**	**U .05**	**U .08**	**T .05**	**T .08**
2x2	Y	Y	N	N	30	25	64	94
3x3	Y	Y	Y	Y	27	43	156	117
4x4	**N**	Y	Y	Y	24	14	232	33
5x5	Y	Y	Y	Y	21	34	31	29
6x6	Y	Y	Y	Y	49	31	36	44
7x7	Y	**N**	Y	Y	52	68	55	52
8x8	Y	Y	Y	Y	100	93	70	81
9x9	Y	Y	Y	Y	121	137	90	115
10x10	Y	Y	Y	Y	216	182	169	190
11x11	**N**	Y	Y	Y	490	258	140	230
12x12	Y	Y	Y	Y	239	444	393	479

4.7.5 Experiment V: Detection of False Positive Deadlock Warning in Booksim Simulator

In Experiment V, we compare our deadlock detection results with Booksim simulator [24]. In Booksim simulator, we invoke dynamic XY routing for the purpose of reproducing the possible deadlock warning messages. In addition, for reproducing deadlock warning in Booksim, we configure synthetic traffic patterns as uniform and tornado, injection rates as 0.05 and 0.08, the number of VC as one, buffer size as one and packet size as one. By considering NoC of different sizes, we are able to reproduce possible deadlock warning in Booksim simulator as shown in Table 4.5. It may be noted that the simulator stops with a possible deadlock warning in these scenarios. We have extracted the same traffic pattern from Booksim and apply it to our CFSM based deadlock detection framework to compare the results. Experimental results from CFSM based simulation are shown in Table 4.6. From the experimental results in Table 4.5 and Table 4.6, most of the cases, our CFSM framework gives confirmed deadlock where Booksim is giving deadlock warning. Out of 42 test scenarios with deadlock warning in Booksim, we found confirm deadlock in 39 cases in our CFSM based framework. In three cases, *all packets get delivered and no deadlock is reported by our framework. Therefore, our framework finds false-positive deadlock warning in Booksim.* This shows the usefulness of our method in detecting application specific deadlocks. Since the CFSM based formal model rigorously checks the system, our framework is taking more time than that of Booksim. However, the run-time of our tool is less than 9 minutes for all cases. Both the experimental results and theoretical proof presented in Subsection 4.5.2 indicate that CFSM based model can detect deadlock accurately.

4.7.6 Justification for Deadlock in Adaptive Routing Algorithms

For the purpose of designing a deadlock free routing algorithm some turns are prohibited in the turn model work [9]. Fig. 4.16 shows the restricted turn for XY routing, West-First routing and modified West-First routing. In XY routing all possible turns from Y-direction

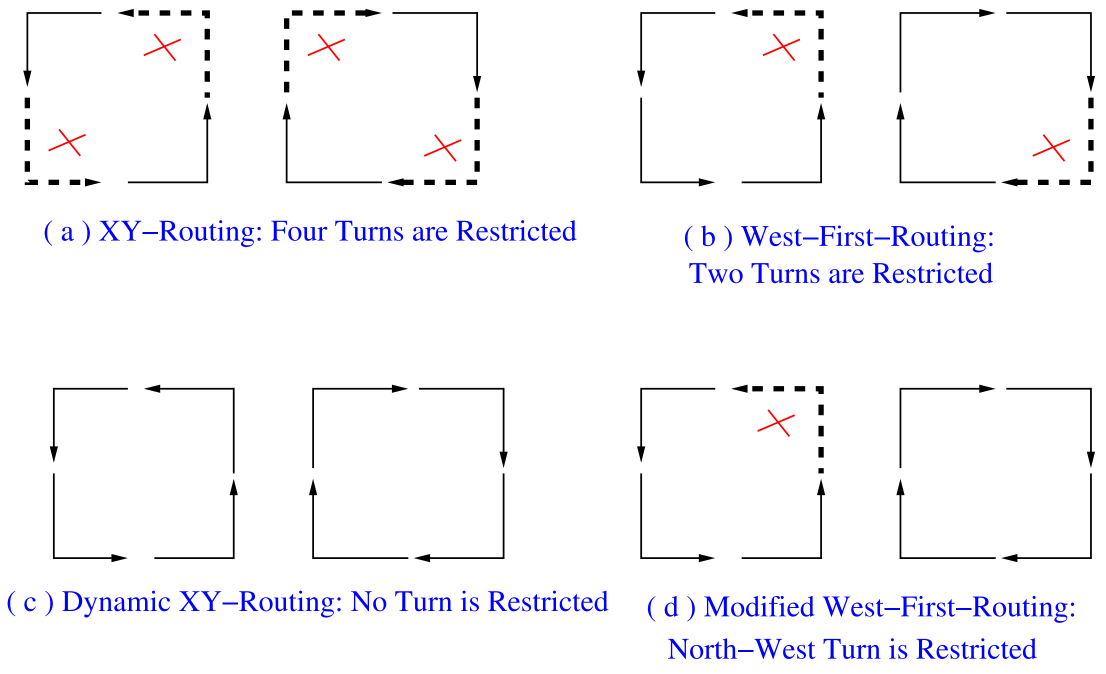

(a) XY–Routing: Four Turns are Restricted
(b) West–First–Routing: Two Turns are Restricted
(c) Dynamic XY–Routing: No Turn is Restricted
(d) Modified West–First–Routing: North–West Turn is Restricted

Figure 4.16: *Turn model for routing algorithms: (a) XY-routing, (b) Dynamic XY-routing, (c) West-First routing and (d) Modified West-First routing*

to X-direction are restricted to make it deadlock free. West-First algorithm is a partially adaptive deadlock free algorithm [9] with restriction for two turns as shown in Fig. 4.16. In modified West-First routing, we have restricted one turn as shown in Fig. 4.16. As per the theorem on turn model [9], the minimum number of turns that must be prohibited to prevent deadlock in a two dimensional Mesh is two. On the contrary, there is no restriction on any turn in dynamic XY [8] routing and only one turn is restricted on modified West-First algorithm as shown in Fig. 4.16. Since no turn is restricted in dynamic XY algorithm and only one turn is restricted in modified West-First routing, the deadlock is possible for both dynamic XY routing and modified West-First routing algorithm. Our CFSM based framework also confirms the same under different traffic patterns.

4.8 Conclusion

In this chapter, we have presented an approach of creating NoC model with different topologies using CFSM. The CFSM based NoC model generation process is automated. We have developed a simulation framework on top of the CFSM based NoC model. It detects application specific deadlock with respect to a given routing algorithm. We have demonstrated the working of our approach for one static and two adaptive routing algorithms in Mesh and Torus topologies. Experimental results show that our framework is scalable and robust to handle a large number of packets. Deadlocks are detected for dynamic XY-routing in Mesh NoC and for XY-routing in Torus NoC by this framework. This approach reports confirmed deadlock instead of a warning given by a commonly used NoC simulator. In fact, our framework identifies that some deadlock warnings of Booksim simulator are false-positives. As the proposed scheme is based on formal models, it does not report any false deadlocks. Moreover, on detecting a deadlock, this framework reports the deadlock scenario with detailed resource dependency. Deadlock scenarios with detailed resource dependency are helpful on deadlock avoidance in NoC.

5

Deadlock Avoidance in Torus NoC Using Arc Model and Directional Dependency Graph

5.1 Introduction

Torus NoC is a symmetric NoC topology where each router has an equal number of neighbors. The folded structure of a two dimensional Torus NoC looks like a hollow pipe with both ends are connected. A router is connected with its neighbours via a communication channel. The folded structure helps to avoid having wraparound channel substantially longer than the rest of the channels. The wraparound channels connect two routers present in two boundaries in opposite directions. Wraparound channels facilitate an alternate path between a given source and destination pairs. It is useful in reducing hop counts on a given traffic when the source and destination are near to the boundaries of the opposite side of the plain. For convenience, a Torus NoC is visualised as a combination of *Mesh sub-network* and *wraparound channels* as shown in Fig. 5.1. Here, the channels in red color are the wraparound channels and the rest of the channels are from the Mesh sub-network. *Torus NoC is deadlock prone due to the inherent cyclic path created by the wraparound channel.* Even for XY-routing, where sufficient routing restrictions are imposed for avoiding deadlock, if we use the algorithm in Torus without any additional precautions, it results in deadlock.

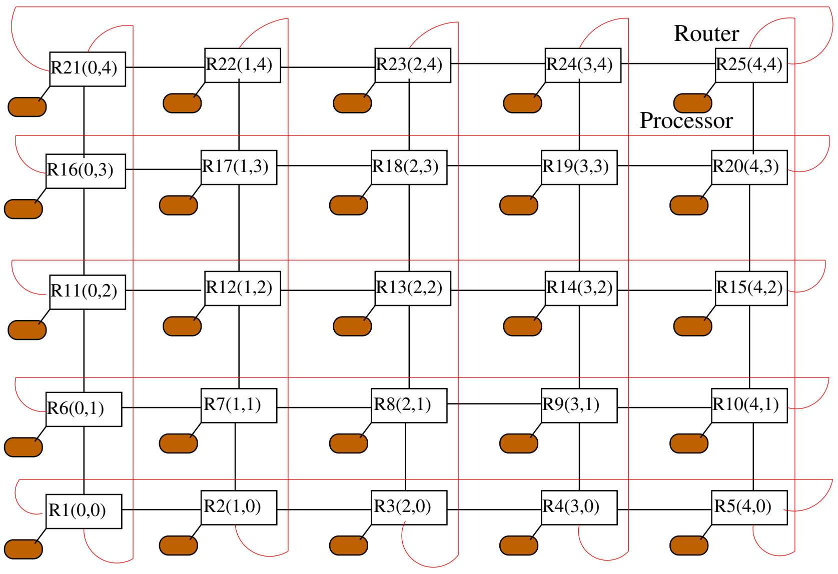


Figure 5.1: *Visualising* 5x5 *Torus NoC as a combination of Mesh sub-network and wraparound channels.*

5.1.1 Cyclic Resource Dependency in Torus NoC

In an NoC, packets traverse from their sources to respective destinations via passing through many intermediate routers. While reaching another intermediate router, a packet needs to be stored in the input port buffer of that router. If the buffer in that router is full, the packet has to wait until the intended buffer becomes free. The packet has to hold its current buffer during this waiting time. If a set of packets are involved in such hold and wait, and form a cycle of dependencies, no packet can make progress due to occurrence of deadlock. We have illustrated such a cyclic resource dependency scenario with help of the following example.

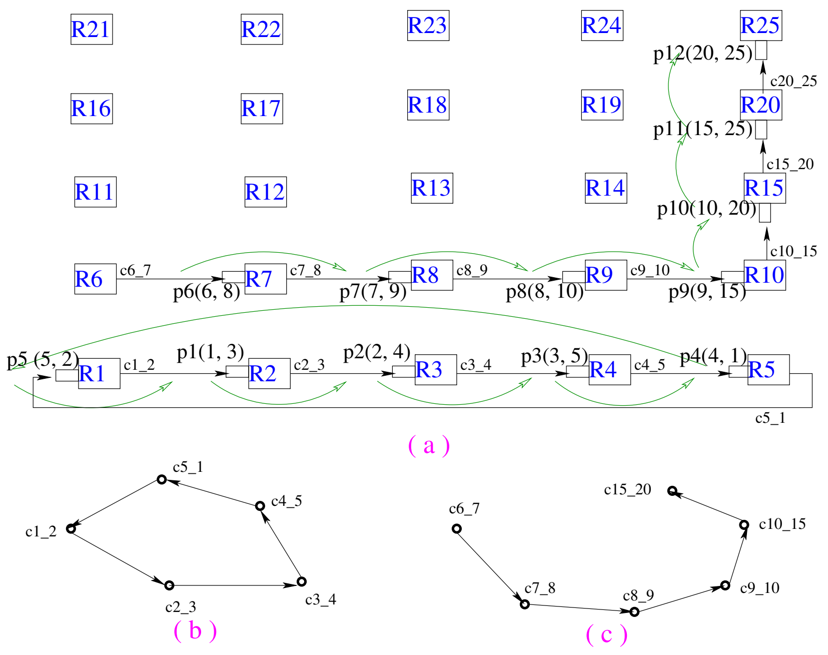


Figure 5.2: *(a) Resource dependency scenarios for a set of packets (source and destination are shown in the bracket), (b) Channel Dependency Graph representing deadlock scenario, (c) Channel Dependency Graph showing a resource dependency.*

Example 5.1. Let us consider five packets p1(1, 3), p2(2, 4), p3(3, 5), p4(4, 1) and p5(5, 2) in a 5x5 Torus NoC in Fig. 5.2(a). For each packet, the source and destination router numbers are put in the bracket. We have considered XY-routing along with wraparound channel for the traversal of the packets [3]. Wraparound channels are used only if the considered path using wraparound channel is shorter than the path permitted by XY-routing. Each packet has arrived the input buffer of the adjacent router towards destination from the source in their first hop movement as depicted in Fig. 5.2(a). For example, the packet p1(1, 3) has started from the source router R1 and moves towards the destination router R3. The packet has reached the West input port buffer of router R2 as shown in the figure. For the packet p4(4, 1), the shortest path is $R4 \rightarrow R5 \rightarrow R1$ with the wraparound channel. In similar way, path to be followed for p2(2, 4) is $R2 \rightarrow R3 \rightarrow R4$, for p3(3, 5) is $R3 \rightarrow R4 \rightarrow R5$, and for p5(5, 2) is $R5 \rightarrow R1 \rightarrow R2$. Packets p1, p2, p3, p4, and p5 are transmitted at the same time from their respective source routers. They have reached the input buffer of their next router, R2, R3, R4, R5, R1, respectively, as shown in Fig. 5.2(a). Let us assume that each router has a buffer with a capacity of one packet. Therefore, the

West input port buffer of each router in Fig. 5.2(a) is full at this moment. Packet p1(1, 3) is occupying West buffer of router R2, waiting for the release of West buffer of R3. Packet p2(2, 4) is occupying the same R3 buffer and is requesting R4 buffer. Packet p3(3, 5) is occupying R4 buffer and is requesting R5 buffer. Packet p4(4, 1) is occupying R5 buffer and is requesting R1 buffer. Lastly, packet p5(5, 2) is occupying R1 buffer and is requesting R2 buffer, currently occupied by packet p1. Thus, all five packets are waiting for each other for the release of buffer in a cyclic manner. This is an example of deadlock in Torus NoC. The other example with packets p6, p7, p8, p9, p10, p11 and p12 do no create cyclic dependency or deadlock. Dependency cycle is broken as the packet p12 has already arrived the destination router R25 and it would get delivered to the local core.

One point worth to be noted from the Example 5.1 is that the source and destination for each of the packets are considered to be two hops away from each other. We consider this because the resource dependency is possible amongst a group of packets iff the source and destinations are at least two hops away from each other [2]

5.1.2 Deadlock and its Representation in NoC

Representing deadlock in an informative way helps in understanding the root cause of deadlock and to work on its avoidance as well. Classically, there are two approaches namely Channel Dependency Graph (CDG) and Turn model that are used for this purpose.

5.1.2.1 Channel Dependency Graph

Deadlock in an interconnection network occurs when no packet can advance towards its destination as the respective queues in the network cycle is full [2, 105]. Dally et al. [2] propose an approach for analysing resource dependency in a network by considering dependency across communication channels. In this approach, each channel and the connected queue (i.e., buffer) are represented together as a single unit [2]. The combination of channel with its associated buffer are considered as a vertex and a graph can be constructed by applying a routing algorithm in a given topology. This graph is called Channel Dependency Graph

(CDG). This approach facilitate sufficient information for representing resource dependency. A vertex (channel-buffer combination) is connected with another vertex with a directed edge permitted by a routing algorithm. The directed edge represents a communication channel. A directed path in a CDG connecting several vertices represents a possible path permitted by a routing algorithm. For constructing a CDG, it is convenient to consider a set of packets and the graph is formed by applying the routing algorithm on those packets. However, it is not mandatory to consider a set of packets for constructing a CDG. A given topology and given routing algorithm is sufficient to form CDGs [2]. Several such CDGs are possible with respect to a given routing algorithm and a given topology. With respect to a given routing algorithm, if cycle formation is possible in any of the CDGs then the routing algorithm is deadlock prone. There might present sequence of packets in that cyclic path represented by CDG to form a deadlock. As per Dally's theory, a routing algorithm in a given topology is deadlock free if no cycle exists in any of the CDGs for the given routing algorithm [2].

Two resource dependency scenarios are shown in Fig. 5.2(a). Out of them, one leads to a deadlock situation due to cyclic resource dependency and the other one is a resource dependency which would be resolved eventually. Both these scenarios are represented using CDG in Fig. 5.2(b) and Fig. 5.2(c), respectively. A unique name is given to each channel and the associated buffer. For example, the channel from router R1 to router R2 and the connected buffer at W input port of the router R2 are combined and is denoted as c1_2 in the figure. The CDG in Fig. 5.2(b) represents a deadlock. The other CDG in Fig. 5.2(c) is not a deadlock scenario.

5.1.2.2 Turn Model

In Mesh NoC, a packet exercises suitable Turn to reach its destination from the eight possible Turns shown in Fig. 5.3. An anti-clockwise cycle is formed by the four Turns namely West-South (WS), South-East (SE), East-North (EN) and North-West (NW) are shown in Fig. 5.3(a). Allowing these four Turns in a routing algorithm makes the algorithm deadlock prone [1, 9]. Similarly, allowing ES (East-South), SW (South-West), WN (West-

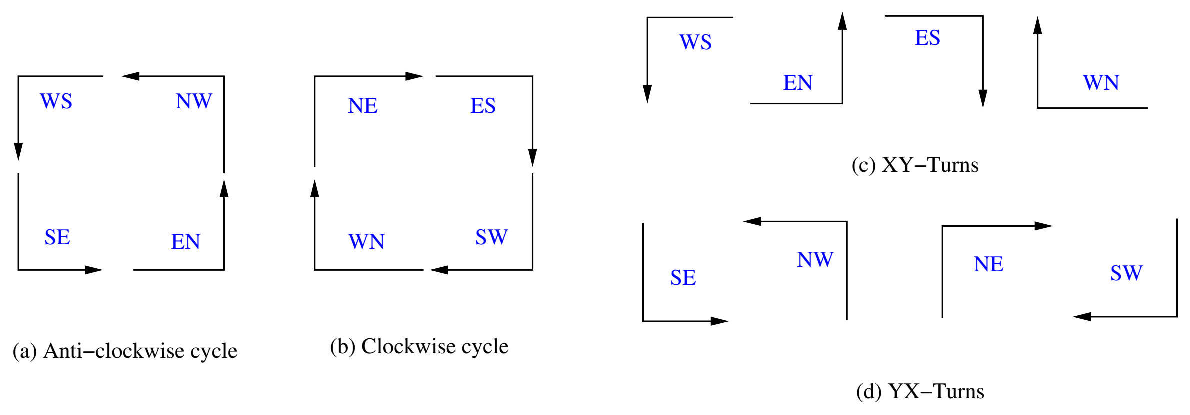


Figure 5.3: *Turn model: (a) Anti-clockwise cycle, (b) Clockwise cycle, (c) XY-Turns, (d) YX-Turns*

North) and NE (North-East) in a routing algorithm has the potential of creating cyclic dependency in clockwise direction that leads to a deadlock. The set of Turns that are sufficient for creating a clockwise cycle are shown in Fig. 5.3(b). All possible Turns in a Mesh NoC are classified into two groups: XY-turns and YX-turns. In XY-turns, a packet first moves in a X-direction (East or West) and then it changes the direction by moving towards a Y-direction (North or South). Four possible XY-turns are shown in Fig. 5.3(c). In XY-routing algorithm [8] for Mesh NoC, only XY-turns are permitted. A packet first traverses in X-direction towards the destination. After X-distance is covered, it moves in Y-direction. YX-turns are restricted in XY-routing. The four possible YX-turns are shown in Fig. 5.3(d). XY-algorithm is deadlock free in Mesh NoC as per *Turn model* [9] since neither clockwise nor anti-clockwise cycle formation is possible with permitted Turns. Similarly, many deadlock free routing algorithms like West-First routing, Negative-First routing etc. are proposed for Mesh NoC based on the *Turn model* in [1, 9].

5.1.3 Does Deadlock Always Possible in Torus due to Wraparound Channel?

For 3x3 and 4x4 Torus NoCs, the wraparound channel is used only for communication between the boundary routers because the use of wraparound channel will not reduce the hop count for any other pairs of source and destination. One such scenario is shown in

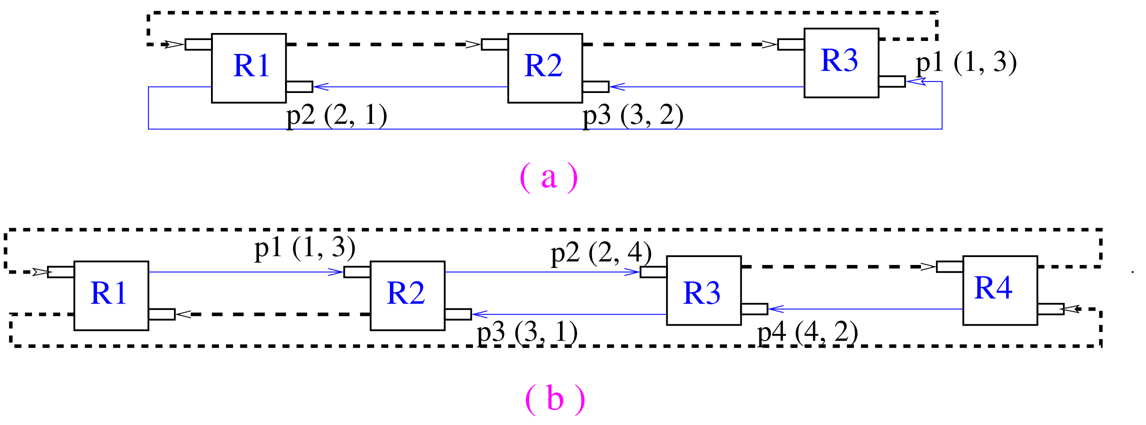


Figure 5.4: *Wraparound channels in a single row of a* 3x3 *and* 4x4 *Torus NoC*

Fig. 5.4(a). Here, wraparound channel is used for sending packet from router R1 to router R3. Therefore, it would not create any cyclic dependency. Similarly for a 4x4 Torus NoC, the wraparound channel is used only for communication between boundary routers R1 and R4 as shown in Fig. 5.4(b). Any communication to R2 and R3 would not take the wraparound channel. Therefore, there is no deadlock in 4x4 Torus NoC as well. The deadlock scenario arises only when a packet takes the wraparound channel and then moves backward in the same row/column. This case arises when communication to the router which are not in the boundary takes the wraparound channel. In Fig. 5.2(a), the packet p5(5, 2) takes the wraparound channel and then needs to move backward from R1 to R2. This observation is concluded in Lemma 5.1.1. We, therefore, consider NoC of size 5x5 or higher in all of our experiments.

Lemma 5.1.1. *Deadlock due to cycle path involving wraparound channel is possible in a* NxN *Torus NoC iff* $N \geq 5$.

5.2 Motivation and Contribution

In this section we describe our motivation for the work on deadlock avoidance in Torus NoC. We also highlight the contributions from this chapter.

5.2.1 Deadlock Representation in Torus NoC

For a given source and destination addresses in a Torus NoC, there presence alternate paths for both X-direction and Y-direction due to the presence of cyclic paths via wraparound channel. The first step for XY-routing in Torus NoC is to decide whether wraparound channel is applicable or not. Each of the X-direction and Y-direction of a Torus NoC can be traversed either in clockwise or in anti-clockwise direction. Therefore, the first step is to determine the clock wise or in anti-clock wise path for both X-direction and Y-direction to obtain the shortest or the preferred path [3]. Then a packet traverse using XY-routing.

Whether wraparound channel is taken or not, in either cases, the packet has to traverse through a path that belongs to an inherent a cyclic path. Therefore, deadlock freedom for XY-routing in Torus NoC cannot concluded with help of the Turn model because the behaviour of wraparound channel cannot be covered with help of the given set of Turns in the Turn model. Therefore, the deadlock scenario in Fig. 5.2(a) cannot be represented with help of the turns shown in Fig. 5.3. The method of representing resource dependency with help of CDG is helpful for all scenarios including scenarios for Torus NoC. The deadlock scenario of Fig. 5.2(a) is represented in Fig. 5.3(b) with help of CDG. Presence of the Turn information and wraparound channel information in a deadlock scenario are helpful to take needed measure for avoiding that deadlock. Whereas, there is no wraparound channel in the CDG representation. Therefore, we propose a Directional Dependency Graph (DDG) in this chapter incorporating Turn information, wraparound channel information and cycle information.

