# FPGA IMPLEMENTATION OF A NETWORK-ON-CHIP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İSMAİL ÖZSEL KILINÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2011

Approval of the thesis:

**FPGA IMPLEMENTATION OF A NETWORK-ON-CHIP**

submitted by **İSMAİL ÖZSEL KILINÇ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

**Examining Committee Members:**

Prof. Dr. Hasan GÜRAN _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Semih BİLGEN _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Gözde BOZDAĞI AKAR _____
Electrical and Electronics Engineering Dept., METU

M.Sc. Cemil KIZILÖZ _____
Electronical Design Department, ASELSAN, MGEO

**Date:** **September 7, 2011**

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last Name : İsmail Özsel KILINÇ

Signature :

## ABSTRACT

FPGA IMPLEMENTATION OF A NETWORK-ON-CHIP

Kılınç, İsmail Özsel

M.Sc., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Cüneyt Bazlamaçcı

September 2011, 93 pages

This thesis aims to design a Network-on-Chip (NoC) that performs wormhole flow control method and source routing and aims to describe the design in VHDL language and implement it on an FPGA platform. In order to satisfy the diverse needs of different network traffic, the thesis aims to design the NoC in such a way that it can be modified via a user interface, which changes the descriptions in the VHDL source code. Network topology, number of router ports, number of virtual channels, buffer size and flit size are the features of the designed NoC that can be modified. In this thesis, interfaces and operations of the blocks in the NoC are defined through block diagrams and algorithmic state machines. Verification of these blocks is performed not only on computer environment via simulations tools, but also in real world. To achieve this, source nodes generating dummy flits are also designed which communicate with our user interface via RS-232 generating flits according to the information provided by the user and monitoring the received flits from other source nodes in real-time.



Keywords: On-Chip Networks, NoC, FPGA

# ÖZ

FPGA İLE BİR YONGA-İÇİ-AĞ GERÇEKLEMESİ

Kılınç, İsmail Özsel

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Cüneyt Bazlamaçcı

Eylül 2011, 93 sayfa

Bu tez solucan deliği akış denetimi ve kaynak yönlendirme yapan bir Yonga-içi-Ağ (YiA) tasarlamayı ve bu tasarımı VHDL dilinde tanımlayıp FPGA platformunda uygulamayı hedefler. Farklı türdeki ağ trafik ihtiyaçlarını karşılamak için, bu tez tasarlanan YiA'yı VHDL kaynak kodundaki tanımı değiştiren kullanıcı arayüzü aracılığı ile değiştirilebilir olarak tasarlamayı hedefler. Ağ topolojisi, yönlendirici bağlantıların ve sanal kanalların sayısı, arabellek ve flit boyutu bu tasarlanan YiA'nın değiştirilebilir özellikleridir. Bu tezde, YiA'daki blokların arayüzleri ve işlemleri blok çizeneği ve algoritmik durum makinaları aracılığı ile tanımlanmıştır. Bu blokların doğrulaması sadece bilgisayar ortamında simülasyon araçları ile değil, aynı zamanda gerçek dünyada da yapılmıştır. Bunu gerçekleştirmek için, kullanıcı tarafından verilen bilgiye göre flit üreten ve gerçek zamanda diğer kaynak düğümlerden alınan flitleri gözlemleyen RS-232 aracılığıyla kullanıcı arayüzü ile iletişim kuran yapay flit üreten kaynak düğümleri de bu tez kapsamında tasarlanmıştır.

Keywords: Yonga-içi-Ağ, YiA, FPGA

*To My Family…*

## ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Cüneyt Bazlamaçcı for his guidance and support in my study. His inspiring suggestions and meticulous feedback in every step of this thesis enabled me to write it and made it an invaluable experience for me. It has been a pleasure to write this thesis under his guidance.

I would also wish to express my sincere gratitude to my thesis committee members Prof. Dr. Hasan Güran, Prof. Dr. Semih Bilgen, Prof. Dr. Gözde Bozdağı Akar and M.Sc. Cemil Kızılöz as they kindly accepted to share their invaluable comments and helpful suggestions with me.

I also thank to Ufuk Dinçer, Fatih Işık, Cemil Kızılöz and all my other colleagues in Aselsan who have contributed to improve myself in VHDL and digital design engineering skills.

I deeply express my gratitude to Serkan Naneci and my other dancing friends for their patience and encouragement during the preparation of this thesis. I am also grateful to Ali Yetkin Penbegül, Halil Ertuğrul, Fatih İzciler and Önder Altan for their willingness to share ideas and their belief in me. I am also greatly indebted to Çiler Akyüz for her friendship.

I express my dearest thanks to Şule Akdoğan who did not leave me alone getting through the hardest times. Her presence and her belief in me have been reassuring throughout this study. I am deeply indebted to her for her understanding, patience, respect in what I am doing and encouragement.

Last but not least, most special thanks and love go to my family, Nevin and Tacettin, who have supported me in everything I have done in my life. It could have been impossible to write this thesis without their love and support. They are the true possessors of my success.