5.2.2 Deadlock Avoidance in Torus NoC

Inherent cyclic path via wraparound channel in each row and column of a Torus NoC makes the topology deadlock prone. Such a deadlock scenario is demonstrated with an example in Fig. 5.5(a). For avoiding deadlock in Torus NoC, one approach is to create a spanning tree corresponding to a given topology. In a spanning tree, there exist no cyclic path. Deadlock is avoided in Up*/Down* routing algorithm by following only the available paths permitted by a spanning tree [10]. Many edges (channels connecting routers) remain absent in the spanning tree and such channels are remained unused in that approach. In another approach, a packet is permitted to use the wraparound channel only during its first hop so that the cyclic resource dependency can be discontinued [1]. Many packets are deprived from using wraparound channel if the condition of using wraparound channel only at its first hop does not satisfy. Therefore, the leverage of wraparound channels cannot be used for most of the packets in this approach.

There are other approaches that uses additional resources to avoid deadlock. *Virtual Channel (VC)* is commonly used for designing deadlock free routing algorithm in Torus [2, 11, 27] to overcome the inherent cyclic dependencies. However, VCs have overheads in terms of system resource usage and power consumption. Alternatively, *bubble flow control* is also used for avoiding deadlock in Torus NoC with the help of additional buffer [28, 29]. In this method, injection of new packet or change of direction of a packet is restricted based on availability of an extra buffer.

To the best of our knowledge, there is no other approach available for designing deadlock free routing algorithms for Torus NoC without using additional resources. The question that motivates us is *"can we develop a deadlock avoidance approach without using any additional buffer or VC for Torus NoC that reduce overall hop counts by increasing utility of wraparound channels?"*.

5.2.3 Contributions

The motivation of the work in this chapter is to avoid deadlock in Torus NoC without using any additional buffer or VC and at the same time increase utility of wraparound channels. For achieving that an Arc model is proposed. A *Directional dependency graph* (DDG) is also proposed for deadlock representation and deadlock detection while using Arc model with a set of permitted Turns in the mesh sub-network. Specifically, the contributions of this chapter are as follows:

- An Arc model is proposed for avoiding deadlock in Torus NoC due to cyclic paths via wraparound channels.

- A Directional Dependency Graph (DDG) is proposed by combining Dally's CDG with Turn model. DDG is useful for deadlock representation, deadlock detection, formulating deadlock avoidance and showing deadlock freedom in Torus NoC.

- The applicability of our proposed Arc and DDG are shown with respect to XY-turns. Behaviour of XY-Turns with respect to different Arcs are demonstrated in this work.

5.3 The Arc Model for Avoiding Deadlock in Torus

In this section, we have proposed the Arc model by restricting certain move after using the wraparound channel to avoid deadlock in Torus NoC.

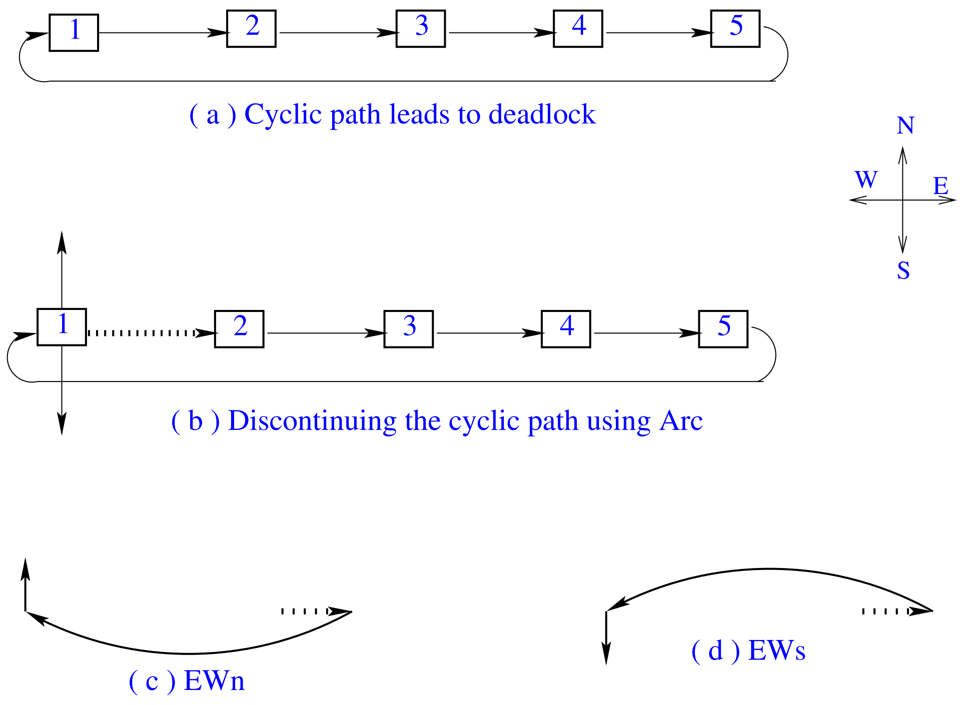

1
2
3
4
5
(a) Cyclic path leads to deadlock
N
W
E
S
1
2
3
4
5
(b) Discontinuing the cyclic path using Arc
(c) EWn
(d) EWs

Figure 5.5: *(a) Potential Deadlock, (b) Deadlock Avoidance, (c) EWn Arc, (d)EWs Arc.*

5.3.1 Restricted Move via Wraparound Channel

The cyclic path through wraparound channel has potential for deadlock. Such an example for deadlock via wraparound channel is already shown in Example 5.1 and Fig. 5.2(a).

For forming a cyclic buffer dependency, the source and destination pairs for each packets involved in that dependency should be at least two hops away [2]. A cyclic path via East to West (EW) wraparound channel is shown in Fig. 5.5(a). *To break that potential cycle, the immediate backward movement just after taking the EW wraparound channel is prohibited in this work.* Two alternate directions either towards North or South are permitted after the EW wraparound channel. The restricted movements after the EW wraparound channel are shown in Fig. 5.5(b). The EW wraparound channel is sub-divided into EWn and EWs Arcs that are shown in Fig. 5.5(c) and Fig. 5.5(d), respectively.

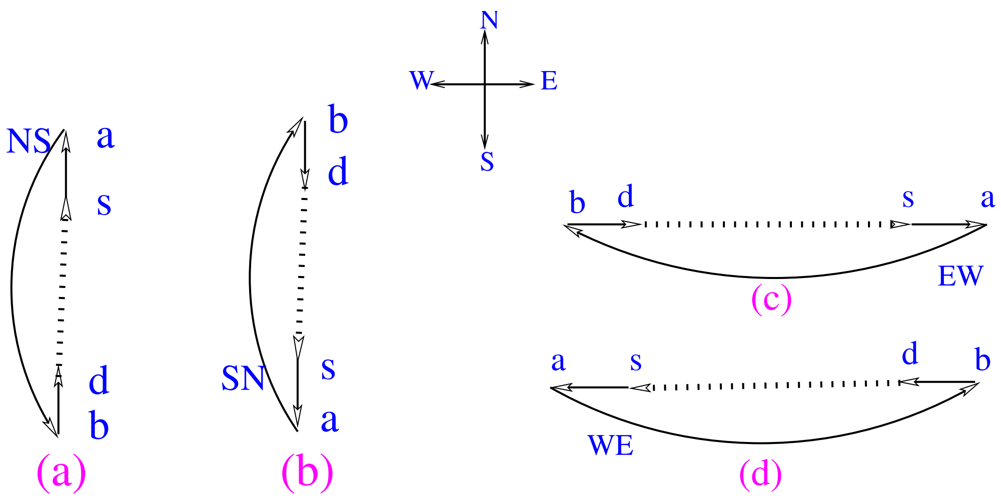


Figure 5.6: *Wraparound channels: (a) NS, (b) SN, (c) EW, (d) WE*

5.3.2 Classification of Wraparound Channels

The wraparound channels in a Torus NoC are classified into four categories in this thesis based on their direction. The wraparound around channels from the North to the South boundary (NS), from the South to the North boundary (SN), from the East to the West boundary (EW) and from the West to the East boundary (WE). In short from, we denote them as NS, SN, EW and WE, respectively. The obvious cyclic paths via wraparound channels NS, SN, EW and WE are presented in Fig. 5.6(a), Fig. 5.6(b), Fig. 5.6(c) and Fig. 5.6(d), respectively. For all the wraparound channels in Fig. 5.6, a possible path is $s \to a \to b \to d \to s$. A sequence of packets present in that path, holding and waiting on each other for the buffer, might lead to a deadlock. One possible way to avoid that possible cyclic resource dependency is to avoid the backward moves $b \rightarrow d$ just after taking the wraparound channel. For achieving this, an *Arc model* is proposed in this work to design deadlock free routing algorithms for Torus NoC without using additional resources.

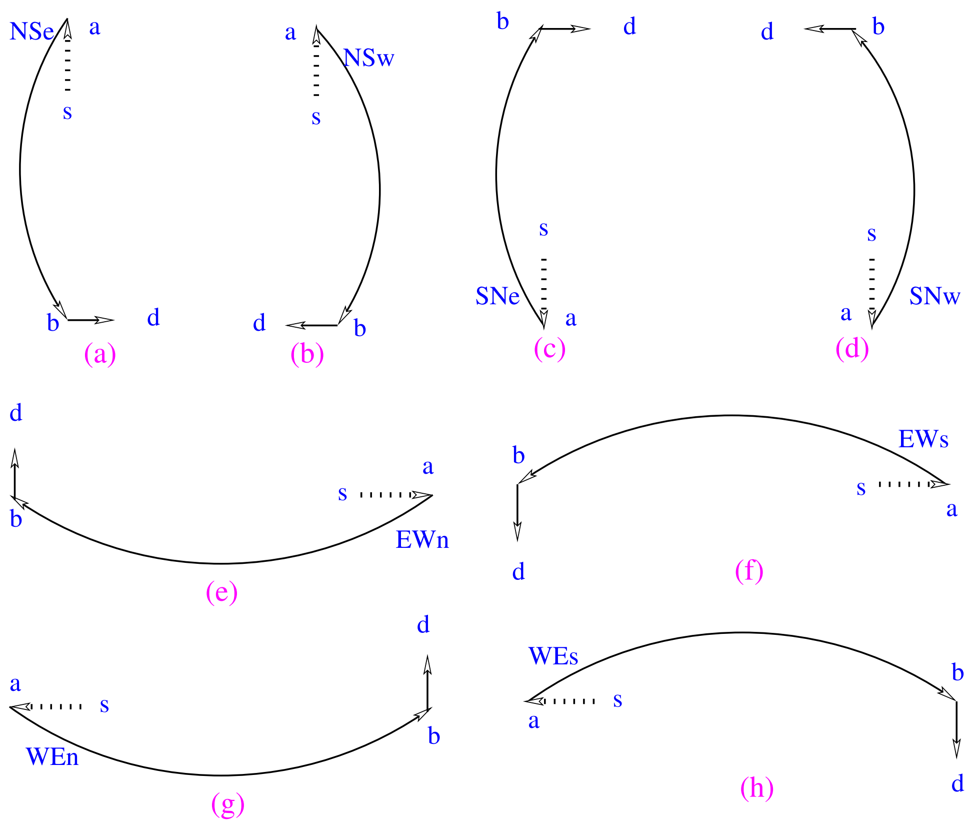


Figure 5.7: *Arc model: Eight possible Arcs in Torus NoC*

5.3.3 The Proposed Arc Model

Wraparound channels are used to reach from one boundary row to another boundary row or from one boundary column to another boundary column. The backward move $b \rightarrow d$ after taking the wraparound channel in Fig. 5.6 is restricted to avoid deadlock in the proposed *Arc model*. On reaching the opposite boundary row (/column) via a wraparound channels, the packet moves in the same boundary row (/column) for at least one hop. Only after that point, the packet can take a backward move if it is required. As a result the obvious deadlock prone cycle is discontinued. In the proposed Arc model, each wraparound channel is divided into two Arcs. For example, the NS wraparound channel in Fig. 5.6(a) is divided into NSe and NSw *Arcs*, as shown in Fig. 5.7(a) and Fig. 5.7(b), respectively. Essentially, after taking the NS wraparound channel, the packet will move one hop either in E or W direction. In a similar way, for breaking the cycle in all the wraparound channels in Fig. 5.6, we have categorised the wraparound channels into eight *Arcs*. The eight possible *Arcs* in our proposed *Arc model* are shown in Fig. 5.7.

NSe *Arc* is applicable if NS wraparound channel helps to reduce hop count and the destination is in the South-East direction. A situation of another consecutive EW wraparound channel just after the NSe Arc does not arises due to the one hop movement towards the East in the NSe Arc. It is ensured or taken care of by the condition for the applicability of NSe *Arc*. Similarly, NSw *Arc* is applicable if NS wraparound channel reduces hop count and destination is in South-West direction. The condition for the applicability of NSw *Arc* ensures that the immediate movement towards W after the wraparound channel does not lead to another wraparound channel WE. The conditions for the applicability of *Arcs* prevent all such scenarios of taking two wraparound channels consecutively.

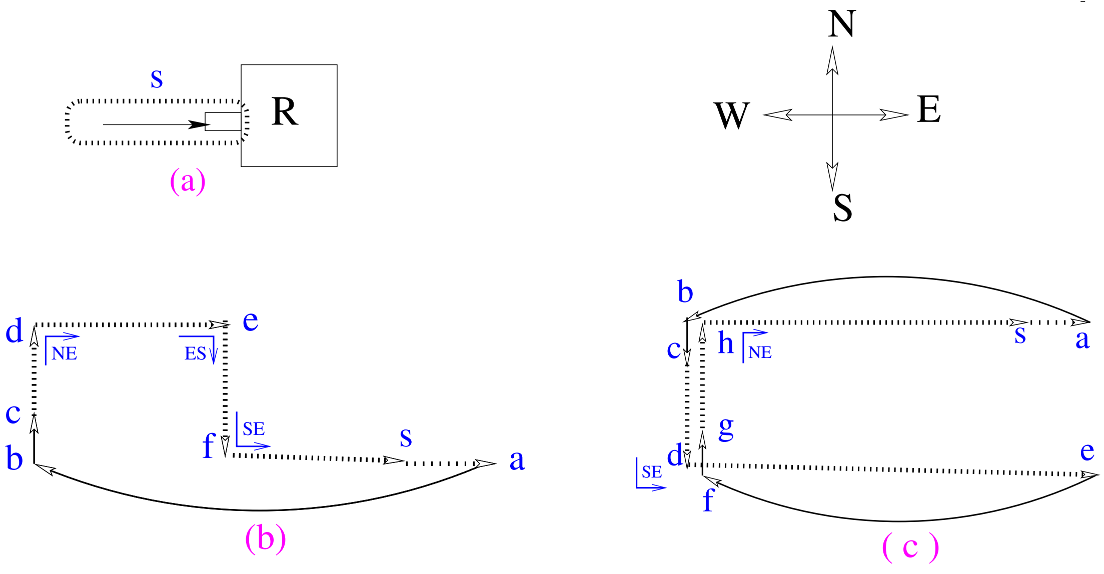


Figure 5.8: *(a) Channel-buffer combination as a vertex, (b) Dependency cycle with EWn, (c) Dependency cycle with EWn and EWs.*

5.3.4 Effect of Arcs with the Permitted Turns in the Mesh Sub-network

The Arc model imposes a restriction to mitigate the immediate deadlock potential via cyclic paths through wraparound channels. Though the most potential deadlock cycle is broken, still there are chances for complicated deadlock cycle depending upon the permitted Turns in the Mesh sub-network. Such cycles are not so straight forward and they might spread across a large number of routers. Fig. 5.8(b) and Fig. 5.8(c) show two possible deadlock cycles. Arcs and Turns involved in the cycle formation are also highlighted. The Arcs are distinguished with a curved arrow. The name of the Turns involved are mentioned in the diagram. Each alphabet in the graphs represents a vertex that connects two edges in the graph. We have not separately put a small circle to represent the vertex for the sake of clarity of the diagram. A vertex represents a channel-buffer combinations as shown in Fig. 5.8(a). A dotted lines are used to represent many intermediate channel-buffer combinations in the same directions where no Turns are involved.

In Fig. 5.8(b), a sequence of packets form a resource dependency cycle across the path $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow s$. Each vertex in the graph is a channel-buffer combination as like CDG [2], i.e., a vertex represents a channel connected to an input port buffer in the adjacent router. For example, vertex s represents a channel from West to East coupled with the West input port buffer in the adjacent router which is shown in Fig. 5.8(a). Similarly, vertex a in Fig. 5.8(b), is another West input port channel-buffer. Dotted line between s and a indicates some more such West input port channel-buffer involved in the dependency cycle. Vertex b represents another West input port buffer coupled with a EW wraparound channel. Vertex c represents the adjacent South input port channel-buffer. A solid-line indicates adjacent channel-buffer. Whereas, a dotted-line indicates dependency across a series of channel-buffers. Vertex d is another South input port channel-buffer from where dependency cycle spread with a NE Turn. The dependency keeps on spreading in this manner and finally reaches the starting vertex s to complete the cycle.

In the second cyclic dependency example in Fig. 5.8(c), two Arcs namely EWn and

EWs and two Turns namely NE and SE are involved. For formal representation of such dependency cycles Directional Dependency Graph (DDG) is proposed in the next section.

5.4 Directional Dependency Graph

Restricting either NE or SE Turn is sufficient to avoid the cyclic dependency in the example shown in Fig. 5.8(c). Since it is possible to avoid deadlock by restricting certain Turn involved in the formation of a deadlock cycle, correct representation of such cycle along with Turn information is important. In Turn model approach, it is not mentioned *how to represent deadlock through wraparound channel.* Using CDG approach, though it is possible to represent the deadlock scenarios via wraparound channel (Fig. 5.8(a) and Fig. 5.8(c)), additional information regarding Turns and directions are absent. To meet these requirements, we proposed DDG inherited from both Turn model and CDG approach. Main characteristics that distinguish DDG from Turn model and CDG are,
(1) Special representation for wraparound channel (addition to Turn model),
(2) To make it more informative, Turn and direction information are incorporated along with channel-buffer node (addition to CDG),
(3) To make it simple, some intermediate vertices can be eliminated (dotted line) if there is no change in direction and wraparound channel is not involved (addition to CDG).

All resource dependency and deadlock cycles in rest of this chapter are represented using DDG. Formal definition of DDG is presented below.

Definition 5.4.1. ***Directional Dependency Graph (V, E)*** *represents a possible path permitted by a routing algorithm with direction information. Each vertex in V represents a channel coupled with its connected buffer. There are three types of edges in E:*
i) A dotted edge with arrow indicates that there is no change in direction and there may be many intermediate vertices in between.
ii) A solid edge with arrow indicates there is a path between the two adjacent vertices permitted by the routing algorithm.

iii) A curved edge with arrow indicates the path through wraparound channels between two adjacent vertices.

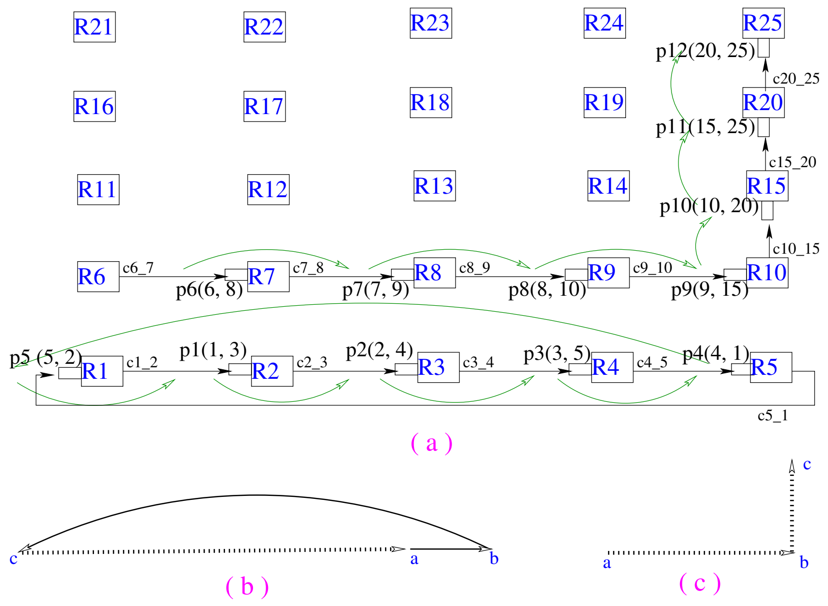


Figure 5.9: *(a) Resource dependency for a set of packets (source and destination are shown in the bracket, b) Directional Dependency Graph representing a deadlock scenario, (c) Directional Dependency Graph representing a resource dependency scenario.*

Example 5.2. The resource dependency scenario of Fig. 5.2(a) is given again for convenience in Fig. 5.9(a). Both of them represent the same resource dependency situation. The deadlock scenario of Fig. 5.9(a) is shown using the directional dependency graph in Fig. 5.9(b). In Fig. 5.9(b), point a, b and c represents the channel c3_4, c4_5 and c5_1 of Fig. 5.9(a), respectively. In Fig. 5.9(c), a directional dependency without deadlock is shown. Here, point a, b and c represents the channel c6_7, c10_15 and c20_25 (Fig. 5.9(a)), respectively. The main advantage of the DDG in Fig. 5.9(b) over the CDG in Fig. 5.2(b) is, the DDG is a simpler representation as it uses less number of vertices. Though both the DDG and CDG represent a deadlock cycle, the DDG also incorporates the wraparound channel information which could be useful for deadlock avoidance. In the other example, the DDG in Fig. 5.9(c) and the CDG in Fig. 5.2(c) represents same resource dependency. The DDG uses only three vertices instead of six vertices in case of the corresponding CDG. Moreover, the EN Turn information is attached in the DDG of Fig. 5.9(c) which is absent in the CDG of Fig. 5.2(c).

All the graphs in Fig. 5.6, Fig. 5.7 and Fig. 5.8 that shows wraparound channels and Arcs in Section 5.3 are represented using DDG.

5.4.1 Application of Arc Model and DDG

For developing deadlock free routing algorithms for Torus NoC without using additional buffer or VC, *the Arc model along with a set of permitted Turns in the Mesh Sub-network are used. DDG is useful in this process for deadlock detection, deadlock avoidance and for showing deadlock freedom for the given combination of Arcs and Turns.* A DDG has different applications:

1. First application of DDG is in *deadlock detection.* If a clockwise or anti-clockwise cycle from a given set of Arcs and Turns are possible, the combination is deadlock prone.
2. Its second application is in *deadlock-avoidance* by restricting or distributing certain Turns involved in a deadlock cycle [90, 106].
3. Another application of DDG is its usability in showing *deadlock freedom.* A given combination of Turns and Arcs are deadlock free if neither clockwise nor anti-clockwise cycle formation are possible.

In this thesis, we consider XY-Turns in the Mesh sub-network (i.e., EN, ES, WN, WS Turns in Fig. 5.3(c)) with Arcs and put forth deadlock detection, deadlock avoidance and deadlock freedom for different Arc combinations.

5.5 Case Study: Arcs with XY-Turns

The XY-routing is a deterministic routing which is free from deadlock and livelock [107]. The routing is called XY-routing as it uses only XY-Turns in a Mesh NoC. There are four XY-Turns, i.e., EN, ES, WN and WS Turns as shown in Fig. 5.3(c). In a Mesh NoC, traversal within a row is considered as X-direction and traversal within a column is considered as Y-direction. In XY-routing, all packets are first routed in X-direction. Once the destination column is reached, the packet is routed in the Y-direction until the destination router is reached. In this section we are analysing deadlock behaviour of individual Arc and Arc pairs with respect to XY-Turns in Torus NoC.

5.5.1 Single Arc with XY-Turns

Torus NoC is considered as a combination of wraparound channel and Mesh sub-network in this thesis. First we consider each Arc individually with respect to XY-Turns.