**TABLE OF CONTENTS**

## LIST OF TABLES

TABLES

## LIST OF FIGURES

FIGURES

## LIST OF ABBREVIATIONS

ABBREVIATIONS

| | |
|---|---|
| ASM | : **A**lgorithmic **S**tate **M**achine |
| FPGA | : **F**ield **P**rogrammable **G**ate **A**rray |
| GUI | : **G**raphical **U**ser **I**nterface |
| IP | : **I**ntellectual **P**roperty |
| NOC | : **N**etwork **O**n **C**hip |
| SA | : **S**witch **A**llocation |
| SOC | : **S**ystem **O**n **C**hip |
| UI | : **U**ser **I**nterface |
| VA | : **V**irtual **C**hannel **A**llocation |
| VC | : **V**irtual **C**ircuit |
| VHDL | : **V**HSIC **H**ardware **D**escription **L**anguage |
| VHSIC | : **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit |
| VLSI | : **V**ery **L**arge **S**cale **I**ntegration |

# CHAPTER 1

# INTRODUCTION

As a result of developing technology, bits needed to describe the available information are increasing day by day. To keep pace with this excess demand, chips should process and transfer data more and more quickly. Until today the developers have made an effort to reduce the switching delay of flip-flops which directly affects the computation speed of the chip instead of dealing with the delay due to wires between the flip-flops, because the communication delay due to wires was negligibly small in comparison to computation delay due to flip-flops. However, the communication delay today is not negligible any more. Switching speed of the flip-flops increased dramatically so communication delay now starts to be a limit in the bit rate. As a consequence of these developments, designers now focus on communication inside the chip [1].

Most of the contemporary chips use a bus structure between the blocks inside the chip [2]. As the chip size expands and block count increases, wire lengths in the bus also expand. This results in an overhead in the transmission delay on the bus due to capacitive effect. The other way to connect the blocks inside chip is to link them by using dedicated wires between each two of them. From transmission speed point of view, this method is the best that can be achieved, but from productivity and scalability point of view, this is the worst. Since this is an ad-hoc method, for each new design with new blocks, a designer should redesign the links between them from scratch. The number of communication links is in the order of $N^2$ where N is the number of blocks. Hence designers should find an intermediate way that meets

the advantages of the two methods. The new method should be shared as a bus structure and should also be end-to-end using dedicated links. In other words it should be a *shared end-to-end* medium, which is the definition of a *network*. Therefore a simplified version of the general computer networks is proposed to be used for communication inside the chip and is called *Network-on-Chip (NoC)* [3].

At first view, NoCs and today's computer networks are similar to each other in terms of their main functions. Their purpose is to provide communication between distributed terminals over a shared medium. Depending on the type of the network, either a temporary connection between source and destination is established or packets which include source and destination addresses are generated to achieve this purpose. Even if the purpose is identical, NoCs differ from computer networks in their application domain. For computer networks, any terminal is far away from another one in terms of meters or even kilometers so that the medium is affected dramatically by noise and attenuation on the links (cables). However in NoCs, the scale decreases to the order of micrometers or even nanometers. As a result NoCs do not suffer from noise mechanisms and attenuation as much as computer networks. Due to these advantages, a reduced layered structure can be applied to NoCs. These layers can be considered as physical layer, network layer, transport layer and application layer [4]. Each layer is implemented by different units of the NoC architecture.

Because of its advantages over other communication structures, circuit designers have started to prefer on-chip networks. Although there are several proposed and implemented examples [5], unlike on-chip buses, a standard has not been defined for on-chip networks yet.

In this thesis, an example on-chip network is designed, implemented on FPGA and verified both on computer environment and on real-time. Implemented NoC performs wormhole flow control and source routing. In order to use it for different

types of network traffic, some properties of the NoC such as topology, virtual channel number, buffer depth and flit size can be modified. According to user needs and parameters, required VHDL source codes are generated via the provided user interface and NoCs created by these source codes can be used instead of other communication structures in existing systems. For example, in ASELSAN shared bus structures are used for system on chips with a single master unit and crossbar structures are used for systems with multiple masters. As a product of this thesis work, on-chip network structures are now available to be used with their supreme advantages. Even though some of the existing NoC examples are open source, they are not preferred to be used in military applications due to reliability and possible licensing issues in future. Also to keep up with the new improvements and to enhance our NoC according to new requirements that will arise in the future, developing our own design for such a recently introduced topic is preferred instead of importing existing designs.

In Figure 1.1, the road map followed to obtain our NoC is given. At first, NoC with specific characteristics is designed and described in VHDL language. Then, VHDL source codes are simulated on computer environment. As the next step, source nodes generating dummy traffic and serial communication interface to control and monitor the traffic generated by these nodes are designed. Together with these auxiliary blocks, the designed NoC is implemented on FPGA platform. After the verification of our NoC in real-time, VHDL source nodes are generalized and parameterized to span various topologies, buffer sizes, VC numbers and flit sizes.

Organization of the thesis is parallel to the design flow chart. Basic concepts and background information about on-chip networks will be given in the second chapter. Some of the existing NoC examples will also be summarized. In the third chapter, building blocks of the implemented NoC will be explained in detail. Their operation will be described through block diagrams and algorithmic state machines. Then, in the fourth chapter, verification of these blocks will be carried out both in

computer environment and in real-time. Waveforms obtained in simulation of specific scenarios will be presented. Two user interfaces developed for traffic generation in real time implementation and for creating our VHDL source codes will also be described. And finally, the thesis will be concluded with possible enhancements that can be carried out in the future.

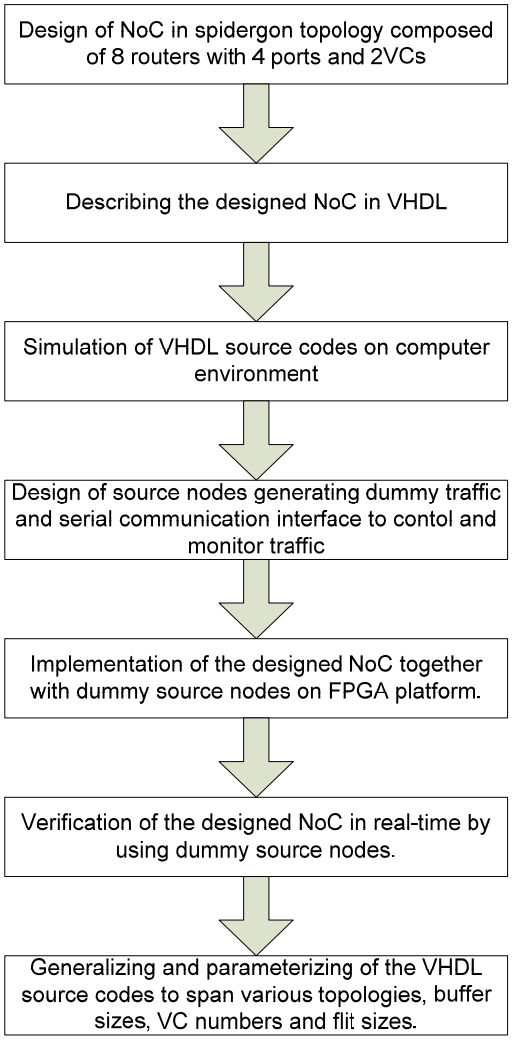


Figure 1.1: Flow Chart to be Followed

# CHAPTER 2

## ON-CHIP NETWORKS

A typical example of NoC equipped systems-on-chip, which is composed of three distinct types of units, is illustrated in the Figure 2.1. IP cores are the units whose communication needs will be met. They transmit and receive data packets throughout the network. Network interface units split packets generated by IP cores into flits which are ready to be injected into the network. At the same time they combine ejected flits and constitute packets ready to be transmitted to IP cores. Router blocks, fundamental components of the Network-on-Chip, lead flits from their sources to their destinations. During its transmission, a flit passes through multiple routers.

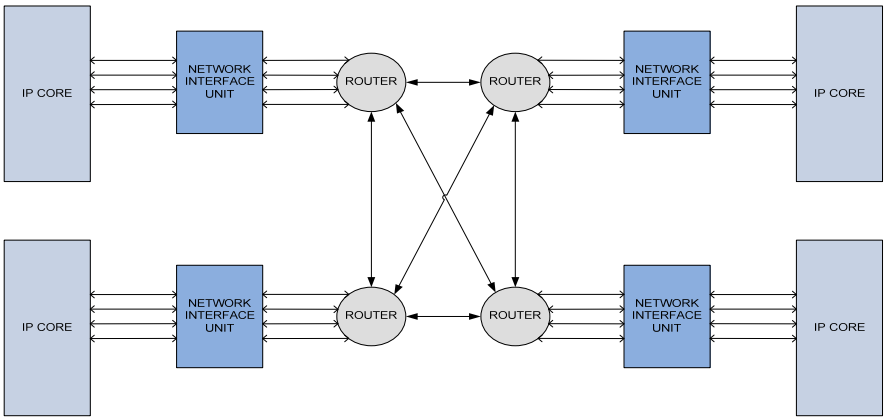


Figure 2.1: Typical NoC Equipped SoC Example

In this chapter, firstly on-chip networks will be compared to other widely used communication structures in systems-on-chips. Then, they will be analysed in relation to off-chip networks. In the third section, basic concepts of NoC will be examined; and finally, existing NoC examples will be reviewed.

## 2.1 COMMUNICATION STRUCTURES IN SYSTEMS-ON-CHIP

In Figure 2.2, examples of basic communication structures for Systems-on-Chip are given. Each structure has both advantages and disadvantages over others.

The bus structure is a well-known one, which can be implemented easily. However, as the number of units in the system increases, capacitive load on the bus also increases. This side effect causes an increase in power consumption and latency. Although poorly scalable, an intermediate solution between bus structure and dedicated point-to-point links is a crossbar, which still has some of the drawbacks of the bus.