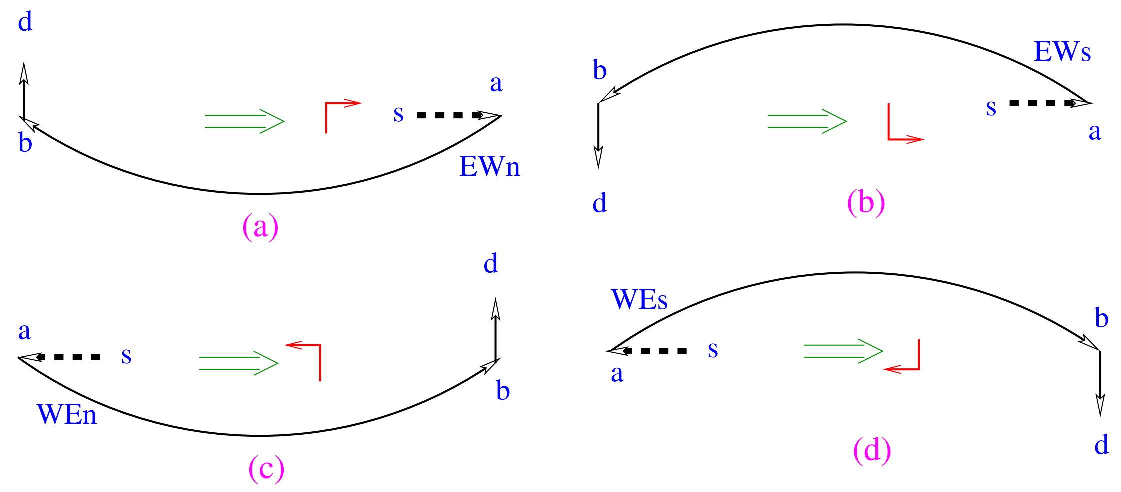


Figure 5.10: *Additional Turns introduced with respect to XY-Turns a) EWn Arc adds NE Turn, b) EWs Arc adds SE Turn, c) WEn Arc adds NW Turn, and d) WEs Arc adds SW Turn*

5.5.1.1 Turns Introduced due to Arcs

In case of NSe *Arc*, after taking the NS channel the packet has to move immediately one hop distance in East direction. After that, XY-routing is used. No additional Turn is used in this case because the packet is moving in X-direction first followed by Y-direction and XY-routing is considered as the background routing algorithm. Following the NS wraparound channel, the movement of the packet is similar to XY routing in Mesh sub-network. For the same reason, all *Arcs* with Y-wraparound channels namely, NSe, NSw, SNe, and SNw, do not add any extra Turns with respect to XY-routing.

The case is different for *Arcs* EWn, EWs, WEn, WEs with X-wraparound channels. After taking the X-wraparound channel, a packet has to move in Y-direction first, then it is allowed to follow XY-routing. Therefore, new YX-Turns are enabled. For example, if destination is in the West-North direction and EW wraparound channel is applicable, EWn *Arc* (Fig. 5.10(a)) is to be used. After taking EW wraparound channel, the packet then moves single hop Y-distance (*towards North*). After that, the packet follows XY-routing. It will cover X-distance (*towards East*) first followed by Y-distance. The movement following the wraparound channel introduces one additional *North-East* Turn. *North-East* Turn is restricted in XY-routing. The newly added *North-East* Turn is depicted with red color in Fig. 5.10(a). Similarly, for all the other *Arcs* in X-direction, one new Turn is added. The new Turns are shown using red color in Fig. 5.10. It may be noted that the newly introduced Turns by the *Arc model* is applicable only at the corresponding boundary. These Turns are not permitted in other places of Mesh sub-network.

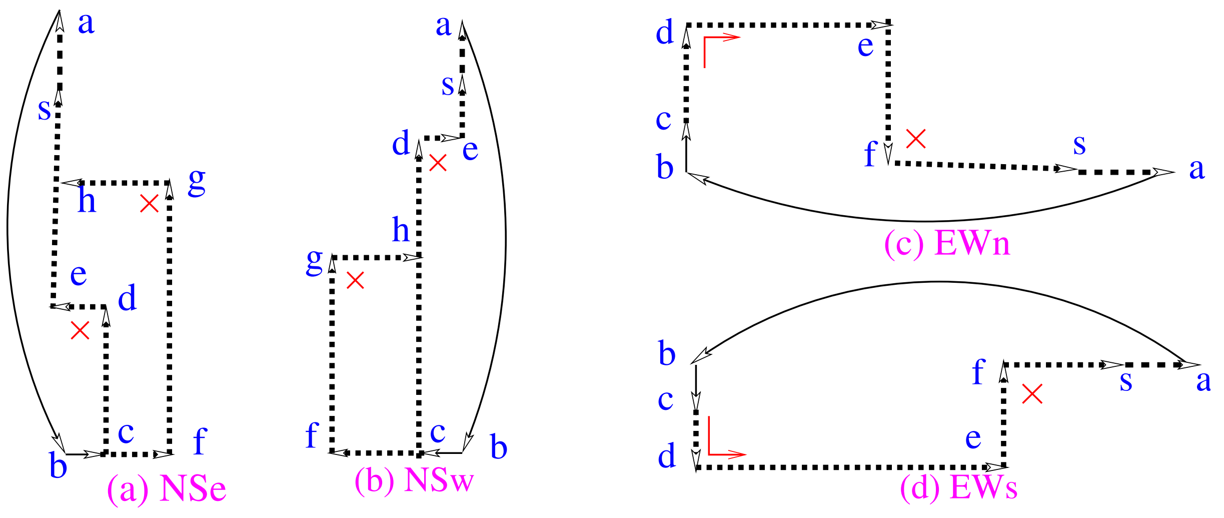


Figure 5.11: *Deadlock freedom for (a) NSe, (b) NSw, (c) EWn, (d) EWs Arcs with XY-Turns*

5.5.1.2 Deadlock Freedom for Individual Arc

Since XY-Turns are deadlock free, a possibility for deadlock cycle using Arcs and Turns need to check. One DDG for NSe Arc with XY-Turns are shown in Fig. 5.11(a). Since, clockwise cycles are not possible involving NSe Arc, anti-clockwise cycles are shown in Fig. 5.11(a). Next step is to justify a cycle with the set of XY-Turns. For a valid cycle, a NW Turn is needed at some point d or g. The NW is a YX-Turn and is restricted in this case. Therefore, deadlock cycle is not possible for NSe Arc with XY-Turns and the combination is deadlock-free. A DDG for NSw Arc is shown in Fig. 5.11(b). For deadlock occurrence a

NE Turn is needed at some point d or g. Since NE Turn is restricted, NSw Arc is deadlock free with respect to XY-Turns. In similar way, deadlock freedom for SNe and SNw can be shown.

A DDG for EWn Arc is shown in Fig. 5.11(c). A YX-Turn NE is enabled at some vertex d. Still we need another YX-Turn SE at some vertex f to complete a dependency cycle. Therefore, deadlock is not possible for EWn Arc with XY-Turn as the SE Turn is restricted in the routing. A similar DDG for EWs Arc with XY-Turns are shown in Fig. 5.11(d). Deadlock is not possible in this case as the NE Turn is restricted which is shown at vertex f. In similar way, deadlock freedom for WEn and WEs Arcs can be shown using DDG. In each case, cycle formation is not possible as some Turns are restricted that can form a deadlock cycle. Our proposed DDG representation in Fig. 5.11 help us in proving deadlock freedom of individual Arc with respect to XY-Turns.

5.5.2 Deadlock Detection for Arc Pairs with XY-Turns

Though all eight *Arcs* are individually deadlock free with XY-routing in the Mesh sub-network, while using two *arcs* at a time, their resultant effects might results in a deadlock. In this section, we identify such deadlock prone *Arc* pairs and eliminate such pairs from using in a routing algorithm. From eight possible *Arcs*, total $\binom{8}{2} = 28$ such pairs are possible. Out of them, 14 pairs are identified as deadlock prone.

5.5.2.1 Deadlock in Arc Pairs from the same Wraparound Channel

Each wraparound channel in Torus NoC is divided into two *Arcs* in the *Arc model*. For example, EW wraparound channel is divided into EWn and EWs *Arcs*. If two such *Arcs* are used in the same routing algorithm, they create a deadlock scenario. This is presented in Lemma 5.5.1. The validity of this Lemma is shown using directional dependency graph.

Lemma 5.5.1. *Using of two Arcs from the same wraparound channel along with XY-turns in the Mesh sub-network in a routing algorithm lead to deadlock.*

From the four wraparound channel four such *Arc* pairs are possible. These pairs are (EWn + EWs), (WEn + WEs), (NSe + NSw) and (SNe + SNw).

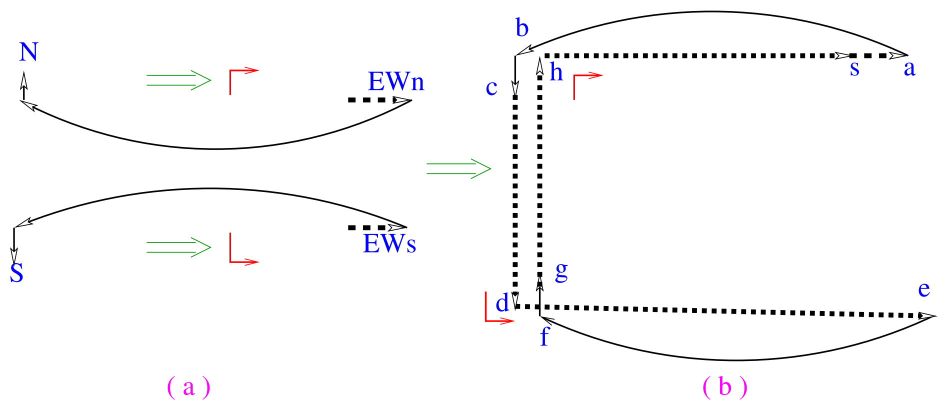


Figure 5.12: *a) EWn and EWs Arcs, b) Deadlock Scenario with EWn Arc, EWs Arc and XY-Turn*

Considering EWs and EWn *Arcs* with XY-routing, they introduce two additional Turns SE and NE that are shown with red color in Fig. 5.12(a). These two additional turns do not create any cycle with the permitted Turns in XY-routing (Fig. 5.10(a)). Hence, there is no deadlock as per Turn model. Next we are checking using DDG if any dependency cycle is possible. One possible DDG cycle is shown in Fig. 5.12(b). A deadlock cycle through vertices $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow s$ are shown. The cycle is created using two *Arcs* and the SE and NE Turns introduced by the *Arcs* in the West boundary by a sequence of packets. SE Turn is introduced at d by a EWs *Arcs* that takes place in an adjacent vertex. Similarly, NE Turn is introduced by a EWn *Arc*. In a similar way, deadlock is possible for (WEn + WEs) with XY-routing.

Let us consider the directional dependency graph with two *Arcs* NSe and NSw from a same wraparound channel in Fig. 5.13. NSe and NSw *Arcs* do not introduce any additional Turns. A cycle formation is possible as shown in Fig. 5.13 with vertices $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow s$ by a sequence of packets. Here, vertices b and f represents wraparound channel. The EN Turn at vertex b is permitted by XY-routing. From vertex c, dependency spread across d, e and reach vertex s. The WN Turn at vertex e is permitted by XY-routing. Thus, it is possible to form a cyclic dependency or deadlock using NSe and NSw *Arcs* with XY-routing. In the same manner, deadlock is possible for (SNe + SNw)

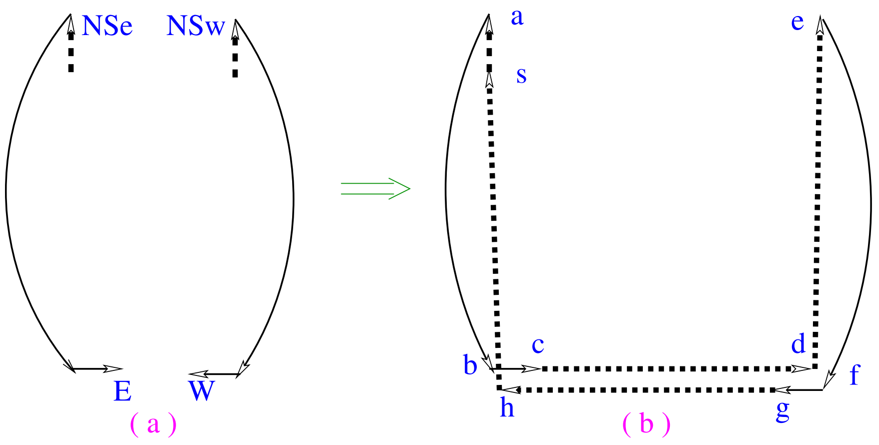


Figure 5.13: *a) NSe and NSw Arcs, b) Deadlock Scenario due to combination of NSe and NSw with XY-routing.*

5.5.2.2 Deadlock due to Added Turns by Arcs

Deadlock is possible if new Turns are introduced by the considered *Arcs* and they form a cycle with the permitted Turns in the Mesh sub-network. Two such *Arc* combinations are found with respect to the XY-turns in the Mesh sub-network. These two *Arc* pairs are (EWs + WEn) and (EWn + WEs). For the (EWs + WEn) *Arc* pairs, EWs *Arc* introduces SE Turn and WEn *Arc* introduces NW Turn. These two Turns along with the XY-Turns create a cycle in the anti-clockwise direction. The XY-Turns are shown in Fig. 5.3(c). Similarly, (EWn + WEs) introduces NE and SW Turns, respectively. They also create a cycle with XY-routing in the clockwise direction. Therefore, *Arc* combinations (EWs + WEn) and (EWn + WEs) are deadlock prone according to first condition.

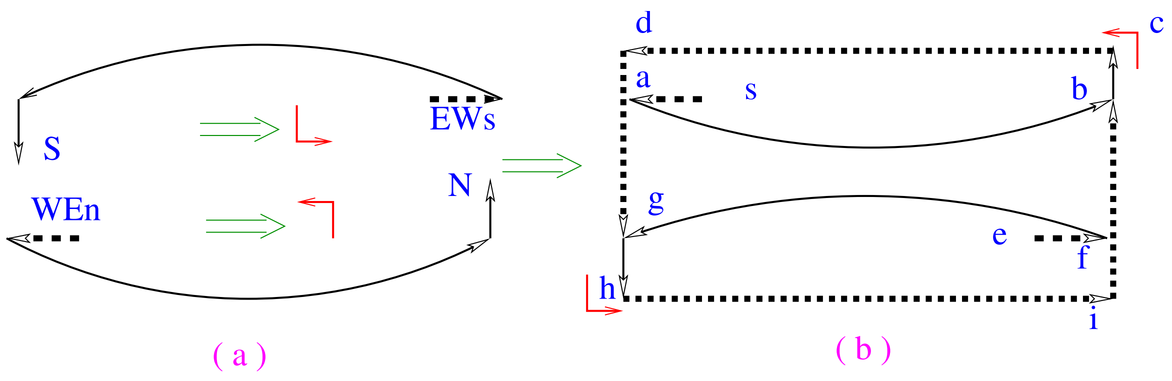


Figure 5.14: *a) EWs and WEn Arcs, b) Deadlock Scenario due to combination of EWs and WEn*

Turn introduced by (EWs + WEn) and a corresponding deadlock scenario with XY-routing are given in Fig. 5.14(a) and Fig. 5.14(b), respectively. The deadlock cycle is through the vertices $c \rightarrow d \rightarrow h \rightarrow i \rightarrow c$. Though, wraparound channels b and g are not directly included in the cyclic dependency, they enable NW Turns at the East boundary and SE Turn at the West boundary which lead to deadlock. Similar kind of dependency cycle formation is possible for (EWn + WEs) with XY-routing.

5.5.2.3 Deadlock with a Combination of X-Arc and Y-Arc

So far, the *Arcs* whose primary direction are either in X-direction (EW/WE) or Y-direction (NS/SN) are considered for finding deadlock pair. If one *Arc* in X-direction and another *Arc* in Y-direction are considered with XY-routing, Lemma 5.5.2 is applicable for determining deadlock pair.

Lemma 5.5.2. *If one X-Arc and one Y-Arc are used with XY-routing, there are two types of Arc combinations that have potential deadlock:*

- *Arc combination1: Xn and NSx*
- *Arc combination2: Xs and SNx*

Here, X = EW or WE, x = e or w. For any value of X (EW/WE) and x (e/w), deadlock is possible for these two types of Arc combinations.

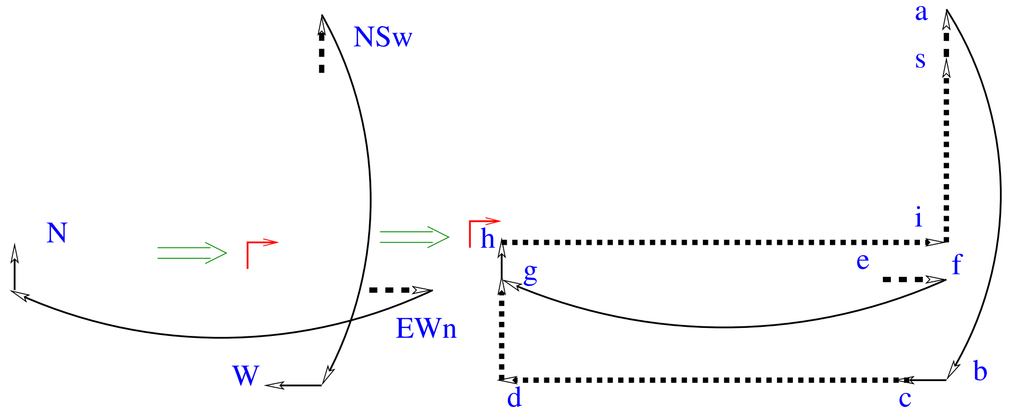


Figure 5.15: *(a) EWn and NSw Arcs (b) Deadlock Scenario due to combination of EWn and NSw*

There are eight possible *Arc* pairs with deadlock potential that fall under Lemma 5.5.2. These pairs are namely, (EWn + NSw), (EWn + NSe), (EWs + SNw), (EWs + SNe), (WEn + NSe), (WEn + NSw), (WEs + SNe), and (WEs + SNw). As an example, deadlock scenario for (EWn + NSw) is shown with dependency graph in Fig. 5.15. There is a possible cycle through the vertices $s \to a \to b \to c \to d \to h \to i \to s$. Here, b is a wraparound channel present in the cycle and g is another wraparound channel that enables NE Turn but not directly present in the cycle. Deadlock scenario in the rest of the seven pairs are also possible in a similar way.

In this chapter, total fourteen pairwise deadlock prone *Arcs* pairs are found out. We have summarised the pairwise deadlock prone Arcs with respect to XY-Turns and the reason for deadlock in Table. 5.1. Remaining fourteen *Arcs* pairs out of twenty eight *Arcs* pairs do not create deadlock using Lemma 5.5.1 or Lemma 5.5.2 or due to Turns introduced by *Arcs*. Analysis for their deadlock freedom using DDG are given in Subsection 5.5.4.

Table 5.1: *Deadlock prone and deadlock free Arc pairs with respect to XY-routing*

Deadlock prone Arc pairs		**Deadlock free Arc pairs**	
Arcs	**Reason**	**Arcs**	**Reason**
SNw + SNe	Lemma 5.5.1	SNw + NSw	No cycle in DDG
NSw + NSe	Lemma 5.5.1	NSe + SNe	”
EWs + EWn	Lemma 5.5.1	EWn + WEn	”
WEs + WEn	Lemma 5.5.1	EWs + WEs	”
EWs + WEn	Added Turns	SNw + NSe	”
WEs + EWn	Added Turns	NSw + SNe	”
EWn + NSe	Lemma 5.5.2	EWn + SNe	”
EWn + NSw	Lemma 5.5.2	EWn + SNw	”
EWs + SNe	Lemma 5.5.2	EWs + NSe	”
EWs + SNw	Lemma 5.5.2	EWs + NSw	”
WEn + NSe	Lemma 5.5.2	WEn + SNe	”
WEn + NSw	Lemma 5.5.2	WEn + SNw	”
WEs + SNe	Lemma 5.5.2	WEs + NSe	”
WEs + SNw	Lemma 5.5.2	WEs + NSw	”

5.5.3 Deadlock Avoidance using DDG Representation

Representing a deadlock situations using DDG helps us in understanding the root cause of deadlock. Because, a DDG highlights all the Turns used and the wraparound channels involved if any. By restricting such a Turn or wraparound channel, deadlock can be avoided. Therefore, DDG representation of deadlock cycles could act as useful input in formulating deadlock avoidance.

5.5.3.1 Deadlock avoidance for (EWs + WEn) Arcs with XY-Turns

For the dependency cycle using (EWs + WEn) Arcs in Fig. 5.14(b), one solution for deadlock avoidance is to restrict WS Turn at vertex d (or EN Turn at vertex i) and break the cyclic dependency. Since, dependency cycle forms only via boundary column in Fig. 5.14(b), it is not necessary to restrict WS Turn across the Mesh. It is better to avoid deadlock for (EWs + WEn) Arc pair by restricting WS Turn across West boundary column only. This is known as Turn distribution approach [106]. Moreover, it is feasible to add additional Turns besides XY-Turns with (EWs + WEn) Arcs safely. Turn distribution would be beneficial for such deadlock avoidance and for adding additional Turns safely [90, 106].

5.5.3.2 Deadlock avoidance for (NSw + EWn) Arcs with XY-Turns

A DDG representation of deadlock scenario for (NSw + EWn) Arcs with XY-Turns is shown in Fig. 5.15. A YX-Turn, i.e., NE Turn at vertex h, is introduced. An EN Turn, which is an XY-Turn, is also involved at some vertex i. One possibility for avoiding deadlock in this scenario is to restrict the EN Turns. Thus, instead of using all XY-Turns, permitting only ES, WN and WS Turns with (NSw + EWn) Arcs would make the algorithm deadlock free. Thus by introducing one YX-Turn (NE Turns) and by restricting one XY-Turn (EN) deadlock freedom for the algorithm can be maintained. I may be possible to add more Turns to the combination of (NSw + EWn) Arcs with XY-Turns with a careful CDG analysis.

Deadlock avoidance could be applied for all deadlock prone Arc pairs in a similar way. Information from a deadlock scenario represented using DDG could be useful even for applying virtual channel (VC) approach to avoid deadlock. In that approach, instead of restricting a Turn for breaking the cycle, a separate class of VC needs to be used on taking that Turn [2,27].

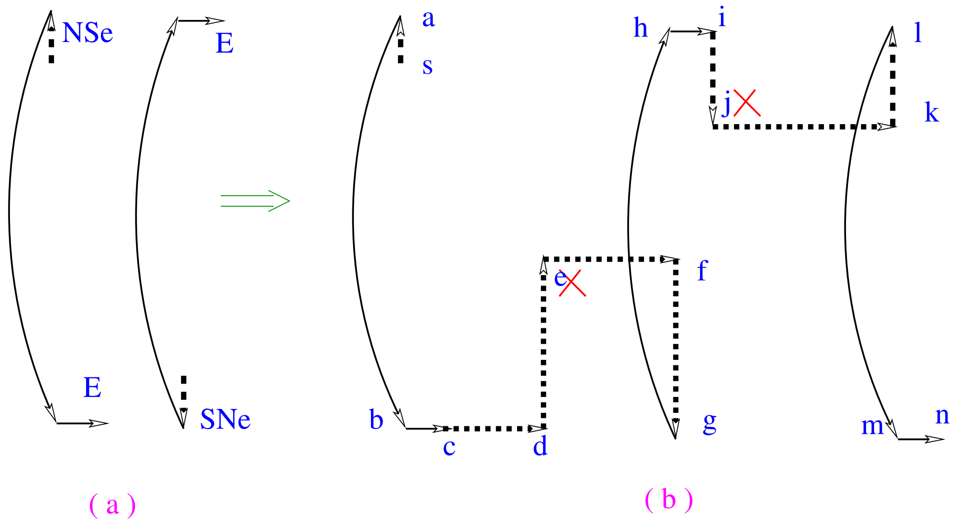


Figure 5.16: *(a) SNe and NSe Arcs (b) Dependency graph: deadlock is not possible by SNe and NSe Arcs*

5.5.4 Checking Deadlock Freedom using DDG

In a given topology, starting from a channel-buffer combination or a vertex in DDG, if the routing algorithm under consideration allows a packet to reach the same vertex (channel-buffer) via intermediate vertices, we conclude that the routing algorithm is deadlock prone. We have mentioned about a single packet just for the path discovery purpose. In real scenario, if such a cyclic path is possible, sequence of packets will present in the different buffers of that path to form an actual deadlock cycle. If reaching of the starting vertex is not possible even after trying all possible permitted path by the routing algorithm, we conclude that the algorithm is deadlock free on that topology. We are applying this principle for checking deadlock freedom using DDG in this thesis.