Dedicated point-to point links offer optimum solution for bandwidth, resource usage, latency and power consumption [6]. Even though it is easy to design such a system, reusability and flexibility are problem areas for these structures. In order to add a new unit to the system, a designer has to remodel the existing units. Another disadvantage is the $O(n^2)$ growth of links with increasing number of units.

On the other hand, a data-routing network can be considered as the best solution for maximum flexibility and scalability. Identical networks can be used for diverse systems and identical routers can be used in diverse networks. Furthermore, because of point-to-point links, performance of such a system does not change with scaling and multiple transmissions can occur simultaneously inside the network. On the other hand, design complexity and latency due to contention are considerable

handicaps of on-chip networks. Table 2.1 presents a comparison between on-chip buses and on-chip networks.

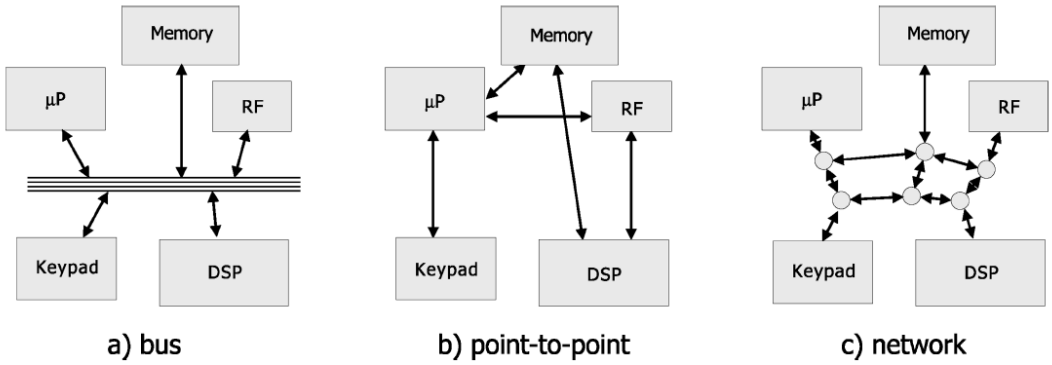


Figure 2.2: Communication Structures for Systems-on-Chip [6].

Table 2.1: Comparison of On-Chip Buses and On-Chip Networks [6].

| ***On-Chip Buses*** | | | ***On-Chip Networks*** |
|---|---|---|---|
| Due to paracitic capacitance, latency and power usage increase with each new attached unit. | - | + | Due to point-to-point one way links between routers, performance does not change with scaling. |
| All units share the limited bandwidth. Each new attached unit decreases the available bandwidth. | - | + | Bandwidth also increases with the addition of new units. |
| Arbiter is specific to the system and delay due to arbitration increases with number of masters. | - | + | Distributed routing decisions are made by reuseable routers. |
| Latency is constant after the bus is granted by the unit. | + | - | Latency can increase because of network contention. |
| Easy to design due to simple and well understood concepts. | + | - | More difficult to design due to new concepts. |

## 2.2 ON-CHIP NETWORKS VS. OFF-CHIP NETWORKS

On-chip networks operate as simplified versions of the off-chip networks due to unique VLSI constraints. These constraints limit the routers of the network in terms of area and power. Thus, light weight and hardware implemented protocols are implemented in routers, which are connected by short, reliable point-to-point links with high bandwidth. Routers and links are not affected by dynamic changes unlike the ones in off-chip networks [7].

On-chip communication is composed of well-defined interacting layers similar to computer networks. A simplified version of the ISO-OSI reference model can be adapted for typical on-chip networks. Although it is treated differently in various approaches, physical, network, transport and application layers are the usually applied ones. Applied layers, for example, Spidergon STNoC design [8] is illustrated in Figure 2.3.

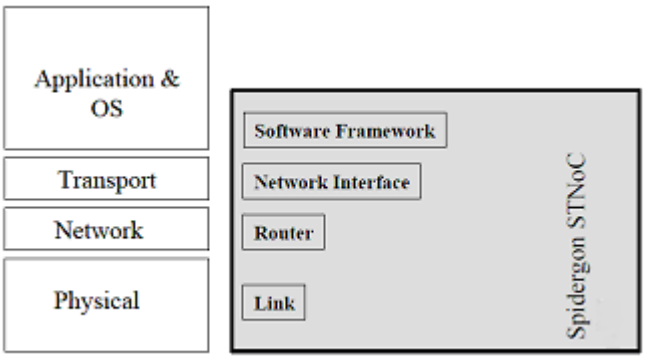


Figure 2.3: Layers in Spidergon STNoC Design [8].

## 2.3 BASIC CONCEPTS IN NETWORK-ON-CHIPS (NOCS)

### 2.3.1 Topology

Network topology describes the layout of the routers and connections between them. Topology affects network performance and cost dramatically. For instance, as the number of the routers that a flit must traverse increases, latency and power consumption also increase. Topologies are generally compared according to performance and cost metrics that are summarized in Table 2.2.