For (SNe + NSe) Arcs with XY-Turns, we try to form a channel-buffer dependency cycle using DDG in Fig. 5.16(b). SNe and NSe Arcs are individually deadlock free with XY-Turns as shown in Section 5.5.1.2. Whether, SNe and NSe Arc induces each other in cycle formation that need to be checked. As per the DDG of Fig. 5.16(b), two YX-Turns either NE(in vertex e) or SE(in vertex j) is needed even to join the two Arcs. It shows that cycle creation is not possible for (SNe + NSe) Arcs with only XY-Turns. Therefore, the combination is deadlock free. We perform such analysis for all Arc pairs using DDG. It is not possible to form any cyclic dependency with 14 Arcs pairs (deadlock-free Arc pairs) shown in Table. 5.1. Therefore, they are deadlock free.

We have analysed all the twenty eight Arc pairs with respect to XY-Turns. Using DDG, it is found that no cycle formation is possible for fourteen Arc pairs out off twenty eight possible Arc pairs. Therefore, these fourteen Arc pairs are deadlock free with respect to XY-Turns. The deadlock free Arc pairs are shown in Table. 5.1.

5.6 Experimental Deadlock Detection

In this section we check the validity of theoretical analysis presented in Subsection 5.5.2 and Subsection 5.5.4. For detecting experimental deadlock, our formally modeled CFSM based simulation framework presented in Chapter 4 is used [41]. A traffic sample of 100000 packets generated using a random number generating function (Random traffic) is applied to 5x5 Torus NoC. All experiments are performed in an Intel Core i5 3.20GHz, 8GB RAM machine.

5.6.1 Experimental Results for Arc Pairs with XY-Turns

In case of a deadlock situation, simulation reaches a state from where no transition is possible and simulation is aborted by reporting deadlock. A trace is also generated for such a case which depicts the snapshot for detailed cyclic resource dependency. For deadlock free Arc pairs, all packets get delivered and simulation completed. Out of 28 possible *Arc* pairs, all packets get delivered for 14 *Arc* pairs without reporting deadlock. Verification time for these deadlock free *Arcs* are shown in the right part of Table 5.2. In the case of *Arc* pairs with deadlock, verification time varies significantly as shown in Table 5.2. Only specific traffic patterns create deadlock to a particular algorithm. Therefore, it takes different time to come across traffic pattern that leads to deadlock. For (EWn + EWs) Arcs with XY-Turns, deadlock is detected in 183.17 seconds as shown in Table 5.2. This deadlock is correctly predicted using DDG in Fig. 5.12(b). For (EWs + WEn) Arcs, deadlock is detected in 2641.21 seconds. Deadlock is detected using DDG as well for the same Arc pairs (EWs + WEn) as shown in Fig. 5.14(b). In few cases, the deadlock scenario does not arises in the first attempt. We generate further random test cases and tried again. The deadlock scenario is produced within five attempts in all cases.

Table 5.2: *Deadlock detection for Arc pairs with XY-Turns*

Deadlock prone Arc pairs			Deadlock free Arc pairs		
Arcs	**Deadlock**	**Time (sec)**	**Arcs**	**Deadlock**	**Time (sec)**
SNw + SNe	Yes	31.62	SNw + NSw	No	4281.27
NSw + NSe	Yes	155.91	NSe + SNe	No	4093.74
EWs + EWn	Yes	183.17	EWn + WEn	No	4298.65
WEs + WEn	Yes	193.38	EWs + WEs	No	4532.07
EWs + WEn	Yes	2641.21	SNw + NSe	No	4609.10
WEs + EWn	Yes	930.88	NSw + SNe	No	4713.49
EWn + NSe	Yes	1655.73	EWn + SNe	No	4578.67
EWn + NSw	Yes	1556.76	EWn + SNw	No	4425.87
EWs + SNe	Yes	844.85	EWs + NSe	No	4705.92
EWs + SNw	Yes	830.87	EWs + NSw	No	4321.15
WEn + NSe	Yes	615.52	WEn + SNe	No	4473.03
WEn + NSw	Yes	836.80	WEn + SNw	No	4708.56
WEs + SNe	Yes	927.53	WEs + NSe	No	4345.90
WEs + SNw	Yes	734.35	WEs + NSw	No	4601.61

Similar experiments are performed using NoC of different grid sizes and using different traffic patterns containing more than 100000 packets with different injection rates generated from Booksim simulator [24]. All experiments lead to same conclusions regarding deadlock free and deadlock prone *Arc pairs*. Experimental results corresponding to 5x5 Torus NoC is presented here for convenience in representing the deadlock snapshot. If deadlock is not detected for a routing algorithm using a given traffic pattern, theoretically it does not guarantee deadlock freedom. Nevertheless, by using sufficiently large traffic with different mode, if deadlock is still not detected, it is likely that the routing is deadlock free. The theoretical analysis in Subsection 5.5.2 and the experimental results do not contradict in any cases. It indicates correctness of the deadlock detection using DDG.

5.6.2 Deadlock Scenarios generated by CFSM Framework

The CFSM based simulation framework has reported deadlock scenarios for all the fourteen deadlock prone Arc pairs. In this sub-section, we have presented only three experimental deadlock scenarios from the experiments in Table 5.2.

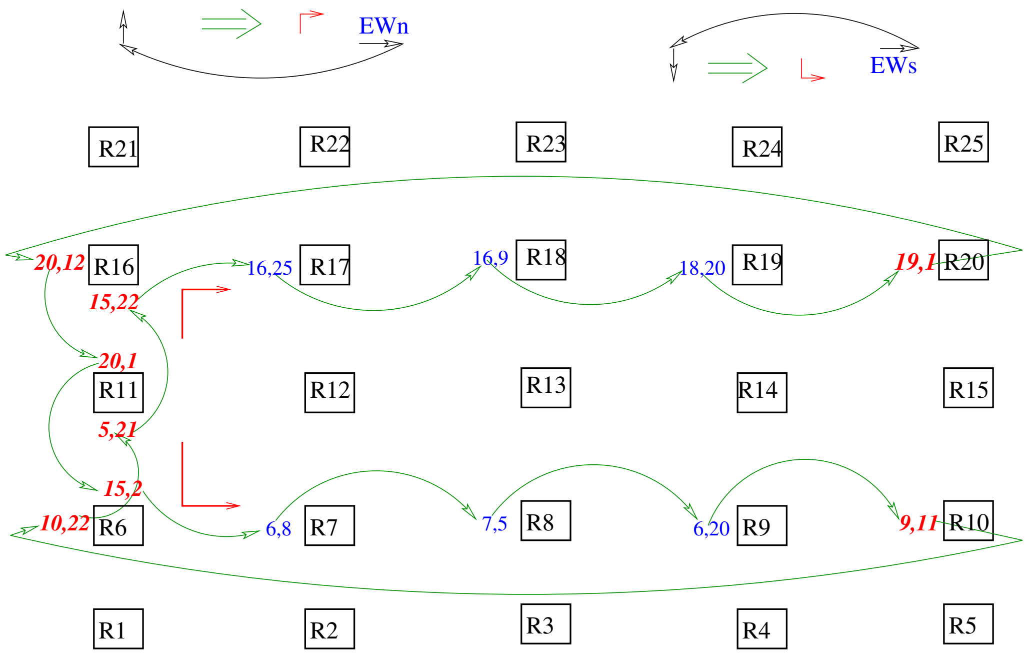


Figure 5.17: *Experimental deadlock scenario in a 5x5 Torus NoC for EWn and EWs Arcs with XY-routing as per Lemma 5.5.1*

5.6.2.1 Deadlock Scenarios for (EWs + EWn) Arc with XY-Turn

The (EWs + EWn) Arc pairs are from the same wraparound channel EW. Both of the EWs and EWn Arcs are part of the same types of wraparound channel EW. As per Lemma 5.5.1, two Arcs from the same wraparound channels create deadlock. Therefore, (EWs + EWn) Arc pairs are deadlock prone with respect to XY-Turns in the Mesh sub-network. The experimental deadlock scenario for (EWs + EWn) with XY-routing is shown in Fig. 5.17. (EWs + EWn) Arc pairs introduce new Turns. The NE Turn is introduced Turns by EWn Arc and SE Turn is introduced by EWs *Arcs*, respectively. These new Turns are shown with red color in Fig 5.17.

The packets shown with red color and *italic font* have used either EWs *Arc* or EWn *Arc*. The packet (19, 1) is currently present at the West input port of R20. It intends to use the same EWs Arc between R20 and R16 which is already used by (20, 12) and (20, 1). The packet (15, 2) has used another EWs Arc between R15 and R11 and has reached the North port buffer of R6 where it was involved in the deadlock cycle and stuck for ever. It intend to follow XY-routing for the rest of the path from R6 by taking a SE Turn. In this way the resource dependency keeps on spreading across packets.

Similarly, the packets that has used or intend to use EWn Arcs are (9, 11), (10, 22), (5, 21) and (15, 22). They are highlighted with red color. Dependencies between all the packets are shown with a curved arrow. It is challenging to construct such an experimental dependency showing resource dependency in detail level by a manual process. Our CFSM framework has automated this process and make it convenience for us. The experimental packets wise detailed dependency in Fig. 5.17 matches with the DDG shown in Fig. 5.12. It implies correctness of the DDG presented in the theory part.

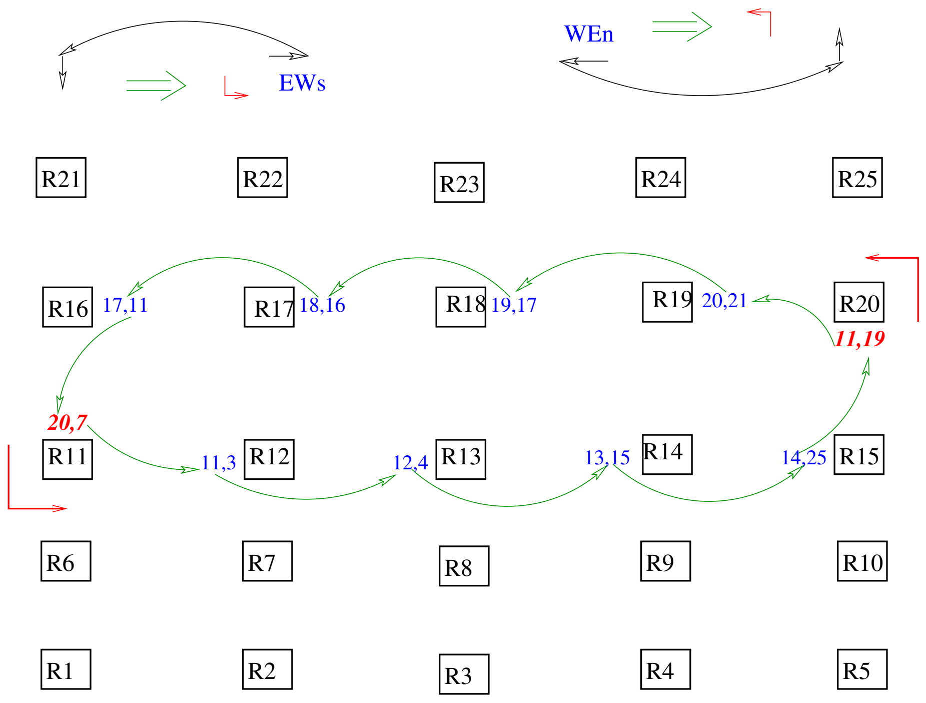


Figure 5.18: *Experimental deadlock scenario in a 5x5 Torus NoC for EWs and WEn Arcs with XY-routing due to new Turns introduced by Arcs*

5.6.2.2 Deadlock Scenarios for (EWs + WEn) Arc with XY-Turn

The Arc pair (EWs + WEn) introduces two YX-Turns. WEs introduces SE Turn and WEn introduces NW Turn. We have already considered all XY-Turns are in the Mesh sub-network. Due to these two newly introduced YX-Turns deadlock cycle is possible as described in the Subsection 5.5.2. The experimental deadlock snapshot for (EWs + WEn) with XY-Turn in the Mesh sub-network is depicted in Fig. 5.18.

In the Fig. 5.18, the packet (11, 19) at the South port buffer of R20 has traversed using the WEn Arcs between R11 and R15. The packet would follow XY-Turns from R20 to reach the destination. Thus it intends to take a NW Turn and stuck in the dependency cycle for ever. Similarly, the packet (20, 7) present at the North port of R11 uses the EWs Arcs between R20 and R16. It intend to take SE Turn and stuck in the cyclic resource dependency. The experimental packets wise dependency cycle in Fig. 5.18 is matching with the dependency graph shown in Fig. 5.14.

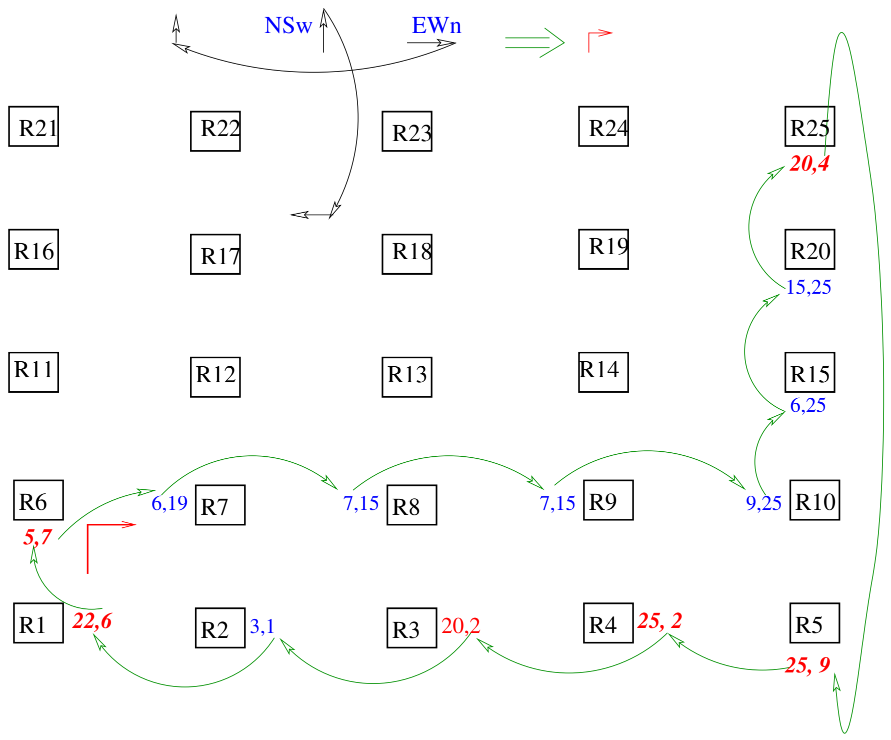


Figure 5.19: *Experimental deadlock scenario in a 5x5 Torus NoC for EWn and NSw Arcs with XY-routing as per Lemma 5.5.2*

5.6.2.3 Deadlock Scenarios for (EWn + NSw) Arc with XY-Turn

In the third example for experimental deadlock scenario, we have considered (EWn + NSw) Arc pairs. As per Lemma 5.5.2, EWn and NSw Arcs are deadlock prone. Deadlock detection time for (EWn + NSw) Arc with respect to XY-Turn is 1556.76 seconds from the Table 5.2. The experimental deadlock scenario for (EWn + NSw) with XY-routing is depicted in Fig. 5.19. This deadlock snapshot complies with the dependency graph shown in Fig. 5.15. The packet (20, 4) intends to use NSw Arc and reached the South port buffer of R25. The packet (25, 9) at the South port of R5, (25, 2) at the East port of R4, (20, 2) at the East port of R3 has already used the NSw Arcs between R25 and R5. The packet (22, 6) at the East port of R1 has used the NSw Arc between R22 and R2. The EWn Arc between R5 and R1 is used by the packet (5, 7). The packet has reached the South port of R6 and intend to use NE Turn to reach its destination. The NE Turn is introduced due to EWn Arc.

5.7 Conclusion

This chapter proposes a deadlock avoidance approach for Torus NoC. An Arc model is proposed for avoiding deadlock in Torus NoC. It imposes routing restrictions to break the cyclic path via wraparound channel. A directional dependency graph (DDG) is proposed which is be convenient for deadlock detection, useful in formulating deadlock avoidance and showing deadlock freedom. As an application of Arc model and DDG, the use of Arcs are discussed with XY-Turns. Arc pairs are categorised into deadlock free and deadlock prone pairs with respect to XY-Turns. Application of deadlock avoidance using DDG cycle of deadlock prone Arc pairs are shown with examples. All of the 28 possible Arc pairs along with XY-Turns in the Mesh sub-network are implemented in our CFSM based simulation framework and deadlock scenarios are experimentally validated by extensive random simulations. The proposed Arc model and DDG would be helpful in formulating deadlock free routing algorithms for Torus by considering maximum possible Arcs and Turns. We discuss various routing algorithms using Arc model in the next chapter.

6
Deadlock Free Routing Algorithms for Torus NoC using Arc Model

6.1 Introduction

The wraparound channels help in reducing overall hop counts in a given traffic in Torus NoC. On the other hand, these inherent circular paths act as a catalyst in the formation of circular dependency where a sequence of packets are involved. Thus, deadlock arises and the system stops functioning eventually. For avoiding deadlock in Torus NoC different techniques like bubble flow control [28, 29] or virtual channels (VC) [3] are used. In the bubble flow control approach, extra buffers are needed to ensure empty buffer slots at any circumstances [28,29]. In the VC approach, different classes of VCs are used to break cyclic dependencies [2,11,27]. However, these methods use additional resources like buffer or VCs to prevent from occurring deadlock.

Only a few algorithms exist for Torus NoC that avoids deadlock without using any additional resources. Up*/Down* routing is a classical deadlock avoidance approach applicable for any communication network including Torus NoC [10]. A given topology is converted into a spanning tree in Up*/Down* routing. A spanning tree is acyclic in nature. These acyclic paths corresponding to a given NoC topology are used for routing packets. Due to the acyclic routing paths via a spanning tree deadlock is not possible in Up*/Down* routing. Though deadlock is avoided, most of the packets use longer paths in Up*/Down* routing approach [94]. Turn model is another approach where deadlock is avoided by controlling the movements of packets by restricting certain Turns [1, 9]. The primary significance of Turn model is that it avoids deadlock without using additional buffers and virtual channels. To deal with the inherent cyclic paths in Torus NoC via wraparound channel, FirstHop approach is used along with Turn model approach. In FirstHop approach, wraparound channels in Torus NoC are permitted to use by the packets that are injected from the boundary routers. This restriction helps in discontinuing the resource dependency via wraparound channels in Torus. Thus, deadlock is avoided in FirstHop approach.

In this chapter, we have shown that our Arc model is specifically applicable for avoiding deadlock in Torus NoC without using any additional resources. In Arc model approach, inevitable cyclic dependency in circular paths via wraparound channels are broken by restricting immediate backward movement just after taking the wraparound channel. Each wraparound channels are logically subdivided into two Arcs based on next move just after traversing through wraparound channel. Thus, there are eight logical Arcs from four wraparound channels. Arcs are needed to be chosen carefully considering a known deadlock free routing algorithm in the Mesh sub-network such that the combined effects of Turns in the routing algorithm and the Arcs do not create deadlock. All Arcs cannot be used at a time as some of the Arcs combinations are deadlock prone. One advantage of Arc over First

Hop method is that Arcs are applicable not only for packets injected from boundary nodes, but also for packets injected form non boundary nodes as well. *Therefore, combining the suitable subset of Arcs and a routing algorithm in Mesh sub-network is a promising approach to develop deadlock free routing algorithms for Torus NoC.*

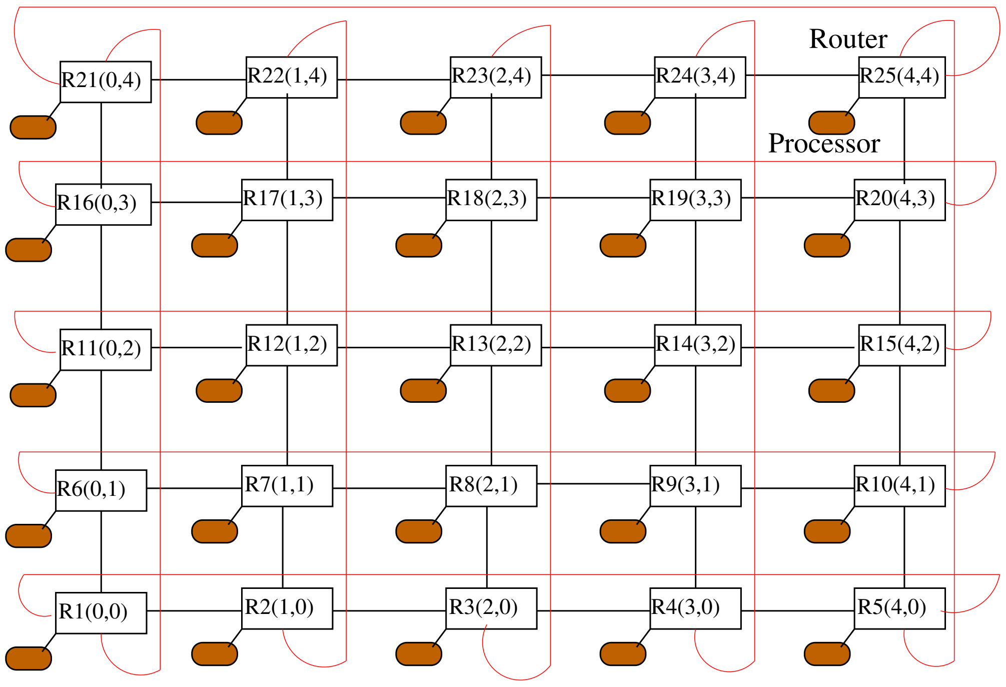


Figure 6.1: 5x5 *Torus NoC*

6.1.1 Contributions

In this work, we have demonstrated various Arc based routing algorithms in Torus and have evaluated their effectiveness in saving hop counts in comparison with FirstHop and Up*/Down* algorithms. We have also shown a way to combine both Arc model and FirstHop approach without creating deadlock. We have demonstrated the first routing algorithm using two Arcs, the second routing algorithm using three Arcs and the third routing algorithm by combining three Arcs along with First Hop approach. The contributions presented in this chapter are summarised as follows:

- A deadlock free deterministic routing algorithm for Torus using two Arcs with XY-Turns in the Mesh sub-network is presented.

- A deterministic deadlock free routing algorithm for Torus using three Arcs with XY-routing is presented.
- A deterministic deadlock free routing combining three Arcs along with FirstHop approach is presented.
- Deadlock freedom are shown using DDG for the proposed algorithms. Experimental deadlock detection is also performed to check if there is any contradiction with DDG analysis for deadlock freedom.
- Effectiveness of our proposed algorithms in terms of saving in hop counts are evaluated using different traffic patterns. The proposed Arc based algorithms are compared with FirstHop and Up*/Down* routing approaches. Experimental results show that the Arc based routing algorithms are effective in terms of hop counts in a given traffic pattern in comparison to the FirstHop and the Up*/Down* routing.

The rest of the chapter is organized as follows. Design approach for deadlock free routing algorithm for Torus NoC using Arc model is presented in Section 6.2. A routing algorithm using two different Arcs combination and its deadlock freedom are presented in Section 6.3. Number of maximum Arcs that can be used in a routing algorithm is presented in Section 6.4. Routing using three Arcs, routing using combination of Arc and FirstHop approach and deadlock freedom for the respective algorithms are presented in Section 6.5 and Section 6.6, respectively. Experimental results are presented in Section 6.7. Finally, the chapter is concluded in Section 6.8.