Topologies can be classified as direct and indirect. In direct topologies each router is attached to several other routers of the network and at least one IP core. Thus number of routers and IP cores are equal. Ring, octagon, spidergon, mesh and torus are major examples for direct topologies. Today, direct topologies are preferred on most designs. On the other hand, in indirect topologies there exist routers that are attached only to other routers. In these topologies number of routers is greater than the number of IP cores. Crossbar and multistage interconnect networks are examples for indirect topologies [8].

In this thesis study, only direct topologies are implemented. Connection diagrams and detailed information about these topologies will be given in section 3.2. Also influence of applied topology on performance and cost metrics will be discussed. For more information on topological issues, reader may see [9] and [10].

Table 2.2: Performance and Cost Metrics of Topologies

| *Metric* | *Definition* |
|---|---|
| **Network Size** | The number of IP cores connected to network. |
| **Network Cost** | The number of routers, cross points, communication links, wire length, wire density etc. |
| **Extendibility** | Possibility for enlarging the network without changing the topology. |
| **Node Degree** | The number of edges connected to a router. |
| **Network Degree** | Maximum node degree in the network. |
| **Edge Bisection Width** | Wire density in the network |
| **Network Diameter** | Maximum router count on the shortest path between any two IP cores. |
| **Avarage Distance** | Avarage router count on all the shortest paths between IP cores. |
| **Connectivity** | Ability to operate in the case of disabled components. |

### 2.3.2 Routing Algorithm

Figure 2.4 presents a classification of routing algorithms. Depending on the number of the destination, routing algorithms can be classified as unicast, in which packets have single destination, or multicast, where packets are destined to multiple nodes. Unicast routing has four categories according to the place where the routing decisions are made. In source routing, route is decided before the packet is injected into the network while in distributed routing, decision are taken inside the network

during the transmission of a packet. A single unit makes the decision in centralized routing. All routing decisions can be implemented either by using lookup tables or finite state machines.

Adaptability is another classification criterion. For the given source and destination nodes, if the routing algorithm always decides on the same route, this kind of routing is called deterministic, which generally chooses the shortest path between the nodes. XY, north first, south first, east first, west first, across first, across last routings are examples of deterministic routing. Unlike deterministic routing, adaptive routing takes also network traffic into account. Implementation of adaptive routing algorithms is more complex and costly than deterministic routing algorithms [11].

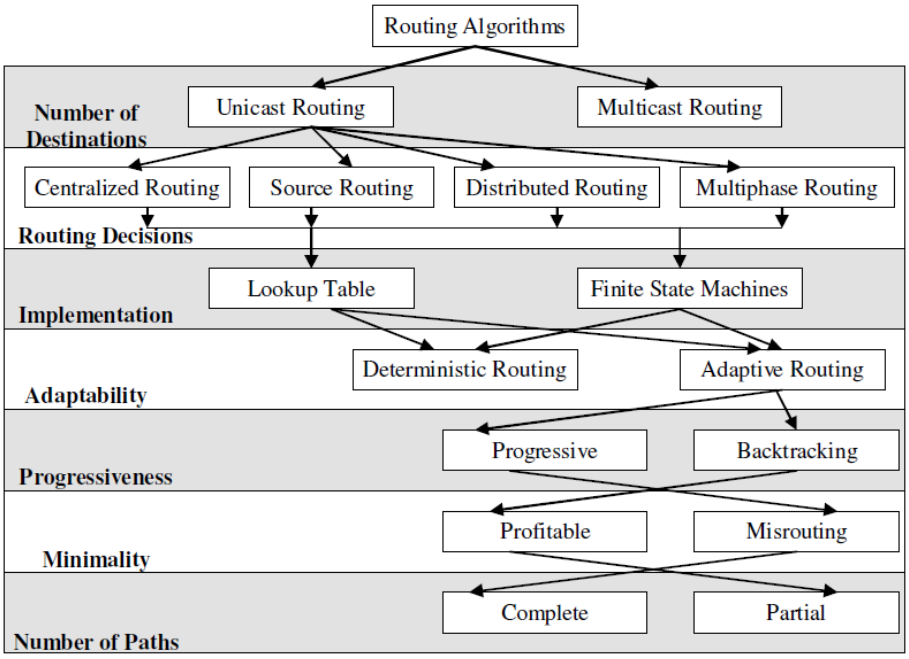


Figure 2.4: Classification of Routing Algorithms [12]

In progressive routing algorithms flits are not allowed to backtrack. Profitable (greedy) algorithms are also progressive because flits come closer to the destination in every routing decision. Deterministic routing algorithms are generally profitable. And final classification can be done as complete or partial routing according to number of paths considered.

In this thesis, source routing is implemented using finite state machines. Routing information is stamped to the header flit of the packet before it is injected into the network. More information about NoC routing algorithms exists in [13] and [14].

### 2.3.3 Flow Control (Switching) Methods

Buffer and link allocations are performed by flow control mechanisms which are classified by the granularity of the allocated resources. As shown in Figure 2.5 messages, before been injected into the network, are divided into packets, packets into flits and flits into phits. Size of the smallest segment determines the flow control method.

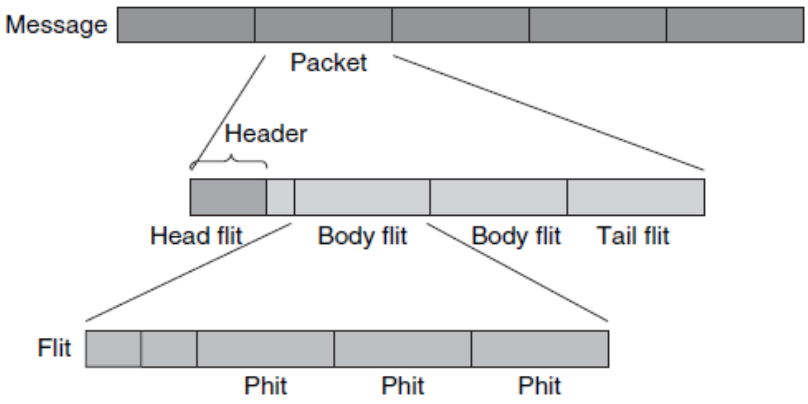


Figure 2.5: Messages, Packets, Flits and Phits [4]

A classification of flow control mechanisms is given in Figure 2.6. In circuit switching, resources are allocated for messages. Link pre-allocation takes place and links are reserved to the entire message. Thus, buffers are not needed at each router. A router architecture performing circuit switching is proposed in [15]. In store-and-forward and virtual-cut-through techniques messages are segmented into packets and packets are switched through the network as proposed in [16]. In store-and-forward, before forwarding to the next one, routers wait for the entire packet to be stored. This mechanism causes long delays in routers and needs large buffer space for the entire packet. In order to prevent delays, virtual-cut-through flow control starts to forward packet to the next router before the entire packet is stored. However, packet-sized buffers are still needed in this method. Flit-based flow control mechanisms emerged as a solution to large buffer area requirements of other flow control mechanisms.

Wormhole flow control operates in a way similar to the virtual-cut-through method. However in wormhole, availability of single flit sized empty buffer space in next router is enough for transmission. Therefore, main difference between these two methods is the necessary buffer space in the routers. During the transmission of a packet, header flit of that packet constructs a path in the network which is followed by other flits of the same packet. Although buffer allocation is done in units of flits, links are allocated for the entire packet which results in inefficient use of links. Then, if the header cannot proceed inside the network, whole packet is blocked Thus, allocated links are left idle. For more information please see [11] and [12].

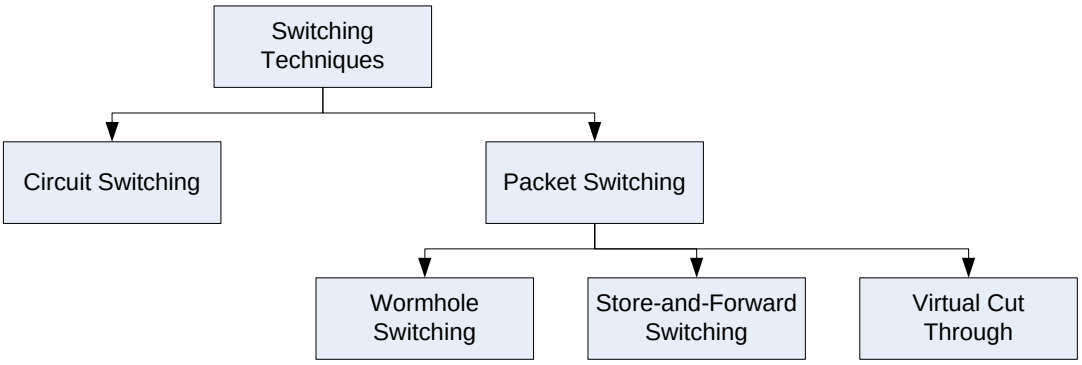


Figure 2.6: Classification of the Flow Control Mechanisms [12]

Wormhole flow control is the most common switching technique for both commercial off-chip network routers and on-chip network routers since it allows affordable and fast routers [11]. Also in this thesis study, wormhole flow control is preferred because of its convenience for power, timing and area constraints of on-chip networks. Store-and-forward, virtual-cut-through and wormhole mechanisms are compared in Figure 2.7 in terms of their timing performance.

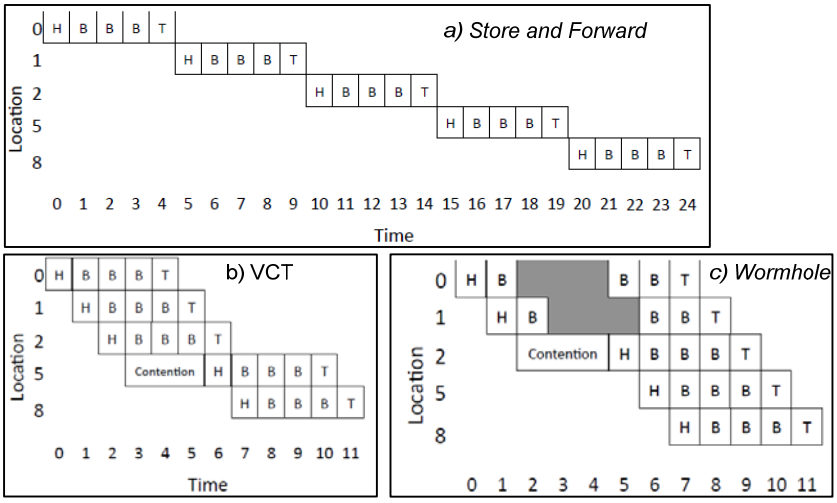


Figure 2.7: Timing Comparison of Flow Control Mechanisms [7]
a) Store-and-Forward b) Virtual-Cut-Through c) Wormhole