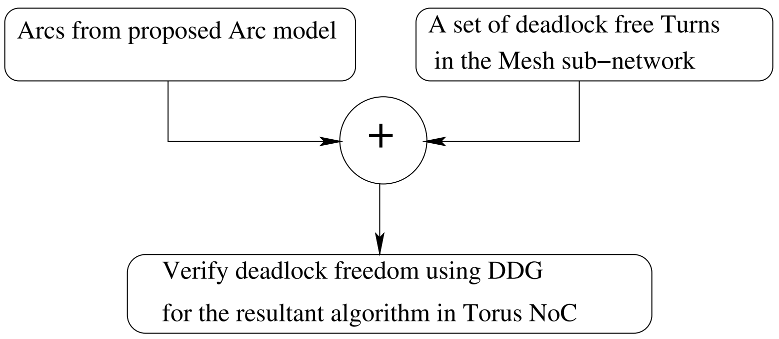

Arcs from proposed Arc model
A set of deadlock free Turns in the Mesh sub−network
+
Verify deadlock freedom using DDG for the resultant algorithm in Torus NoC

Figure 6.2: *Designing deadlock free routing for Torus NoC using Arcs*

6.2 Deadlock Free Routing Algorithm Design Approach using Arc Model

In a routing algorithm for Torus NoC using Arc model, different deadlock free Arc combinations are possible to be used. A Torus NoC is visualised as a combination of wraparound channels and Mesh sub-network. Therefore, a set of mutually deadlock free Turns in the Mesh sub-network are considered along with suitable Arc combinations for designing a deadlock free routing algorithm for Torus NoC in the thesis. Our approach of designing a deadlock free routing algorithm for Torus NoC is shown in Fig. 6.2. In our approach, we consider Arcs from the Arc model along with a set of permitted Turns in the Mesh sub-network in such a way that their resultant effects do not create cyclic dependency in the Torus NoC. Since some *Arcs* introduces new Turns, those new Turns along with the permitted Turns in the Mesh sub-network might also create cycle in the resultant routing algorithm for Torus NoC. Considering all such scenarios, we propose the following theorem to verify deadlock freedom for an Arc based routing algorithm in Torus NoC .

Theorem 6.2.1. *A routing algorithm for Torus NoC that uses a set of Arcs for wraparound channel along with a set of deadlock free Turns for the Mesh sub-network is deadlock free iff following two conditions are satisfied:*

- *The new Turn(s) introduced due to use of Arc(s) should not complete a clockwise or anti-clockwise cycle along with the other permitted Turns in the Mesh sub-network.*
- *The Arc(s) itself/themselves should not complete a clockwise or anti-clockwise cycle along with other permitted Turns in a Directional Dependency Graph.*

Proof. As per *Turn model*, deadlock is possible in an algorithm if the permitted Turns form a cycle. The first condition states that the new Turns introduced by Arcs should not complete a cycle with the permitted Turns in the Mesh sub-network. Therefore, the first condition of this Theorem is supported by Turn model theory [1]. The second condition is based on the properties of Directional Dependency Graph (DDG). If cycle is possible in a DDG, deadlock is present in the corresponding routing algorithm. If DDG cycle is not possible it indicates deadlock freedom for that algorithm. To form a DDG cycle, from any starting vertex s in that DDG, there is possibility to spread the dependency in such a way that the starting vertex s is reachable. If formation of such cycle is not possible in any possible DDG, the routing method is deadlock free. Thus, the second condition of the theorem is supported by the properties of DDG. □

Using this design approach, we are going to propose deadlock free routing algorithms for Torus NoC. Routing algorithms with different combination of Arcs and with various combination of deadlock free Turns in the Mesh sub-network are demonstrated in the subsequent section. We first present deadlock freedom using DDG for the considered Arc and Turn combinations. Formal algorithm is presented after that. We consider only XY-Turns for the Mesh sub-network in this thesis. Same procedure is applicable with other combination of Turns as well.

6.3 Routing using Two Arcs along with XY-Turns

There are fourteen pair wise deadlock free *Arc* pairs as shown in Table. 5.1. By using any of these fourteen pairs with XY-routing, fourteen deadlock free routing algorithms for Torus NoC are possible. In this section, we consider one routing algorithm using (EWs + NSe) Arcs with XY-Turns. We have presented the algorithmic steps in Algorithm 1 and shown the deadlock freedom for the algorithm using DDG.

6.3.1 Routing steps for Algorithm 1

A deadlock free routing algorithm using (EWs + NSe) *Arcs* with XY-routing is presented in Algorithm 1. EWs Arc is applicable if the packet is destined West-South direction and the EW wraparound channel could help in reducing hop counts. The condition for applicability of EWs Arc is given in line# 4 in Algorithm 1. If EWs *Arc* is applicable (line# 4 in Algorithm 1), the packet moves towards East boundary until the East boundary of the

Algorithm 1 WS-SE Algorithm

```
1: function WS-SE($S, D$)
2:   ▷ $S = (x_s, y_s)$ and $D = (x_d, y_d)$ are the source(S) and destination(D) co-ordinates of
     a packet in an NxN Torus NoC. Between a source and destination pair, the X-distance
     is $\Delta_x = |x_d - x_s|$ and the Y-distance is $\Delta_y = |y_d - y_s|$. After each move, S is updated.
     The packet reaches destination when $S = D$.
3:     while ( $(x_s \neq x_d) \vee (y_s \neq y_d)$ ) do
4:         if ( $(y_s > y_d) \wedge (x_s > x_d) \wedge (\Delta_x > N/2)$ ) then
5:             EWs Arc is applicable. Keeps on moving in East direction.
6:             Once East boundary is reached, move using EW wraparound channel.
7:             Move one step South for the EWs Arc after taking EW wraparound channel.
8:             Follow XY-routing.
9:         else if ( $(x_s < x_d) \wedge (y_s > y_d) \wedge (\Delta_y > N/2)$ ) then
10:            NSe Arc is applicable. Keeps on moving in North direction.
11:            Once North boundary is reached, move using NS channel.
12:            Move one step East for NSe Arc after taking the NS wraparound channel.
13:            Follow XY-routing.
14:        else
15:            Follow XY-routing.
16:        end if
17:    end while
18: end function
```

Mesh sub-network is reached. At the East boundary, it takes EW wraparound channel and reaches the West boundary. In the next step, the packet moves one hop distance in the South direction as per the rules of EWs Arc. After that, XY-routing is followed until it reaches the destination.

Similarly, NSe Arc is applicable if the packet is destined towards the South-East direction and the NS wraparound channel helps in reducing the hop counts. A packet traverses using NSe Arc if the condition (line#9 in Algorithm 1) is satisfied. Conditions for taking *Arcs* ensure that EWs would not be taken at bottom row and NSe would not be taken at right most column. Therefore, none of the packet will use both the Arcs. If neither EWs Arc nor NSe Arc conditions are applicable, control goes to line#15 and XY-routing is followed until the packet reaches its destination. If neither EWs nor NSe Arc is applicable, XY-routing is followed. Even after traversing through EWs or NSe Arc, a packet follows XY-Turns for the rest of the paths. Since the Arcs are applicable only to the packets that are destined towards West-South or South-East direction, we term Algorithm 1 as *WS-SE algorithm.*

The conditions for EWs and NSe Arcs can be checked at any order in the algorithm and it would not effect the routing of a packet by the algorithm.

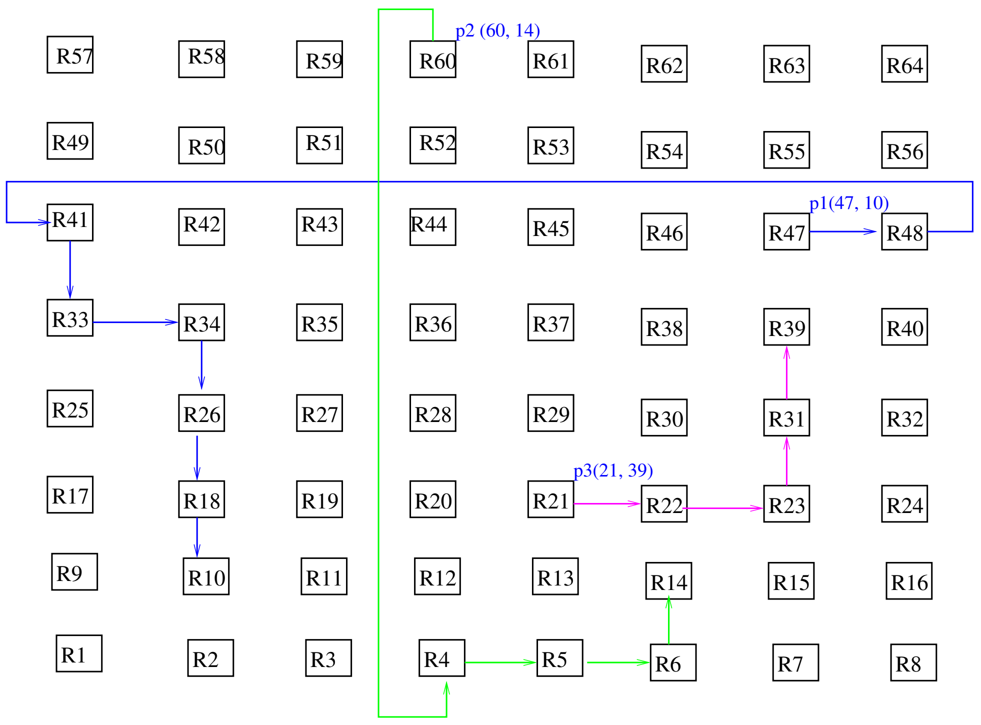


Figure 6.3: *Paths for packet p1(47, 10), p2(60, 14) and p3(21, 39) as per Algorithm 1*

Example 6.1. The routing path for packet p1(47, 10), p2(60, 14) and p3(21, 39) in a 8x8 Torus NoC with respect to Algorithm 1 are shown in Fig. 6.3. The packet p1(47, 10) uses EWs Arc and the hop count saved is 9 - 7 = 2. Here, 9 is the hop count for XY-routing and 7 is the hop counts due to use of EWs Arc. The packet p2(60, 14) uses NSe Arc and the hop count saved is 8 - 4 = 4. For the packet p3(21, 39) no Arc is applicable. It follows XY-routing and hop count traversed is 4. There is no saving in hop counts.

In a similar way, fourteen such routing algorithms are possible from the deadlock free *Arc* pairs in Table. 5.1 with XY-Turns in the Mesh sub-network. Hop count savings by Arc based algorithm depends on the traffic distribution as well. For example, Algorithm 1 would be more effective if more traffics are destined towards West-South or South-East direction. For uniform traffic, since traffic distribution is uniform, all of the fourteen algorithms are expected to perform similar way in their effectiveness in saving hop counts. One packet using only one Arc in this algorithm. The conditions for Arcs can be checked in any order in Algorithm 1. It would not effect the routing of a packet.

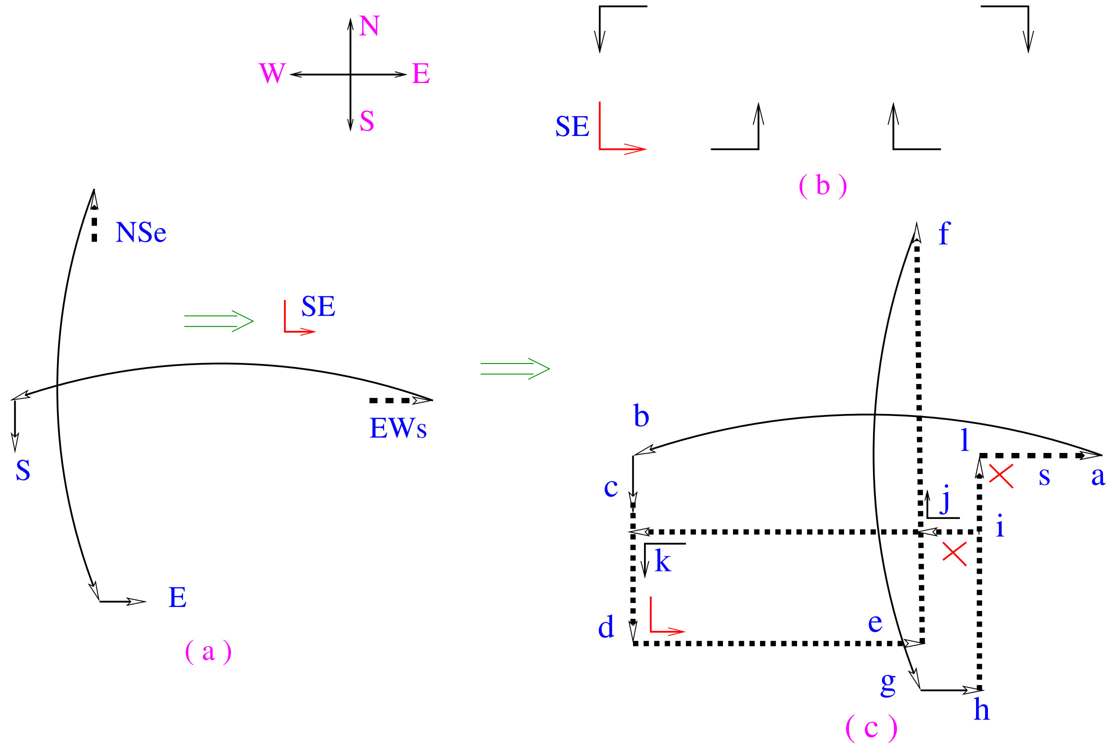


Figure 6.4: *(a) EWs and NSe Arcs (b) Dependency graph: deadlock is not possible by EWs and NSe Arcs*

6.3.2 Deadlock freedom for Algorithm 1

The EWs Arc introduces SE Turn when a packet has to move back to reach destination after taking the Arc and shown in Fig. 6.4(a). Therefore, five Turns namely EN, ES, WN, WS and SE are considered for the movement in the Mesh sub-network. Cycle creation is not possible with these five Turns which is shown in Fig. 6.4(b). Therefore, the first condition of Theorem 6.2.1 for deadlock freedom is satisfied.

For checking the second condition, let us consider a directional dependency graph using EWs and NSe *Arcs* in Fig. 6.4(c). Since each individual *Arcs* are deadlock free with respect to XY-routing, let us check if deadlock is possible by resultant effects of both EWs and NSe *Arcs*. Let a sequence of packets produce a directional dependency via vertices $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow l$. Though this graph involved both EWs and

NSe *Arcs*, cycle creation is still not possible. A NW turn is needed at a point i for creating two possible cycle via vertices j or k. For another possible cycle, a NE turn is needed at a point l. Since both NW and NE turns are not permitted here, cycle creation is not possible. Therefore, Algorithm 1 using EWs and NSe *Arcs* with XY-routing in the Mesh sub-network is deadlock free.

6.4 Maximum Possible Arcs with XY-Turns in a Routing Algorithm

In this section, we evaluate maximum number of Arcs that can be used with XY-Turns in a deadlock free routing algorithm for Torus NoC. We analyse all possible Arc combinations for such algorithms in this section and demonstrate such an algorithm in the next section.

6.4.1 Evaluation plan

One approach of determining the maximum possible deadlock free Arc combination is to start with one deadlock free Arc pair from Table 5.1 and keep on adding Arc from the remaining six Arcs one by one into the deadlock free Arcs pairs such that deadlock freedom is maintained. After examining all the six Arcs, we found a combination of possible deadlock free Arcs.

We categorise the Arcs as X-Arcs and Y-Arcs. X-Arcs are from the two wraparound channels EW and WE in the X-direction, i.e., EWn, EWs, WEn and WEs Arcs. Similarly, Y-Arcs are from the two wraparound channels NS and SN in the Y-direction, i.e., NSe, NSw, SNe and SNw Arcs. First we evaluate the maximum deadlock free Arc combination using X-Arcs. Y-Arcs are added to the group one by one if no deadlock is created. We then consider the Y-Arcs and add X-Arcs one by one to analyse the deadlock.

6.4.2 Considering deadlock free Arc pairs in X-direction

At first, let us consider deadlock free *Arc* pairs (EWn + WEn). From the remaining six *Arcs* namely EWs, WEs, NSe, NSw, SNe and SNw, we check by adding one by one if the resultant combination is free from deadlock.

- EWs cannot be added to (EWn + WEn) as (EWs + EWn) are from deadlock prone due to new Turn introduced by Arcs, as explained with example in Subsection 5.5.2.
- WEs cannot be added as (WEs + WEn) are deadlock prone according to Lemma 5.5.1 in Chapter 5.
- NSe cannot be added as per Lemma 5.5.2 in Chapter 5. Both (EWn + NSe) and (WEn + NSe) are deadlock prone.
- NSw cannot be added as per Lemma 5.5.2. Both (EWn + NSw) and (WEn + NSw) are deadlock prone.
- SNe does not create deadlock with (EWn + WEn) as per Table 5.1. Therefore, (EWn + WEn + SNe) is deadlock free with respect to XY-routing in the Mesh sub-network.
- SNw does not create deadlock with (EWn + WEn) as per Table 5.1. Therefore, (EWn + WEn + SNw) is deadlock free.

Either SNe or SNw are safe to add with (EWn + WEn). If we consider both SNe and SNw Arc, they are (SNe + SNw) mutually deadlock prone as per Lemma 5.5.1. Therefore, (EWn + WEn + SNe + SNw) is also deadlock prone. Therefore, a deadlock free combination of four Arcs are not possible while start with (EWn + WEn). Thus, we get two deadlock free combination with three Arcs, (EWn + WEn + SNe) and (EWn + WEn + SNw). In similar way, by considering the other deadlock free Arc pairs (EWs + WEs) in X-direction, we get two other deadlock free Arc combinations (EWs + WEs + NSe) and (EWs + WEs + NSw). Here, (NSe + NSw) are deadlock prone as per Lemma 5.5.1. Therefore, a deadlock free combination of four Arcs are not possible with (EWs + WEs + NSe + NSw).

6.4.3 Considering deadlock free Arc pairs in Y-direction

There are four deadlock free Arc pairs in Y-direction from the Table 5.1. These pairs are, (SNw + NSw), (NSe + SNe), (SNw + NSe) and (NSw + SNe). Let us consider deadlock free Arcs pair (NSe + SNe). In this pair no other *Arc* in Y-direction can be added due to Lemma 5.5.1 in Chapter 5. Similarly, no Arc in X-direction can be added due to Lemma 5.5.2 in Chapter 5. For other three deadlock free Arcs pair in Y-direction as well, it is not possible to add any new *Arcs*. Therefore, four deadlock free Arc combinations only with three Arcs are possible with respect to XY-routing in a routing algorithm for Torus without using additional VCs or buffer. These four deadlock free Arc combinations are (EWs + WEs + NSe), (EWs + WEs + NSw), (EWn + WEn + SNe) and (EWn + WEn + SNw).

6.5 Routing using Three Arcs with XY-Turns

Four routing algorithm using three Arcs are possible as concluded from the previous section. We consider one such routing algorithm that uses (EWs + WEs + NSe) with XY-Turns in this section. We have presented the algorithmic steps in Algorithm 2 and shown the deadlock freedom for the algorithm using DDG.

6.5.1 Routing steps for Algorithm 2

Algorithmic steps for the routing algorithm using (EWs + WEs + NSe) *Arcs* with XY-routing is presented in Algorithm 2. If EWs *Arc* is applicable (line# 4 in Algorithm 2), the packet keeps on moving towards East boundary. At the East boundary, it takes EW wraparound channel and reaches the West boundary. In the next step, the packet moves one hop distance in the South direction for the EWs *Arc*. After that, XY-routing is followed until it reaches the destination.

If WEs *Arc* is applicable (line# 9 in Algorithm 2), the packet keeps on moving towards West boundary. At the West boundary, it takes WE wraparound channel and reaches the East boundary. In the next step, the packet moves one hop distance in the South direction

Algorithm 2 3Arcs Algorithm

```
1: function 3ARCS($S, D$)
2:  ▷ $S = (x_s, y_s)$ and $D = (x_d, y_d)$ are the source(S) and destination(D) co-ordinates of a packet in an NxN Torus NoC. Between a source and destination pair, the X-distance is $\Delta_x = |x_d - x_s|$ and the Y-distance is $\Delta_y = |y_d - y_s|$. After each move, S is updated. The packet reaches destination when $S = D$.
3:     while ( $(x_s \neq x_d)$ $\vee$ $(y_s \neq y_d)$ ) do
4:         if ( $(y_s > y_d)$ $\wedge$ $(x_s > x_d)$ $\wedge$ $(\Delta_x > N/2)$ ) then
5:             EWs Arc is applicable. Keeps on moving in East direction.
6:             Once East boundary is reached, move using EW channel.
7:             Move one step South for the EWs Arc.
8:             Follow XY-routing.
9:         else if ( $(y_s > y_d)$ $\wedge$ $(x_s < x_d)$ $\wedge$ $(\Delta_x > N/2)$ ) then
10:            WEs Arc is applicable. Keeps on moving in West direction.
11:            Once West boundary is reached, move using WE channel.
12:            Move one step South for WEs Arc.
13:            Follow XY-routing.
14:        else if ( $(x_s < x_d)$ $\wedge$ $(y_s > y_d)$ $\wedge$ $(\Delta_y > N/2)$ ) then
15:            NSe Arc is applicable. Keeps on moving in North direction.
16:            Once North boundary is reached, move using NS channel.
17:            Move one step East for NSe Arc after taking the NS wraparound channel.
18:            Follow XY-routing.
19:        else
20:            Follow XY-routing.
21:        end if
22:    end while
23: end function
```

for the WEs *Arc*. After that, XY-routing is followed until it reaches the destination.

Similarly, a packet traverses using NSe *Arc* if condition of line#14 in Algorithm 2 is satisfied. Conditions for taking *Arcs* ensure that EWs and WEs would not be taken at bottom row and NSe would not be taken at right most column. Therefore, one *Arc* does not end up with another consecutive *Arc*. If none of the (EWs + WEs + NSe) *Arc* conditions are satisfied, control goes to line#20 and XY-routing is followed until the packet reaches its destination. The conditions for EWs, WEs and NSe Arcs can be checked at any order and it would not effect the algorithm. If none of the Arcs is applicable, XY-routing is applicable by default. Packets that are destined towards South-East and South-West directions will get benefited form Arc on reducing hop counts to reach their destinations. A single packet is allowed to utilize only one Arc in this algorithm. The conditions for Arcs can be checked in any order. It would not effect the routing of a packet.

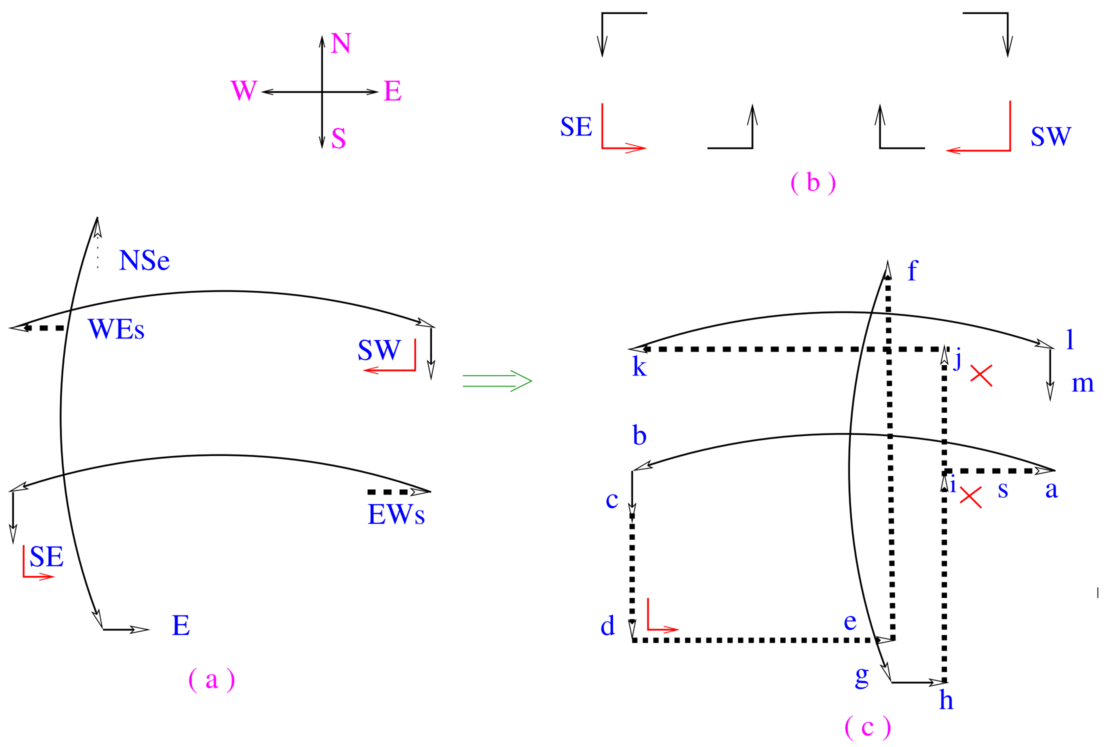


Figure 6.5: *Deadlock freedom: (a) EWs, WEs and NSe Arcs (b) DDG representing deadlock freedom for EWs, WEs and NSe Arcs with XY-Turns*

6.5.2 Deadlock Freedom for Algorithm 2

The EWs, WEs and NSe Arcs are shown in Fig. 6.5(a). The WEs Arc introduces SW Turn and the EWs Arc introduces SE Turn. The resultant Turn model due to these additional Turns is shown in Fig. 6.5(b). Since no cycle formation is possible as per Turn model, the first condition of Theorem 6.2.1 for deadlock freedom is satisfied.