### 2.3.4 Virtual Channels

Virtual channels can be defined as multiple parallel queues (buffers) in routers. Virtual channels share the physical channel and arbitrate for it on a cycle basis. They are used to avoid deadlocks and head-of-line blocking problems. When a packet in one of the virtual channels is blocked, packets in other virtual channels can continue to be transferred. Thus, performance of networks is improved by the implementation of virtual channels in routers [17, 18].

Virtual channels can be implemented for all flow control methods mentioned in the previous section. However, since the integration of virtual channels to wormhole flow control mechanism solves the problem of inefficient link usage which is encountered especially in this mechanism, wormhole flow control method with VCs becomes the perfect choice for on-chip networks.

In this thesis the number of virtual channels can be selected via our configuration tool presented in this thesis. However, for the sake of simplicity, components of the NoC applying wormhole flow control with two VCs will only be examined in detail in the following chapters.

### 2.3.5 Buffer Organization and Backpressure

Network performance is directly related with the buffer organization. Buffers can exist both on the input ports and output ports. Output buffering is needed when speedup of a switch is required to be greater than one. Only input buffers are preferred in the routers of existing on-chip networks. Input buffering can be implemented by three different ways such as single-fixed-length queue, multiple fixed-length queues and multiple variable-length queues, which are shown in Figure 2.8. Multiple queues are used in routers with virtual channels. In variable-length queues, virtual channels share a large buffer according to a ratio changing dynamically with respect to network traffic [7].

Packet dropping is not allowed in most on-chip networks. Thus, a backpressure mechanism is required to stall the traffic. Availability of free space in buffers is signaled to neighbor routers via two commonly used mechanisms. In credit-based approach, number of available buffers is tracked [12]. In on-off backpressure approach, an on-off signal is generated according to a determined threshold. In this thesis study, multiple fixed-length buffers with credit-based backpressure mechanism are applied.

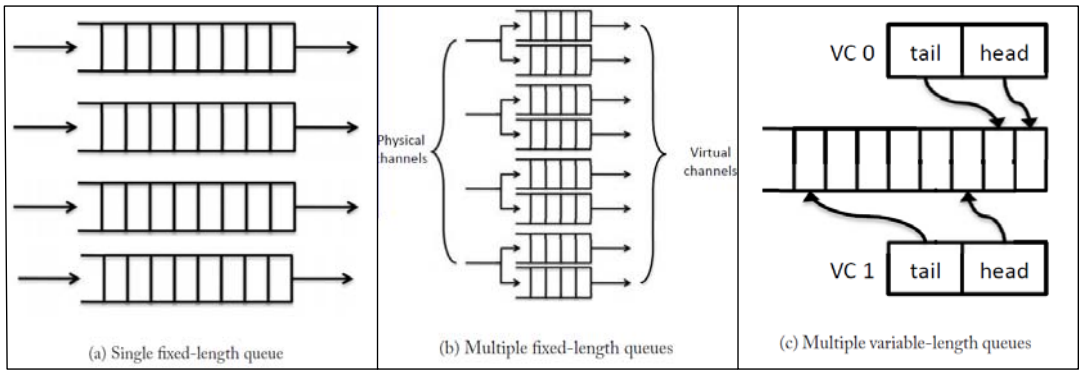


Figure 2.8: Input Buffer Types [7]

### 2.3.6 Allocators and Arbiters

Allocators are used to match multiple requests to multiple resources and arbiters are used to match multiple requests to a single resource. Allocators and arbiters are needed to distribute the resources of routers, namely the output virtual channels and switch ports. Thus two distinct allocation mechanisms are required. Virtual-channel allocation is performed for only header flits and VCs are allocated to entire packet, while switch allocation is performed for all the flits and switch ports are allocated on flit basis [7].

There exist several allocation and arbitration mechanisms. Round-robin, matrix (least-recently-used), first-come-first-serve, priority-based, priority-based-round-robin are examples for these mechanisms. Round-robin arbiter gives the lowest priority to the last served request in the next arbitration while matrix arbiter gives the highest priority to the least recently served request [12].

In this thesis study, round-robin arbitration is applied in both virtual channel allocation and switch allocation blocks of the routers implemented.