For checking the second condition of Theorem 6.2.1, let us consider a directional dependency graph using EWs, WEs and NSe Arcs in Fig. 6.5. Let us check if deadlock is possible by resultant effects of EWs, WEs and NSe Arcs. Let a sequence of packets produce a directional dependency via vertices $s \to a \to b \to c \to d \to e \to f \to g \to h \to i \to j$. For creating a cycle involving EWs Arc, a NE Turn is needed at some vertex i. Similarly, a NW Turn is needed at a point j for spreading the dependency into WEs Arc. Since both NE and NW turns are not permitted in this combination, cycle creation is not possible using

Arcs and Turns.

Both the conditions of Theorem 6.2.1 are satisfied for Algorithm 2. Therefore, Algorithm 2 using EWs, WEs and NSe Arcs Arcs with XY-routing in the Mesh sub-network is deadlock free.

6.6 Combination of Arc Model and FirstHop Method

In this section, we explore whether it is feasible to use Arcs and First Hop approach together to design a deadlock free routing algorithm for Torus NoC. Wraparound channel are safe to use at its first hop, if the packet is originated from a boundary router and use of the wraparound channel could reduce the hop counts [1]. In this section, we plan to develop routing algorithm by using unused wraparound channel in the first hop along with with a set of deadlock free Arcs to make the algorithm more effective in saving hop counts. Deadlock free Arcs from the Arc model would use leverage of wraparound channel in a better way. Using the unused wraparound channel in the same Arc based algorithm in the first hop for the boundary packets would help in taking advantages from both Arc model and FirstHop approaches. We combine Algorithm 2 with FirstHop approach and develop Algorithm 3. The deadlock freedom for the modified algorithm has to check while adding wraparound channels into the algorithm with the FirstHop restriction.

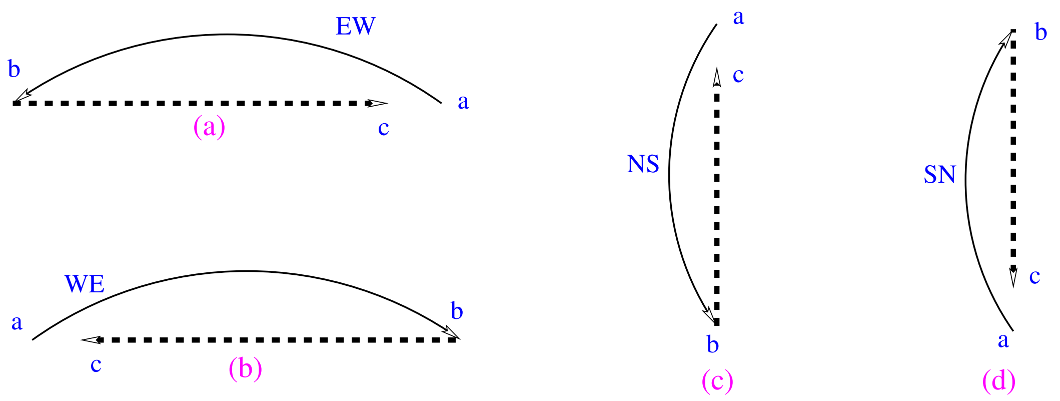


Figure 6.6: *Directional Dependency graphs for (a) EW, (b) WE, (c) NS and (d) SN wraparound channel with FirstHop restriction*

6.6.1 DDG for Wraparound Channel with FirstHop Restriction

Before combining the FirstHop approach with Arc based algorithm, we elaborate why FirstHop approach is deadlock free with the help of DDG. The dependency graph for EW, WE, NS and SN wraparound channels with FirstHop approach is shown in Fig. 6.6(a), Fig. 6.6(b), Fig. 6.6(c) and Fig. 6.6(d), respectively. For creating a deadlock configuration, the source and destination of each packet should be at least two hops away [2]. To stop such spreading of resource dependency between two vertices c and a, in Fig. 6.6(a), Fig. 6.6(b), Fig. 6.6(c) and Fig. 6.6(d), FirstHop restriction is proposed while using a wraparound channel [1]. The path from vertex c to a exist, whereas restriction is imposed on reaching vertex a. In Fig. 6.6(a), if a packet that has reached vertex c, it can also move to the vertex a. Whereas, the same packet cannot use EW wraparound channel. Specifically, such a packet cannot travel both the path $c \rightarrow a$ and $a \rightarrow b$, i.e., the path $c \rightarrow a \rightarrow b$ is restricted for a single packet. In other words, if a packet has to move from vertex c to vertex b, the path $c \rightarrow a \rightarrow b$ is restricted and it has to follow $c \rightarrow b$ via many intermediate routers even if the hop count increases. Only packet that is injected from the router corresponding to vertex a are eligible to take the EW wraparound channel. The dependency is thus discontinued between the vertex c and a by applying first hop restrictions in Fig. 6.6(a), Fig. 6.6(b), Fig. 6.6(c) and Fig. 6.6(d), corresponding to the EW, WE, NS and SN wraparound channels, respectively.

6.6.2 Routing using Algorithm 2 and Wraparound Channel with FirstHop Restriction

We consider Algorithm 2 as the Arc based algorithm and use wraparound channels in the first hop such that the resultant algorithm does not create deadlock. The steps for the resultant algorithm is presented in Algorithm 3 and the deadlock freedom is also shown next.

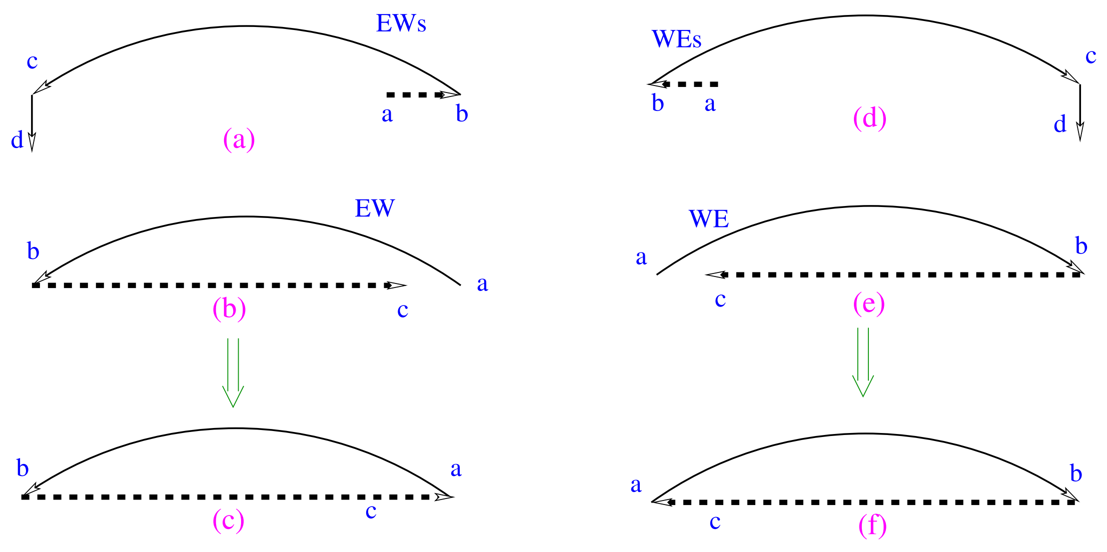


Figure 6.7: *Directional Dependency graphs for (a) EWs, (b) EW, (c) EWs and EW, (d) WEs, (e) WE, (f) WEs and WE*

6.6.2.1 Wraparound Channel Compatible with Algorithm 2

The three Arcs present in Algorithm 2 are EWs, WEs and NSe Arcs with XY-Turns in the Mesh sub-network. Let us check if any wraparound channels form deadlock with Algorithm 2 by considering EW, WE, NS and SN wraparound channel one by one.

Let us consider the EW wraparound channel with FirstHop restriction. The dependency graph for EWs Arcs and the EW wraparound channel with first hop restriction is shown in Fig. 6.7(a) and Fig. 6.7(b). The movements through both EWs *Arcs* and EW wraparound channel with first hop restriction are applicable to a same path or a same physical channel. Since there might present sequence of packets following both EWs *Arcs* and EW wraparound channel in FirstHop in the same path, the FirstHop restriction is automatically violated due to use of EWs Arc. The possible path of their resultant effect is shown in Fig. 6.7(c) using DDG. Thus a deadlock cycle is created while combining EWs *Arcs* and EW wraparound channel even with FirstHop restriction. Therefore, it is not feasible to use EW wraparound channel with Algorithm 2.

Similarly, using WEs *Arc* and WE wraparound channel with FirstHop creates deadlock as shown in Fig. 6.7(d), Fig. 6.7(e) and Fig. 6.7(f). Therefore, WE wraparound channel cannot be added. Similarly, using NS wraparound channel with FirstHop also creates deadlock with

NSe as shown in Fig. 6.8(a-c).

Let us check for deadlock possibilities while using SN wraparound channel. Since SNe or SNw *Arcs* are not present in the considered *Arcs* combination (EWs + WEs + NSe), SN wraparound channel cannot create deadlock with SNe or SNw *Arcs*. Therefore, (EWs + WEs + NSe + SN) is a compatible combination where SN wraparound channel has to use only in first hop of a packet. We consider (EWs + WEs + NSe + SN) with XY-Turns as Algorithm 3 and its deadlock freedom is analysed using DDG.

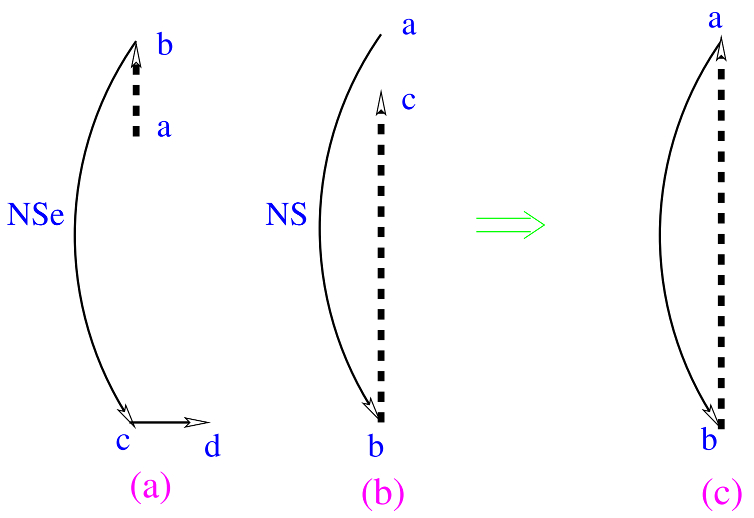


Figure 6.8: *Directional Dependency Graph for (a) NSe, (b) NS, (c) NSe and NS.*

6.6.2.2 Routing steps for Algorithm 3

The routing steps using (EWs + WEs + NSe + SN) are presented in Algorithm 3. Routing steps are similar as that of Algorithm 2. For using SN wraparound channel it is required to check if the packet is injected from a South boundary router and whether hop count would be saved due to SN wraparound channel. There is no mandatory movement after using SN. Just follow XY-routing after SN.

If EWs *Arc* is applicable (line# 4 in Algorithm 2), the packet keeps on moving towards East boundary. At the East boundary, it takes EW wraparound channel and reaches the West boundary. In the next step, the packet moves one hop distance in the South direction for the EWs *Arc*. After that, XY-routing is followed until it reaches the destination.

Algorithm 3 Arc and First Hop Algorithm

```
1: function ArcAndFirstHop($S, D$)
2:  ▷ $S = (x_s, y_s)$ and $D = (x_d, y_d)$ are the source(S) and destination(D) co-ordinates of a packet in an NxN Torus NoC. Between a source and destination pair, the X-distance is $\Delta_x = |x_d - x_s|$ and the Y-distance is $\Delta_y = |y_d - y_s|$. After each move, S is updated. The packet reaches destination when $S = D$.
3:     while ( $(x_s \neq x_d) \lor (y_s \neq y_d)$ ) do
4:         if ( $(y_s > y_d) \land (x_s > x_d) \land (\Delta_x > N/2)$ ) then
5:             EWs Arc is applicable. Keeps on moving in East direction.
6:             Once East boundary is reached, move using EW channel.
7:             Move one step South for the EWs Arc.
8:             Follow XY-routing.
9:         else if ( $(y_s > y_d) \land (x_s < x_d) \land (\Delta_x > N/2)$ ) then
10:            WEs Arc is applicable. Keeps on moving in West direction.
11:            Once West boundary is reached, move using WE channel.
12:            Move one step South for WEs Arc.
13:            Follow XY-routing.
14:        else if ( $(x_s < x_d) \land (y_s > y_d) \land (\Delta_y > N/2)$ ) then
15:            NSe Arc is applicable. Keeps on moving in North direction.
16:            Once North boundary is reached, move using NS channel.
17:            Move one step East for NSe Arc after taking the NS wraparound channel.
18:            Follow XY-routing.
19:        else if ( $(y_s = 0) \land (\Delta_y > N/2)$
20:                $\land$ ( The packet injected from current router) ) then
21:            SN wraparound channel is applicable.
22:            The packet is in a south boundary router and injected from that router.
23:            Move South(S) to take SN wraparound channel.
24:            Follow XY-routing.
25:        else
26:            Follow XY-routing.
27:        end if
28:    end while
29: end function
```

If WEs *Arc* is applicable (line# 9 in Algorithm 3), the packet keeps on moving towards West boundary. At the West boundary, it takes WE wraparound channel and reaches the East boundary. In the next step, the packet moves one hop distance in the South direction for the WEs *Arc*. After that, XY-routing is followed until it reaches the destination.

Similarly, a packet traverses using NSe *Arc* if condition of line#14 in Algorithm 3 is satisfied. Conditions for taking *Arcs* ensure that EWs and WEs would not be taken at bottom row and NSe would not be taken at right most column. Therefore, one Arc does not end up with another consecutive *Arc*.

The condition for SN wraparound channel with first hop is shown in line# 19 in Algorithm 3. The condition for SN wraparound channel is, the packet should injected from a boundary router and using the wraparound channel helps in reducing hop counts. The conditions for EWs, WEs, NSe Arcs and SN wraparound channel in first hop can be checked at any order and it would not effect the algorithm. If none of the conditions for (EWs + WEs + NSe + SN) are satisfied, control goes to line#26 and XY-routing is followed until the packet reaches its destination. The conditions for Arcs and wraparound channels can be checked in any order. It would not effect the routing of a packet even if the first hop condition is checked first. The packets that are destined towards South-East direction and South-West direction and the packets that are injected from the South boundary row and destined towards North would get benefit from Algorithm 2 in saving hop count.

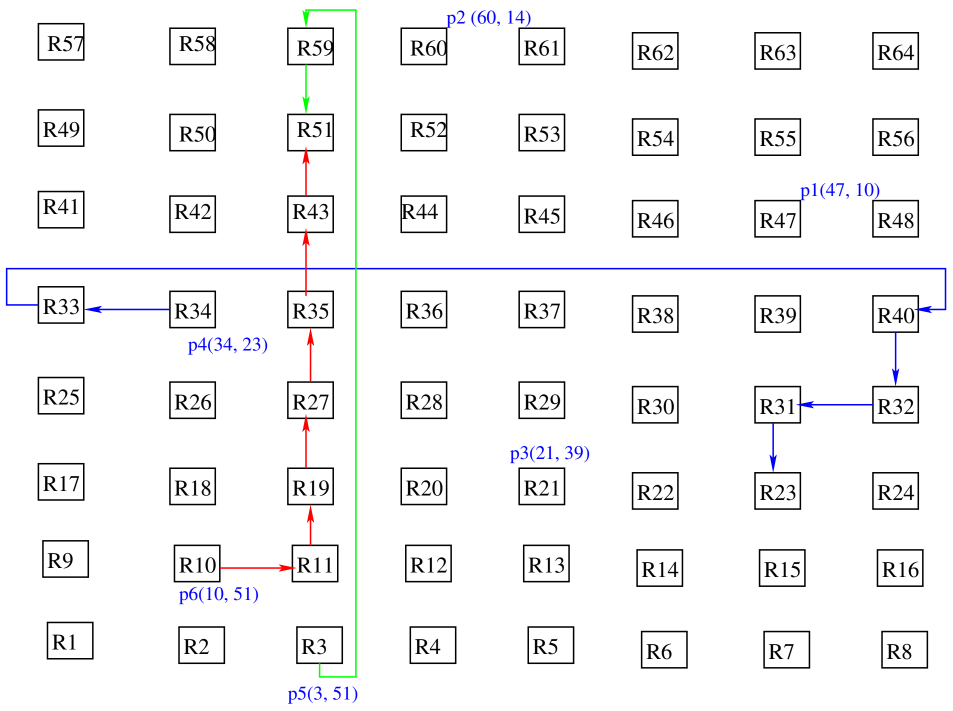


Figure 6.9: *Paths for packet p4(34, 23), p5(3, 51) and p6(10, 51) as per Algorithm 3*

Example 6.2. As per Algorithm 3, packet p1(47, 10) and p2(60, 14) use EWs Arc and NSe Arc, respectively. No Arc is used by the packet p3. The routing paths for packet p1(47, 10), p2(60, 14) and p3(21, 39) are same as shown in Fig. 6.3. They are not shown here again.

The routing path for packet p4(34, 23), p5(3, 51) and p6(10, 51) in a 8x8 Torus NoC are shown in Fig. 6.9. The packet p4(34, 23) uses WEs Arcs and the hop count saved is 7 - 5 = 2. Here, 7 is the hop count for for XY-distance and 5 is the hop counts due to use of WEs Arc. For the packet p5(3, 51), SN wraparound channel with FirstHop is applicable. Hop count saved is 6 - 2 = 4. Considering the packet p6(10, 51), SNe Arc is applicable. Since SNe Arc is not included in Algorithm 3, the packet has not used any Arc. Hop count for p6(10, 51) using XY-routing is 6. There is no saving in hop counts for p6(10, 51).

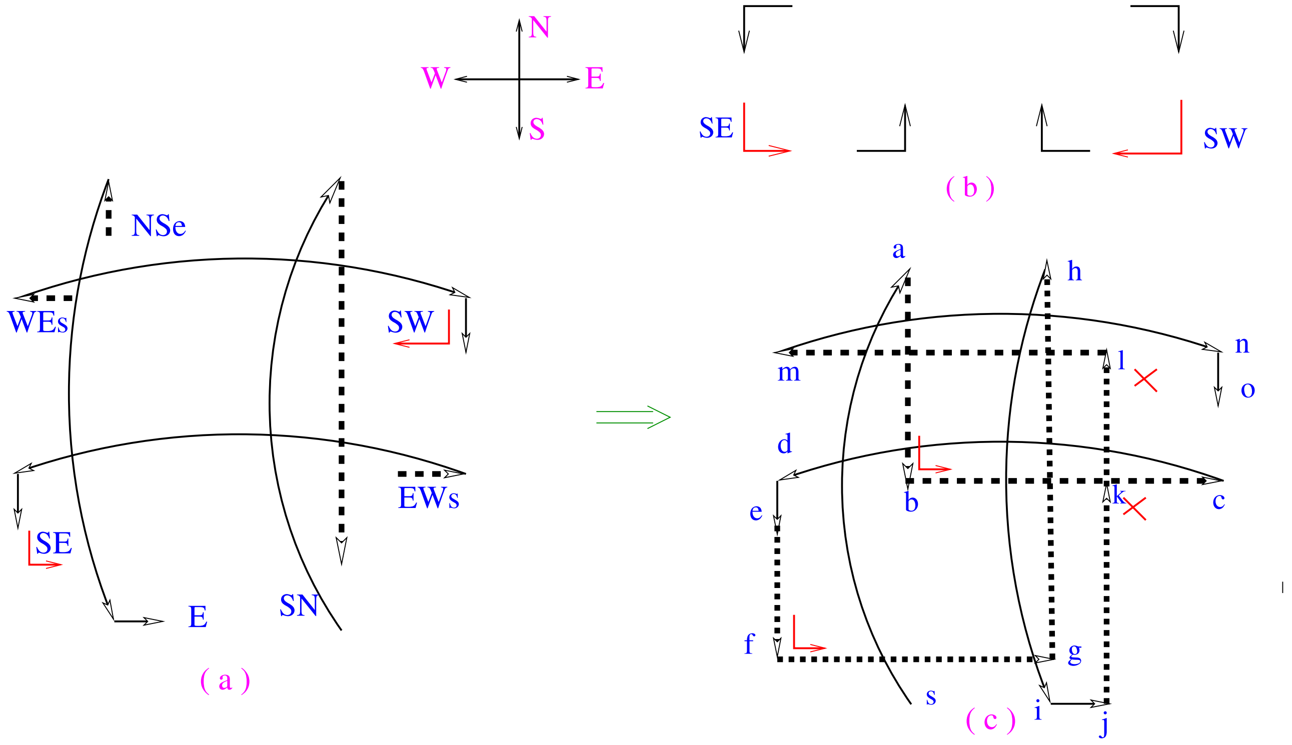


Figure 6.10: *Deadlock freedom: (a) EWs, WEs, NSe Arcs and SN wraparound channel in first hop (b) DDG representing deadlock freedom for EWs, WEs, NSe Arcs and SN wraparound channel in first hop with XY-Turns*

6.6.2.3 DDG to show Deadlock Freedom for Algorithm 3

The EWs, WEs and NSe Arcs with the SN wraparound channel in First Hop is shown in Fig. 6.10(a). The WEs Arc introduces SW Turn and the EWs Arc introduces SE Turn. The resultant Turn model due to these additional Turns is shown in Fig. 6.10(b). Since no cycle formation is possible as per Turn model, the first condition of Theorem 6.2.1 for deadlock freedom is satisfied.

For checking the second condition of Theorem 6.2.1, let us consider a directional dependency graph using EWs, WEs and NSe Arcs in Fig. 6.10(b). Let us check if deadlock is possible by resultant effects of EWs, WEs, NSe Arcs and SN wraparound channel. Let a sequence of packets produce a directional dependency via vertices $s \to a \to b \to c \to d \to e \to f \to g \to h \to i \to j \to k \to l$. For creating a cycle involving EWs Arc a NE Turn is needed at some vertex k. Similarly, A NW turn is needed at a point k for spreading the dependency into a WEs Arc. Since both NE and NW turns are not permitted in Algorithm 2, cycle creation is not possible using Arcs and Turns.

Both the conditions of Theorem 6.2.1 are satisfied for Algorithm 3. Therefore, Algorithm 3 using EWs, WEs, NSe Arcs Arcs and SN wraparound channel with XY-routing in the Mesh sub-network is deadlock free.

6.7 Experimental Results

In this section, we have evaluated the effectiveness of the proposed algorithms for Torus NoC in terms of saving in hop counts. We have also checked for the absence of deadlock for the proposed algorithms by applying different traffic patterns to Torus NoCs of varying grid sizes. The CFSM based simulation framework is used for all the experiments on the proposed deadlock free routing algorithms for Torus. In case of deadlock, it would reports deadlock with an exact deadlock scenario. If there is no deadlock, all packets gets delivered and simulation completes reporting the savings in hop counts. The hop counts saved for a packet with given source and destination is calculated as

Hop counts saved = (Manhattan distance between source and destination) - *(Actual distance traversed using wraparound channels).*

Manhattan distance is simply the XY-distance between source and destination of a packet. Alternately, Manhattan distance is nothing but the hop count between source and destination of a packet using XY-routing in Mesh sub-network. All the experiments on the proposed algorithm are performed in an Intel Core i5 3.20GHz, 8GB RAM machine. All the experimental results on hop count saving percentage presented in this section are the average of ten random simulation results.

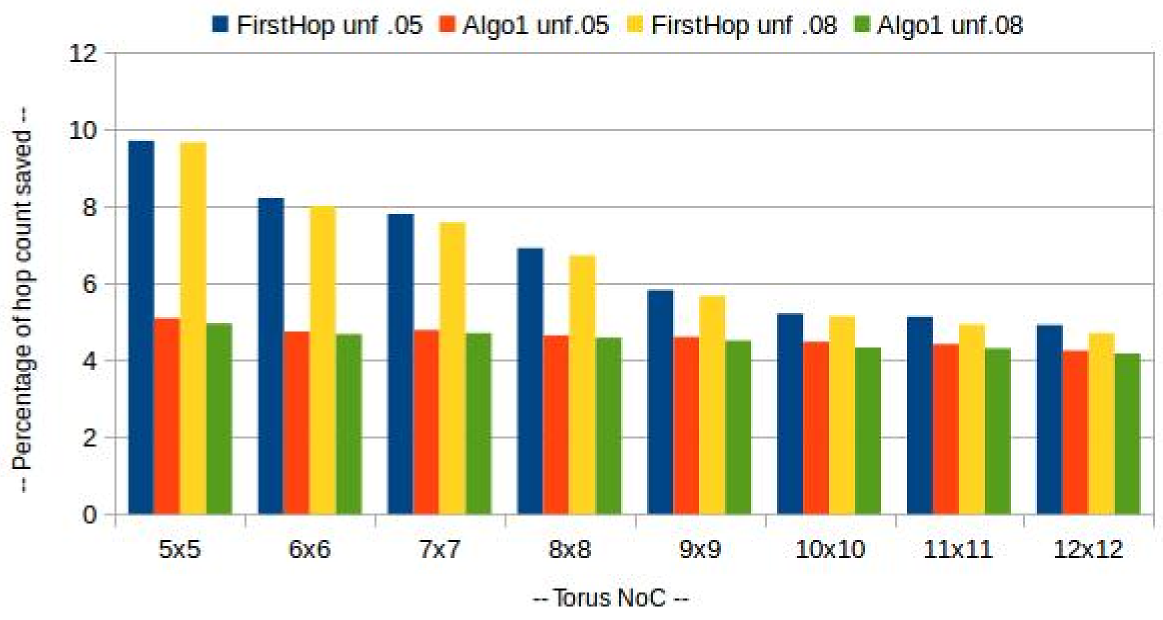


Figure 6.11: *Percentage of Hop counts saved by FirstHop Algorithm and Algorithm 1 using uniform traffic.*

6.7.1 Comparing FirstHop Algorithm with the Algorithm 1

In the first experiment, uniform traffic with injection rate 0.05 and 0.08 are given as input to NoCs with different grid sizes. Simulation has been completed successfully for all the experiments without any deadlock. Thus, there is no contradiction of experimental result with the DDG analysis regarding deadlock freedom. The bar diagram in Fig. 6.11 shows the percentage of saving in total hop counts by FirstHop Algorithm and Algorithm 1 for Uniform traffic with injection rates 0.05 and 0.08. Saving in hop count by FirstHop algorithm is better in comparison to Algorithm 1. The results indicates that saving in hop counts by FirstHop algorithm decreases with the increase of NoC size. For Algorithm 1, saving in hop counts does not very significantly with the increase of NoC size.

Table 6.1: *Percentage of boundary routers in a Torus NoC*

Torus NoC	Total routers	Boundary routers	Boundary routers (%)
5x5	25	16	64 %
6x6	36	20	56 %
7x7	49	24	49 %
8x8	64	28	43 %
9x9	81	32	40 %
10x10	100	36	36 %
11x11	121	40	33 %
12x12	144	44	31 %

In case of the FirstHop algorithm, wraparound channels are applicable only for the packets injected from the boundary routers. Therefore, the hop counts saving by FirstHop Algorithm depends upon the percentage of traffic generated from boundary routers. Total N*N numbers of routers are present in a NxN Torus NoC. Out of them, N + (N-1)*2 + (N-2) numbers of routers are considered to be boundary routers that are connected via wraparound channels. By observing smaller and bigger NoCs in Table 6.1, it is clear that the percentage of the numbers of boundary routers for a smaller NoC is more than that of a bigger NoC. Therefore, there are possibilities for more packets to be injected from boundary routers in smaller NoCs. On the other hand, for bigger NoCs, less number of packets from overall traffic are likely to be injected from boundary routers. Therefore, effectiveness of wraparound channels in saving of hop counts decrease with the increase of NoC size in the FirstHop Algorithm as reflected from the experiments in Fig. 6.11. In case of Arc Model based algorithm, Arcs are applicable not only for packets generated from boundary nodes. Therefore, effectiveness of Algorithm 1 does not effected adversely with increase of NoC size in comparison to the FirstHop Algorithm.

6.7.1.1 Effects of the Percentage of Traffic Injected from the Boundary Routers

In this experiment we consider specific types of traffic where a fixed percentage of packets are injected from boundary routers, i.e., for a fixed percentage of packets the sources are

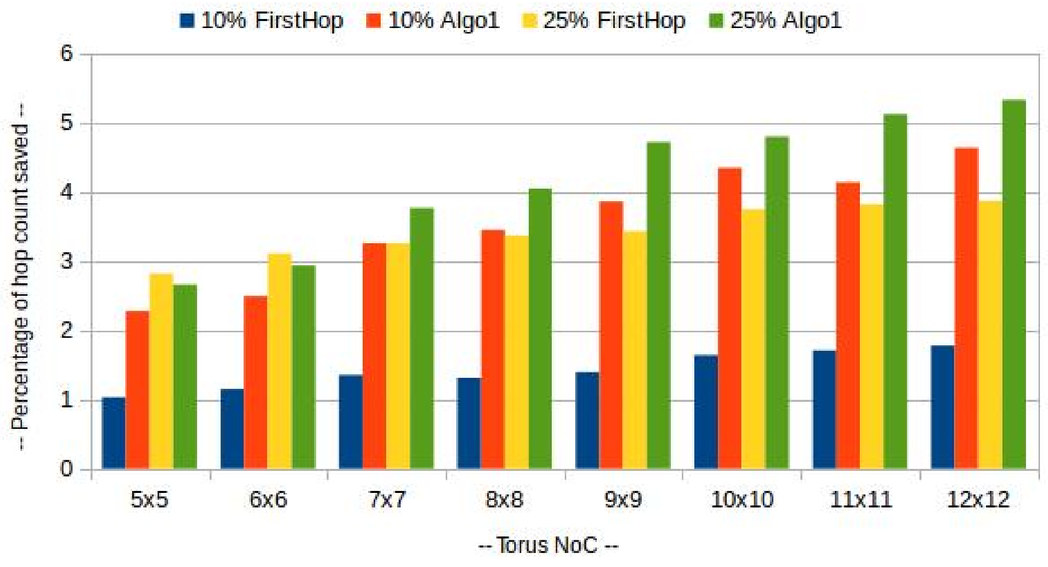


Figure 6.12: *Percentage of Hop counts saved by FirstHop Algorithm and Algorithm 1 using traffic samples with 10% and 25% packets injected from boundary routers.*

some boundary routers. We use such traffic to compare the saving in hop counts by the Algorithm 1 and the FirstHop Algorithm. For one experiment, we consider a traffic where 10% of the packets have sources in the boundary routers. For another experiment, we consider another types of traffic where 25% of total packets are originated from boundary routers. The saving of hop counts for both the experiments are shown in Fig. 6.12.

Experimental results show that Algorithm 1 saves more hop counts than that of the FirstHop algorithm in most of the cases. Wraparound channels are applicable in the FirstHop Algorithm only for the packets originating from boundary routers. Whereas, this restriction is not imposed for Arcs. Therefore, Arcs are applicable for more number of packets and saves more hop counts than FirstHop Algorithm. Another observation from Fig. 6.12 is, saving in hop count increases with the increase of NoC size for both algorithms. Wraparound channels from bigger NoCs are more effective in saving hop counts in comparison to smaller NoC. Therefore, saving in hop count increases with the increase of NoC size when percentage of traffic injected from boundary routers are fixed. This experiment shows that the proposed Algorithm 1 performs better than FirstHop algorithm when the percentage of boundary packets are less than equal to 10%. When percentage of boundary packets are 25%, the proposed Algorithm 1 performs better for 7x7 NoC on wards.

6.7.2 Comparisons of Algorithm 2 with Algorithm 1, Up*/Down* Algorithm and FirstHop Algorithm

The Algorithm 1 and Algorithm 2 are two proposed deadlock free Arc based algorithms for Torus. They use XY-Turns along with two Arcs and Three Arcs, respectively. In this experiment, we compare the saving in hop counts by Algorithm 2 with Algorithm 1, FirstHop algorithm [1] and Up*/Down* [10] algorithm. These experiments are performed on the uniform traffic with injection rate 0.05.

Table 6.2: *Hop count saved by Up*/Down*, FirstHop, Two Arcs (EWs + NSe) and Three Arcs (EWs+WEs+NSe) algorithms*

Torus NoC	**% Hop count saved**				**Wraparound channels used**			
	Up*/ Down*	**First Hop**	**Algo 1 (2 Arcs)**	**Algo 2 (3 Arcs)**	**Up*/ Down***	**First Hop**	**Algo 1 (2 Arcs)**	**Algo 2 (3 Arcs)**
5x5	- 21.18	9.71	5.09	7.61	60406	7650	4831	7250
6x6	- 26.03	8.23	4.75	7.12	52694	5267	3413	5024
7x7	- 28.56	7.81	4.79	6.49	64142	5788	3746	7105
8x8	- 29.80	6.92	4.65	6.16	57657	4537	3484	5120
9x9	- 33.89	5.83	4.61	6.11	66535	4613	4673	6739
10x10	- 36.90	5.21	4.48	6.04	59580	3854	3590	5005
11x11	- 39.11	5.13	4.42	5.91	68090	4073	4585	6346
12x12	- 40.03	4.92	4.25	5.66	63910	3473	3664	4922

In FirstHop Algorithm, resource dependency for deadlock is avoided by restricting wraparound channels only for packets that are injected from boundary nodes [1]. Since the number of boundary routers decrease with the increase of NoC size, saving of hop counts also decrease with the increase of NoC size for the FirstHop algorithm as shown in Table. 6.2. In case of Up*/Down* routing, a spanning tree is generated from the topology under consideration. The path between any two routers in a Torus NoC are as per the path in the corresponding spanning tree [10]. As a result, longer paths with unnecessary wraparound channels are used in many scenarios. For example, in one possible spanning tree corresponding to Fig. 6.1, the path between two adjacent nodes 13 and 14 is 13 $\rightarrow$ 18 $\rightarrow$ 23 $\rightarrow$ 22 $\rightarrow$ 21 $\rightarrow$ 25 $\rightarrow$ 24 $\rightarrow$ 19 $\rightarrow$ 14. The number of times wraparound channels are used in the algorithms are also presented in Table. 6.2. The performance of Up*/Down* routing is worst among all the algorithms even though the algorithm uses wraparound channels heavily, as shown in Table. 6.2.

In case of Arc based Algorithms, since wraparound channels are applicable for packets injected from non boundary nodes as well, utility of wraparound channel decreases in slower speed with increase of NoC size as compared to FirstHop algorithm with XY-Turns. Therefore, decrease in saving of hop counts with the increase of NoC size are also slower for Arc based routing. In the Algorithm 1 that uses two Arcs, savings of hop counts are not as good as the FirstHop algorithm. In the improved algorithm that uses three Arcs (Algorithm 2), hop counts saving surpasses the FirstHop algorithm after 9x9 Torus NoC as shown in Table. 6.2. This experiment concludes that the Algorithm 2 saves hop count better than that of Algorithm 1 and Up*/Down* algorithm for all cases. From 9x9 NoC on wards, the Algorithm 2 saves more hop counts than that of FirstHop algorithm.

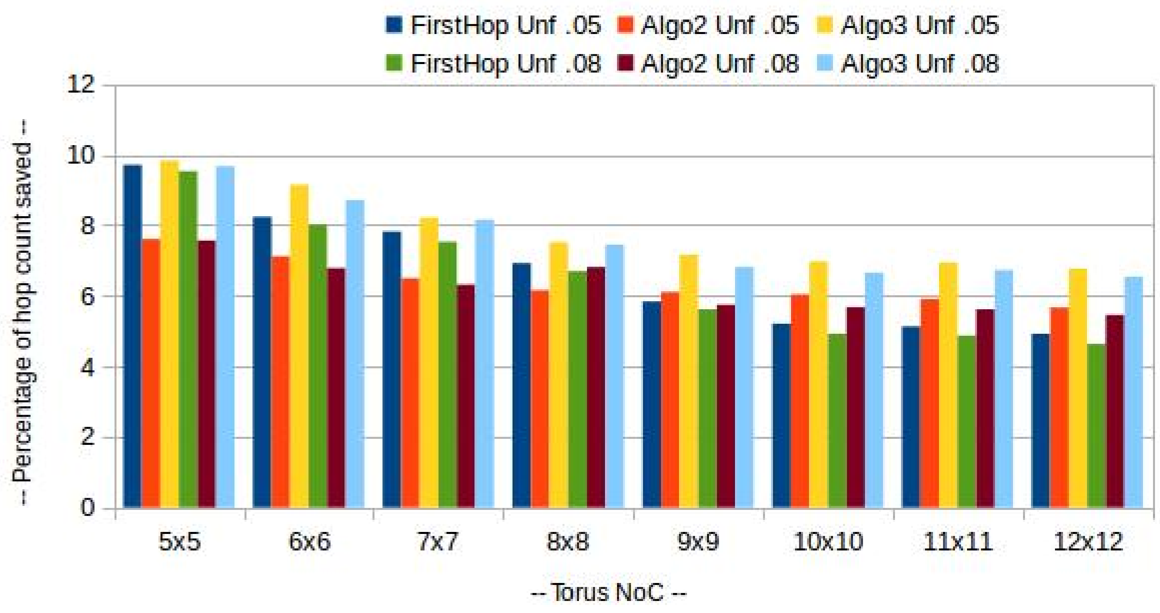


Figure 6.13: *Percentage of Hop counts saved by FirstHop Algorithm, Algorithm 2 and Algorithm 3 using uniform traffic.*

6.7.3 Comparison for the Algorithm 3 with Algorithm 2 and FirstHop Algorithm

In this experiment, we have considered Algorithm 2 which uses three Arcs and Algorithm 3 which uses three Arcs along with a wraparound channel. The wraparound channel is used only in the first hop, i.e., by applying the FirstHop approach only for one wraparound channel. These two algorithms are compared with FirstHop algorithm [1]. Since Up*/Down* approach for deadlock freedom has not saved hop counts as shown in Table 6.2, this approach is not considered in this experiment. Uniform traffic with injection rate 0.05 and 0.08 are used for this experiment. The percentage of hop counts saved by FirstHop Algorithm, Algorithm 2 and Algorithm 3 are shown in Fig. 6.13. Though the experimental results for Algorithm 2 is already shown in Table. 6.2, we have considered that algorithm again so that the changes in hop count savings become visible as compared to Algorithm 3. Experimental results show that the Algorithm 3 saves more hop counts than Algorithm 2 and FirstHop algorithm for all the cases. The experiment suggest that combining FirstHop approach with Arc based algorithm results in efficient routing algorithm for Torus NoC without using any additional resource.

6.7.3.1 Effects of the Percentage of Traffic Injected from the Boundary Routers

As usability of wraparound channels depends upon the traffic that are injected form or near to boundary routers, we have experimented on this behaviour for Algorithm 2 and Algorithm 3. Fig. 6.14 shows the saving of hop counts when two types of traffic where 10% and 25% packets of overall traffic are injected from boundary routers. Both the experiments reflect that, all three algorithm perform better when more traffics are injected from boundary nodes. If this percentage is reduced, saving in hop count by FirstHop algorithm reduced drastically. Algorithm 3 and Algorithm 2 save more hop count that First Hop algorithm when % of packets injected from boundary routers are low. Moreover, if the percentage of packets injected from the boundary routers are kept fixed irrespective of NoC size, the saving in hop count gradually increases with the increase of NoC size as shown in Fig. 6.14.

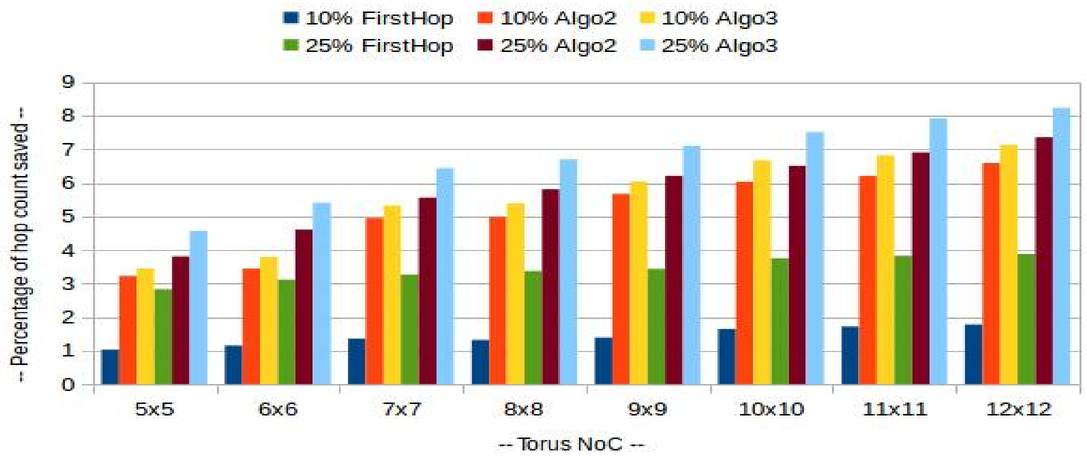


Figure 6.14: *Percentage of Hop counts saved by FirstHop Algorithm, Algorithm 2 and Algorithm 3 using traffic with 10% and 25% packets are injected from boundary routers.*

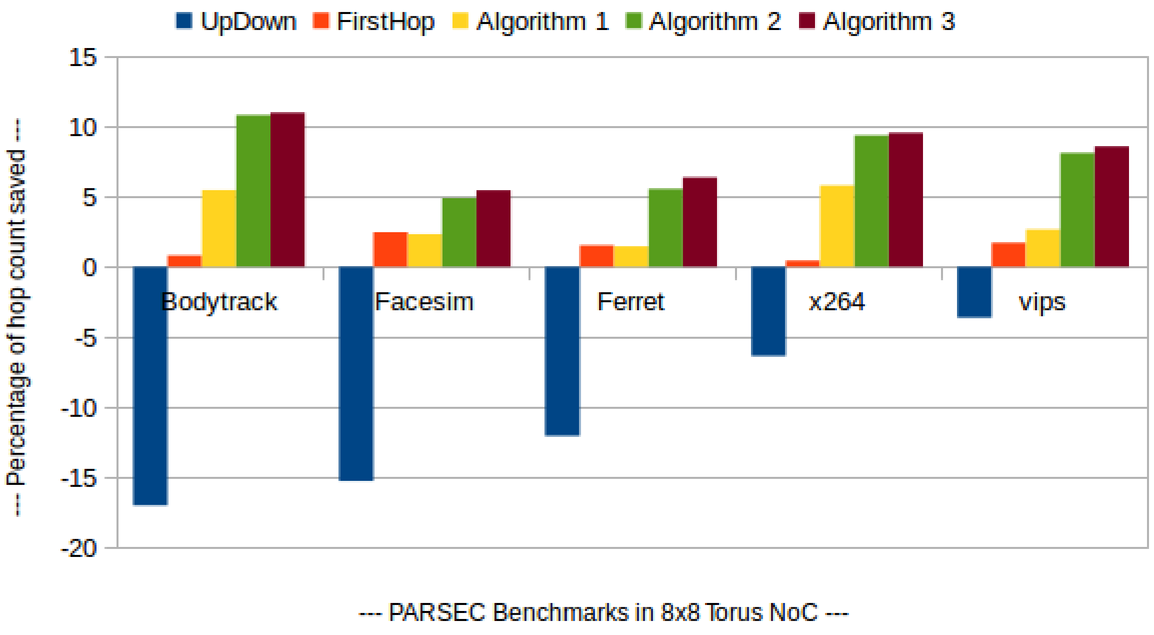


Figure 6.15: *Hop count saved by Up*/Down* Algorithm, FirstHop Algorithm, Algorithm 1, Algorithm 2 and Algorithm 3 using PARSEC benchmarks in an 8x8 Torus NoC.*

6.7.4 Hop Count Savings for PARSEC Benchmark Suites

We have performed experiments on hop count savings for all the five algorithms, Up*/Down* Algorithm, FirstHop Algorithm, Algorithm 1, Algorithm 2 and Algorithm 3, using traffic patterns from the PARSEC benchmark suites [104]. PARSEC benchmark traffic from five different workloads, namely Bodyatrack, Facesim, Ferret, x264, and Vips are applied to an 8x8 Torus NoC. No deadlock is detected in any of the algorithms. The experimental results are shown in Fig. 6.15. For the Up*/Down* routing, there is no saving in hop counts for all five PARSEC workloads as like the uniform traffic experimental results shown in Table. 6.2. For the FirstHop routing with the PARSEC benchmark suites, the saving in hop count is less than 2.5% for all cases. The hop count savings for Up*/Down* and FirstHop algorithms in PARSEC benchmark traffics are lower than the Arc Model based algorithms. Algorithm 1, Algorithm 2 and Algorithm 3 save more hop counts in the PARSEC benchmark traffics in comparison to FirstHop and Up*/Down* routing Algorithms. Algorithm 1 saves more than 5% hop counts for Bodytrack and x264 benchmarks. The hop count saved by Algorithm 2 and Algorithm 3 is almost at the same level for PARSEC benchmark suites. The FirstHop approach is not so effective in saving hop counts for PARSEC benchmark suites, as reflected in the experimental results. Considering the Bodytrack workload, the saving in hop counts is more than 10% for Algorithm 2 and Algorithm 3. Considering the x264 and Vips workload, the saving is more than 7% for Algorithm 2 and Algorithm 3. Considering the Facesim and Ferret, the saving is more than 5% for Algorithm 2 and Algorithm 3. Overall, our proposed Arc based algorithms outperform the Up*/Down* and FirstHop algorithms for the PARSEC benchmark suites.

6.8 Conclusions

Arc model is useful for deadlock avoidance in Torus NoC. Avoiding deadlock in Torus NoC without using dedicated buffer and VC is presented using *Arcs* in this work. Arc combinations along with a set of permitted Turns in the Mesh sub-network are used for developing routing algorithms for Torus NoC. The resultant algorithm is analysed for deadlock freedom using Directional Dependency Graph. Many Arc based algorithms are possible by considering different combination of Arcs along with a set of permitted Turns in the Mesh sub-network. As an application of *Arc model* three such algorithms are demonstrated using

Arcs with XY-Turns in the Mesh sub-network. In Algorithm 3 we have demonstrated the usability of Arcs along with FirstHop approach as well. Different traffic patterns are also applied to Torus NoC of different sizes using CFSM based simulation framework for checking the effectiveness of Arc based algorithms in saving hop counts. Experimental results show that the Algorithm 2 and Algorithm 3 save overall hop counts better than two well known competitive algorithms. This work opens up a development approach for many such deadlock free routing algorithms for Torus NoC by considering various well known deadlock free routing algorithms in the Mesh sub-network such as West-first, North-last, Negative-first etc. and a combination of Arcs from the Arc model.

7 Conclusions and Future Perspectives

In this chapter, we summarize the thesis by highlighting our contributions. The possible future directions of research based on this work also briefly highlighted in this chapter.

7.1 Summary of Contributions

7.1.1 FSM based NoC Model for Verification of Starvation

Modeling components of NoC in detail and maintaining synchronization between them using FSM are presented in this chapter. We have verified starvation, progress and transfer of packets between routers using our FSM models. Considering the complete NoC results in state space explosion problem. Therefore we have introduced the concept of active window in this work to expedite the verification process. Specifically, the verification of starvation in each active window is performed in parallel thread to reduce the overall verification time. Deadlock is a globally dependent property and complete NoC need to be considered for verifying deadlock. Therefore, verification of deadlock with detailed NoC model is found to be infeasible with the FSM models. Even application specific deadlock detection is difficult in the FSM based model since maintaining the global synchronization in the entire NoC at a time is a cumbersome process. Therefore, we have taken CFSM based approach in our next work where simulation is carried out.

7.1.2 Application Specific Deadlock Detection using CFSM based NoC Model

We have developed a CFSM based NoC model and used that model for developing a formal simulation framework in this work. In the CFSM based simulation framework, the synchronization amongst NoC components are maintained elegantly using the message queues of the CFSM model. This framework is useful in detection of application specific deadlock. Our framework can identify confirm deadlock on a given traffic pattern with a given routing algorithm. Once deadlock is detected, the exact deadlock scenario for resource dependency is also reported. The deadlock scenarios with detailed resource dependency help in understanding the causes of deadlock and in applying appropriate measures for its removal. We have also automated the CFSM model generation process for both Mesh and Torus NoC. With the help of this CFSM based simulation framework, we have experimentally shown deadlock in dynamic XY-routing [8] in Mesh and Torus NoC. Moreover, our experimental results have detected deadlock for XY-routing in Torus NoC. Our framework identifies that some of the deadlock warnings in Booksim simulator are not actually a deadlock. Experimental deadlock scenarios in Torus NoC with detailed resource dependency have shown that wraparound channels are the root cause of deadlock in Torus NoC. This leads us to analyse the Torus NoC further and come up with Arc model in our next work.

7.1.3 Deadlock Representation and Avoidance Approach in Torus NoC

We have proposed Directional Dependency Graph (DDG) for representing deadlock in a more informative way. The DDG is useful for predicting deadlock and showing of deadlock freedom. The Turn information, cycle information and wraparound channel information are incorporated in the DDG. Information about intermediate routers, that do not act as prime factor for causing deadlock, are eliminated in the DDG representation for the sake of simplicity. For avoiding deadlock in Torus NoC, we have proposed the Arc model. The Arc model is useful for avoiding deadlock in Torus without using VC or additional buffers. Behaviour of Arcs with respect to XY-Turns are analysed using DDG. Findings from these analysis are verified experimentally. We have presented certain thumb rules in terms of Lemmas for selecting deadlock free Arcs with respect to XY-Turns in the Mesh sub-network. Using this, we develop various routing algorithms for Torus NoC in the next chapter.

7.1.4 Deadlock Free Routing Algorithms for Torus NoC

We have presented three deadlock free routing algorithms for Torus NoC that do not use VC or additional buffer for avoiding deadlocks in this work. Our deadlock free routing algorithms in Torus NoC use a subset of Arcs and the permitted Turns as a deadlock free routing algorithm in the Mesh sub-network. The first algorithm is designed using two Arcs with XY-Turns. The second algorithm is designed using three Arcs with XY-Turns. In the third algorithm, we have used three Arcs and one wraparound channel. The wraparound channel is used in the first hop. Deadlock freedom for all the algorithms are shown using DDG. Experimental results show that our Arc based algorithms take shorter routing paths in comparison to Up*/Down* and FirstHop approaches.

7.2 Future Directions

The contribution of this thesis can be extended in a number of ways. Some of the possible future research directions of this thesis are stated below.

- We believe the FSM based modeling approach demonstrated in this work should successfully applicable for verification of system with manageable state space. We have shown how to model the hand shacking among asynchronous models of NoC using FSMs and CFSMs. Such hand shacking is widely used in many other Internet of Things (IoT) applications such as Vehicular Network, Smart Home, Security Monitoring Systems, Automated Farming Equipment, Automated Irrigation in Farming, Wearable Health Monitors, Shipping Container and Logistics Tracking System, etc.. Our modeling concept of handshaking can be adapted in such system as well.

- There is a scope for enhancing our CFSM based simulation framework to verify fault tolerant capability and verify livelock freedom along with deadlock freedom. Due to the manufacturing defect or any other faulty circumstances, it is important that an NoC should have fault tolerant capability. A fault tolerant algorithm is subject to change or misroute packets on encountering faulty connections. When a routing algorithm misroutes a packet, the livelock phenomenon can arise [108]. Some of the packets might end up in loops and suffer from livelock. Therefore, while designing a fault tolerant routing algorithm, avoiding both deadlock and livelock has importance. For livelock detection, the use of path tracing technique in our CFSM based simulation framework is one possible solution as used in [109].

- Another future direction of our work is to upgrade the CFSM based framework so that it can deal with more topologies like 3D-Mesh topology, butterfly topology, ring topology and other recently proposed topologies [110–112]. At present, two topologies namely 2D-Mesh and 2D-Torus are implemented in our CFSM based framework.

- Another possible future direction of our work is to study the behaviour of Arc model with respect to other routing algorithms. We have presented the behaviour Arc model with only XY-Turns. The example of routing algorithms presented in this work also considers only XY-Turns. There are scopes for developing better routing algorithms by considering other combination of Turns. Using Arc model with better Turn distribution based on NoC location, as applied in Odd-even [90] and Abacus routing [106], is another future direction of our work.

- In future, we also plan to automate the DDG based deadlock detection process while using Arc model and a set of Turns in Torus NoC. Given a set of permitted Turns and a set of Arcs as an input, the program should be able to detect possible cycles. If no cycle is possible, the program should declare deadlock freedom for the given set of Turns and Arcs.

- Security issues are major concern while developing NoC using different third party Intellectual Property (IP) [113]. Enhancing the formal model of NoC for detecting different security attacks in NoC and detection of malicious activity by an intentionally implanted hardware Trojan in NoC is another future direction from this work.

- Analysis of hardware overheads of Arc Model based algorithms and how to minimize the overhead is another future direction of our work.

7.3 Conclusions

Formal modeling and verification of a system like NoC is useful to reduce post fabrication maintenance costs by detecting bugs during design phase before the actual system is manufactured. Formal modeling at a detailed level for a tiny system with manageable state space is very promising for detecting defects during the early design phase. Whereas, state space explosion is the major challenge in verification of a vast system like NoC. In such cases, since model checking based verification using component-wise detailed model is not feasible, we have shown that simulation on a detailed level formal model is useful. Deadlock is so fatal to a system like NoC that its avoidance is a prime concern in designing a routing algorithm. A framework that can detect confirmed deadlock along with a detailed resource dependency scenario help in designing deadlock free routing algorithm for NoCs. This work shows that formal modeling approach could be helpful for finding deadlock and possibly other flaws as well in a routing algorithm and to rectify them. We have demonstrated such a deadlock free routing algorithm design approach for Torus NoC with the help of formal model and have presented three algorithms in the thesis. Many such algorithms are possible using the different combinations of Arcs and Turns. Turn distribution approach can also be applied to such algorithms for further improvement. We believe that our formal model based simulation framework would be useful for other applications like detection of livelock, verification of fault tolerant NoC, etc..

Dissemination out of this work

Journals:

1. **Surajit Das**, Chandan Karfa and Santosh Biswas. “Formal Modeling of Network-on-Chip Using CFSM and its Application in Detecting Deadlock”, *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* , vol. 28, no. 4, pp. 1016-1029, April 2020.

2. **Surajit Das** and Chandan Karfa. “Arc Model and DDG: Deadlock Avoidance and Detection in Torus NoC”, *IEEE Embedded Systems Letters (ESL)* . pp. 1-4, September 2021.

Conferences:

1. **Surajit Das**, Chandan Karfa and Santosh Biswas, “xMAS Based Accurate Modeling and Progress Verification of NoCs”, *VLSI Design and Test (VDAT)* , vol. 711, pp. 792–804, July 2017.

2. **Surajit Das** and Chandan Karfa, “Deadlock Avoidance in Torus NoC Applying Controlled Move via Wraparound Channels”, *Embedded Computing and System Design (ISED)*, Springer Singapore, pp. 87–99, Jan 2022.

3. **Surajit Das** and Chandan Karfa, “Formal Modeling and Verification of Starvation-Freedom in NoCs”, *Embedded Computing and System Design (ISED)* Springer Singapore, pp. 101–114, Jan 2022.

Journals under communication

1. **Surajit Das** and Chandan Karfa, “A Design Approach of Deadlock Free Routing Algorithms for Torus NoC”, Communicated in a Leading Journal

2. **Surajit Das**, Chandan Karfa and Santosh Biswas, “Accelerating NoC Verification using a Complete Model and Active Window”, Communicated in a Leading Journal

References

[1] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *JACM*, volume 41, pages 874–902, September 1994.

[2] Dally and Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[3] W.J. Dally and B. Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, first edition, 2003.

[4] G.E. Moore. Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE*, 86(1):82–85, jan 1998.

[5] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[6] Rajeev Balasubramonian, Norman P Jouppi, and Naveen Muralimanohar. *Multi-Core Cache Hierarchies.* Morgan and Claypool Publishers, 2011.

[7] Yongfeng Xu, Jianyang Zhou, and Shunkui Liu. Research and analysis of routing algorithms for noc. In *3rd International Conference on Computer Research and Development*, volume 2, pages 98–102, March 2011.

[8] Ming Li, Qing-An Zeng, and Wen-Ben Jone. Dyxy - a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *43rd ACM/IEEE DAC*, pages 849–852, July 2006.

[9] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *[1992] 19th ISCA*, pages 278–287, May 1992.

[10] M.D. Schroeder et al. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE J-SAC*, 9(8):1318–1335, 1991.

[11] J. Duato. A new theory of deadlock-free adaptive multicast routing in wormhole networks. In *Proceedings of 1993 5th IEEE Symposium on Parallel and Distributed Processing*, pages 64–71, Dec 1993.

[12] José Duato. A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distribute System*, 6(9):976–987, sep 1995.

[13] J. Duato and T. M. Pinkston. A general theory for deadlock-free adaptive routing using a mixed set of resources. *IEEE TPDS*, 12(12):1219–1235, 2001.

[14] M. Ebrahimi and M. Daneshtalab. Ebda: A new theory on design and verification of deadlock-free interconnection networks. In *ACM/IEEE 44th ISCA*, pages 703–715, 2017.

[15] M. Parasar, H. Farrokhbakht, N. Enright Jerger, P. V. Gratz, T. Krishna, and J. San Miguel. Drain: Deadlock removal for arbitrary irregular networks. In *HPCA*, pages 447–460, 2020.

[16] A. Ramrakhyani, P. V. Gratz, and T. Krishna. Synchronized progress in interconnection networks (spin): A new theory for deadlock freedom. In *ISCA*, pages 699–711, 2018.

[17] M. Parasar, N. Jerger, P. Gratz, J. Miguel, and T. Krishna. Swap: Synchronized weaving of adjacent packets for network deadlock resolution. In *2019 52nd IEEE/ACM MICRO*, 2019.

[18] Balaji Venu and Ashwani Singh. Formal verification methodology considerations for network on chips. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI-2012)*, pages 220–225, 2012.

[19] Harry D. Foster. Trends in functional verification: A 2014 industry study. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[20] Senwen Kan, Matthew Lam, Tyler Porter, and Jennifer Dworak. A case study: Pre-silicon soc ras validation for noc server processor. In *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, pages 19–24, 2016.

[21] Giuseppe Di Guglielmo, Franco Fummi, Graziano Pravadelli, Stefano Soffia, and Marco Roveri. Semi-formal functional verification by efsm traversing via nusmv. In *IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 58–65, 2010.

[22] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model verifier. CAV '99, pages 495–499, London, UK, UK, 1999. Springer-Verlag.

[23] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[24] Nan Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE ISPASS*, pages 86–96, April 2013.

[25] N. Binkert, B. Beckmann, G. Black, Steven K. Reinhardt, A. Saidi, A. Basu, J. Hestness, Derek R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[26] N. Agarwal, T. Krishna, L. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *2009 IEEE ISPASS*, pages 33–42, April 2009.

[27] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, Oct 1995.

[28] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu. Adaptive bubble router: a design to improve performance in torus networks. In *International Conference on Parallel Processing*, pages 58–67, 1999.

[29] V. Puente, C. Izu, R. Beivide, J.A. Gregorio, F. Vallejo, and J.M. Prellezo. The adaptive bubble router. *J. Parallel Distrib. Comput.*, 61(9):1180–1208, September 2001.

[30] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger. Leaving one slot empty: Flit bubble flow control for torus cache-coherent nocs. *IEEE Transactions on Computers*, 64(3):763–777, 2015.

[31] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. A generic model for formally verifying noc communication architectures: A case study. In *NOCS'07*, pages 127–136, May 2007.

[32] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. Executable formal specification and validation of noc communication infrastructures. In *Symposium on Integrated Circuits and System Design*, pages 176–181. ACM, 2008.

[33] S. Ray and R. K. Brayton. Scalable progress verification in credit-based flow-control systems. In *DATE*, pages 905–910, March 2012.

[34] V. A. Palaniveloo and A. Sowmya. Application of formal methods for system-level verification of network on chip. In *2011 IEEE Computer Society Annual Symposium on VLSI*, pages 162–169, July 2011.

[35] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration*, 38(1):69 – 93, 2004.

[36] Y. Chen, W. Su, P. Hsiung, Y. Lan, Y. Hu, and S. Chen. Formal modeling and verification for network-on-chip. In *The 2010 International Conference on Green Circuits and Systems*, pages 299–304, June 2010.

[37] Z. Zhang. *Verification Methodologies for Fault-Tolerant Network-On-Chip Systems.* Ph.D. Dissertation, Department of Electrical and Computer Engineering, The University of Utah, USA, 2016.

[38] D. E. Holcomb, A. Gotmanov, M. Kishinevsky, and S. A. Seshia. Compositional performance verification of noc designs. In *MEMCODE2012*, pages 1–10, July 2012.

[39] Surajit Das, Chandan Karfa, and Santosh Biswas. xmas based accurate modeling and progress verification of nocs. In *21st International Symposium on VLSI Design and Test (VDAT 2017) (Accepted).* IEEE, 2017.

[40] D. Brand and P. Zafiropulo. On communicating finite-state machines. volume 30, pages 323–342. ACM, April 1983.

[41] S. Das, C. Karfa, and S. Biswas. Formal modeling of network-on-chip using cfsm and its application in detecting deadlock. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4):1016–1029, 2020.

[42] Michael Huth and Mark Rayan. *Logic in Computer Science.* Cambridge University Press, 2004.

[43] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[44] K.K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publ., 1993.

[45] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An

open source tool for symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV '2002*, pages 359–364, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[46] Kunal Banerjee, Chittaranjan Mandal, and Dipankar Sarkar. Extending the scope of translation validation by augmenting path based equivalence checkers with smt solvers. In *18th International Symposium on VLSI Design and Test*, pages 1–6, 2014.

[47] Katell Morin-Allory, Marc Boulé, Dominique Borrione, and Zeljko Zilic. Validating assertion language rewrite rules and semantics with automated theorem provers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(9):1436–1448, 2010.

[48] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1):101–115, 2003. First Order Theorem Proving.

[49] F. Verbeek and J. Schmaltz. Automatic verification for deadlock in networks-on-chips with adaptive routing and wormhole switching. In *ACM/IEEE International Symposium*, pages 25–32, May 2011.

[50] F. Verbeek and J. Schmaltz. Easy formal specification and validation of unbounded networks-on-chips architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 17(1):1:1–1:28, January 2012.

[51] Sheila Nurul Huda. Semantic on promela dynamic process creation in concurrent systems. In *2016 International Conference on Instrumentation, Control and Automation (ICA)*, pages 44–47, 2016.

[52] Ying-Cherng Lan, Shih-Hsin Lo, Yueh-Chi Lin, Yu-Hen Hu, and Sao-Jie Chen. Binoc: A bidirectional noc architecture with dynamic self-reconfigurable channel. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 266–275, 2009.

[53] Pao-Ann Hsiung and F. Wang. A state graph manipulator tool for real-time system specification and verification. In *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No.98EX236)*, pages 181–188, 1998.

[54] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. xmas: Quick formal modeling of communication fabrics to enable verification. *IEEE Design Test of Computers*, 29(3):80–88, June 2012.

[55] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. Quick formal modeling of communication fabrics to enable verification. In *2010 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 42–49, June 2010.

[56] D. E. Holcomb, A. Gotmanov, M. Kishinevsky, and S. A. Seshia. Compositional performance verification of noc designs. In *MEMCODE2012*, pages 1–10, July 2012.

[57] Alexander Gotmanov, Satrajit Chatterjee, and Michael Kishinevsky. Verifying deadlock-freedom of communication fabrics. In *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'11, pages 214–231, Berlin, Heidelberg, 2011. Springer-Verlag.

[58] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. Advocat: Automated deadlock verification for on-chip cache coherence and interconnects. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1640–1645, March 2016.

[59] S. J. C. Joosten and J. Schmaltz. Automatic extraction of micro-architectural models of communication fabrics from register transfer level designs. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1413–1418, March 2015.

[60] S. J. C. Joosten and J. Schmaltz. Scalable liveness verification for communication fabrics. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[61] F. Burns, D. Sokolov, and A. Yakovlev. Gals synthesis and verification for xmas models. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1419–1424, March 2015.

[62] F. Verbeek and J. Schmaltz. Towards the formal verification of cache coherency at the architectural level. *ACM Trans. Des. Autom. Electron. Syst.*, pages 20:1–20:16, July 2012.

[63] P. van Wesel and J. Schmaltz. Formal micro-architectural analysis of on-chip ring networks. DAC '18, pages 94:1–94:6. ACM, 2018.

[64] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[65] S. Das, C. Karfa, and S. Biswas. xmas based accurate modeling and progress verification of nocs. In *VDAT 2017, June 29-July 2*, pages 792–804.

[66] Maryna Miroschnyk, Alexander Shkil, Dariia Rakhlis, Elvira Kulak, Inna Filippenko, and Mykyta Malakhov. Hardware implementation of timed logical control fsm. In *2020 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, 2020.

[67] Su-Fu Kuo and Cheng-Wen Wu. Symbiotic controller design using a memory-based fsm model. In *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, pages 874–879, 2018.

[68] Xiaomei Wan and Guohua Liu. Complexity of constructing fsm model of artifact lifecycle. In *2014 10th International Conference on Semantics, Knowledge and Grids*, pages 29–32, 2014.

[69] Ivan Zuzak, Ivan Budiselic, and Goran Delac. Formal modeling of restful systems using finite-state machines. In Sören Auer, Oscar Díaz, and George A. Papadopoulos, editors, *Web Engineering*, pages 346–360, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[70] Tao He and Huaikou Miao. Modeling and composition of web application components using extended fsm. In *2008 Fourth International Conference on Natural Computation*, volume 6, pages 363–368, 2008.

[71] Hrushikesha Mohanty, Jitesh Mulchandani, Deepak Chenthati, and R.K. Shyamasundar. Modeling web services with fsm modules. In *First Asia International Conference on Modelling Simulation (AMS'07)*, pages 100–105, 2007.

[72] Sunghyun Lee, Sungjoo Yoo, and Kiyoung Choi. An intra-task dynamic voltage scaling method for soc design with hierarchical fsm and synchronous dataflow model. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 84–87, 2002.

[73] Dohyung Kim and Soonhoi Ha. Asynchronous interaction between fsm and dataflow models. In *ICVC '99. 6th International Conference on VLSI and CAD (Cat. No.99EX361)*, pages 103–106, 1999.

[74] J. Rubin and C.H. West. An improved protocol validation technique. *Computer Networks (1976)*, 6(2):65 – 73, 1982.

[75] Mohamed G. Gouda and Ji Han. Protocol validation by fair progress state exploration. Technical report, Austin, TX, USA, 1984.

[76] Yao-Tin Yu and M. Gouda. Deadlock detection for a class of communicating finite state machines. *IEEE Transactions on Communications*, 30(12):2514–2518, December 1982.

[77] M.G. Gouda, E.G. Manning, and Y.T. Yu. On the progress of communication between two finite state machines. *Information and Control*, 63(3):200 – 216, 1984.

[78] P. Bogdan, M. Kas, R. Marculescu, and O. Mutlu. Quale: A quantum-leap inspired model for non-stationary analysis of noc traffic in chip multi-processors. In *Fourth ACM/IEEE NOCS*, pages 241–248, 2010.

[79] P. Bogdan and R. Marculescu. Non-stationary traffic analysis and its implications on multicore platform design. *IEEE TCAD*, 30(4):508–519, 2011.

[80] U. Y. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *IEEE TCAD*, 29(12):2001–2013, 2010.

[81] Z. Qian, D. Juan, P. Bogdan, C. Tsui, D. Marculescu, and R. Marculescu. Network-on-chips using learning-based support vector regression model. In *DATE*, pages 354–357, 2013.

[82] Z. Qian, D. Juan, P. Bogdan, C. Tsui, D. Marculescu, and R. Marculescu. A comprehensive and accurate latency model for network-on-chip performance analysis. In *ASP-DAC*, pages 323–328, 2014.

[83] U. Y. Ogras and R. Marculescu. Analytical router modeling for networks-on-chip performance analysis. In *DATE '07*, pages 1–6, April 2007.

[84] J. M. Martinez-Rubio, P. Lopez, and J. Duato. A cost-effective approach to deadlock handling in wormhole networks. *IEEE TPDS*, 12(7):716–729, July 2001.

[85] J. M. M. Rubio, P. Lopez, and J. Duato. Fc3d: flow control-based distributed deadlock detection mechanism for true fully adaptive routing in wormhole networks. *IEEE TPDS*, 14(8):765–779, Aug 2003.

[86] R. Al-Dujaily, T. Mak, F. Xia, A. Yakovlev, and M. Palesi. Embedded transitive closure network for runtime deadlock detection in networks-on-chip. *IEEE TPDS*, 23(7):1205–1215, July 2012.

[87] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M.S. Lam. The stanford dash multiprocessor. *Computer*, 25(3):63–79, 1992.

[88] T. G. Mattson and G. Henry. An overview of the intel tflops supercomputer. *Intel Technology J.*, 1:1–12, 1998.

[89] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu. Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 329–333, May 2009.

[90] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE TPDS*, 11(7):729–738, 2000.

[91] Terrence Mak, Peter Y. K. Cheung, Kai-Pui Lam, and Wayne Luk. Adaptive routing in network-on-chips using a dynamic-programming network. *IEEE Transactions on Industrial Electronics*, 58(8):3701–3716, 2011.

[92] Poona Bahrebar and Dirk Stroobandt. Adaptive and reconfigurable bubble routing technique for 2d torus interconnection networks. In *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2017.

[93] Antonio Robles-Gomez, Aurelio Bermudez, and Rafael Casado. A deadlock-free dynamic reconfiguration scheme for source routing networks using close up*/down* graphs. *IEEE Transactions on Parallel and Distributed Systems*, 22(10):1641–1652, 2011.

[94] J.C. Sancho, A. Robles, and J. Duato. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 15(8):740–754, 2004.

[95] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb 1993.

[96] Dally and Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[97] D. Xiang and W. Luo. An efficient adaptive deadlock-free routing algorithm for torus networks. *IEEE TPDS*, 23(5):800–808, 2012.

[98] Lizhong Chen and Timothy M. Pinkston. Worm-bubble flow control. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, page 366–377, USA, 2013. IEEE Computer Society.

[99] C. Carrion, R. Beivide, J.A. Gregorio, and F. Vallejo. A flow control mechanism to avoid message deadlock in k-ary n-cube networks. In *Proceedings Fourth International Conference on High-Performance Computing*, pages 322–329, 1997.

[100] Xiao Canwen, Zhang Minxuan, Dou Yong, and Zhao Zhitong. Dimensional bubble flow control and fully adaptive routing in the 2-d mesh network on chip. In *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, volume 1, pages 353–358, 2008.

[101] S. El-Ashry, M. Khamis, H. Ibrahim, A. Shalaby, M. Abdelsalam, and M. W. El-Kharashi. On error injection for noc platforms: A uvm-based generic verification environment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(5):1137–1150, 2020.

[102] Peter Linz. *An Introduction to Formal Language and Automata*. Jones and Bartlett Learning, fifth edition, 2012.

[103] W.J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, first edition, 2003.

[104] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, USA, 2011.

[105] Leonard Kleinrock. *Queueing Systems*. Wiley, vol2 edition, 1976.

[106] B. Fu et al. An abacus turn model for time/space-efficient reconfigurable routing. In *ISCA*, pages 259–270, 2011.

[107] M. Dehyadgari, M. Nickray, A. Afzali-kusha, and Z. Navabi. Evaluation of pseudo adaptive xy routing using an object oriented model for noc. In *2005 International Conference on Microelectronics*, pages 5 pp.–, 2005.

[108] M. Coli and P. Palazzari. An adaptive deadlock and livelock free routing algorithm. In *Proceedings Euromicro Workshop on Parallel and Distributed Processing*, pages 288–295, 1995.

[109] Vahid Janfaza and Elaheh Baharlouei. A new fault-tolerant deadlock-free fully adaptive routing in noc. In *2017 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, 2017.

[110] Chun-Ho Cheng, Hong-Lin Wu, Chi-Hsiu Liang, Chao-Chin Li, Chun-Ming Chen, Po-Lin Huang, Sang-Lin Huang, and Chi-Chuan Hwang. Equality noc: A novel noc topology for high performance and energy efficiency. In *2020 International Symposium on Computer, Consumer and Control (IS3C)*, pages 83–86, 2020.

[111] Hossein Doroud, Mahsa Ghorbanian, and Reza Sabbaghi-Nadooshan. Square topology: A novel topology for nocs. In *2011 NORCHIP*, pages 1–4, 2011.

[112] Yung-Chang Chang and Ching-Te Chiu. A study of noc topologies and switching arbitration mechanisms. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 1643–1647, 2012.

[113] R. Manju, Abhijit Das, John Jose, and Prabhat Mishra. Sectar: Secure noc using trojan aware routing. In *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, 2020.

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati 781039, India