### 2.3.7 Switches

Packets are transported from input ports to output ports via switches. In this thesis, like other blocks, crossbar switches used in our routers are described as VHDL entities and synthesized for FPGA platforms. Since there exists no ready-to-use crossbar core in FPGA, these switches are composed of multiple multiplexers. As a result these switches cannot operate at higher clock frequencies as crossbar based switches. In Figure 2.9, a crossbar switch implementation composed of multiplexers is given and in Figure 2.10, a 5x5 crosspoint crossbar switch of *w* bits wide is illustrated.

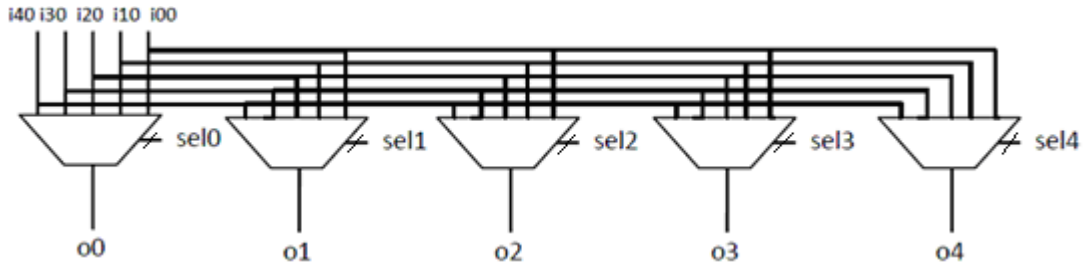


Figure 2.9: Crossbar Composed of Multiplexers [7]

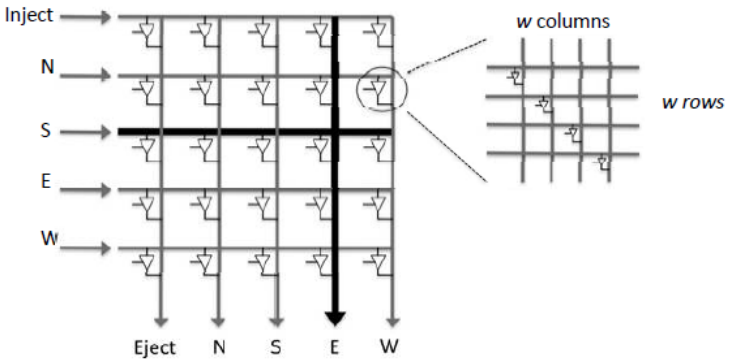


Figure 2.10: 5x5 Crosspoint Crossbar Switch [7]

## 2.4 EXISTING NoC EXAMPLES

In this section, some of the existing NoC implementations will be briefly reviewed.

SPIN [19], The Scalable Programmable Integrated Network, uses indirect fat-tree topology with two one-way data path in 32 bits width. Wormhole switching is implemented in SPIN.

QNOC [20, 21], The Quality of Service NoC, uses direct network with irregular mesh topology. Wormhole switching, credit-based backpressure mechanism and XY minimal routing are implemented by QNoC which is developed by Technion in Israel.

The SOCBUS NoC [15] uses direct 2-D mesh topology. Deadlock free circuit switching is applied in the SOCBUS NoC, which is developed at Linköping University.

The Nostrum NoC [22], which is developed at KTH in Stockholm, uses direct 2-D mesh topology. Store-and-forward switching is applied in the Nostrum NoC.

In the Æthereal NoC [23], wormhole switching is applied with contention free source routing. Both synchronous indirect and irregular topologies are supported by Æthereal NoC, which is developed by Philips.

Xpipes [24], developed by the Universisy of Bologna and Stanford University, applies wormhole switching and source routing. It contains a SystemC library of switches and links to create specific network components. Flit size, network diameter are some of the tunable parameters of the instantiated NoC. Different topologies such as mesh, torus, hypercube, clos and butterfly can be implemented by using Xpipes.

Spidergon STNoC [8], developed by STMicroelectronics, uses direct polygonal topology, which is obtained by generalizing the octagonal topology. Packet switching or circuit switching can be executed.

HERMES [25-27], developed by Faculdade de Informâtica PUCRS in Brazil, applies direct 2-D mesh topology. Wormhole switching is implemented with minimal XY routing algorithm. There exist several tunable parameters such as flit size, number of virtual channels, buffer depth. There are also several other NoCs with different properties developed by Faculdade de Informâtica PUCRS. These NoCs are listed in Figure 2.11.

| Parameters | Hermes | Hermes TB | Hermes TU | Hermes SR | Hermes CRC | Mercury | XHiNoC |
|---|---|---|---|---|---|---|---|
| Topology | 2D Mesh | 2D Torus | 1D Torus | 2D Mesh | 2D Mesh | 2D Torus | 2D Mesh |
| Virtual Channels | 1, 2 or 4 | 1 | 2 | 1 or 4 | 2 | 1 | unlimited (chosen at runtime) |
| Flit Width | 8, 16, 32 or 64 | | | 16 | 16 | 8, 16, 32 or 64 | 40 (32 data + 8 ctrl) |
| Buffer Depth | 4, 8, 16 or 32 | | | | | | 2 |
| Routing Algorithm | XY or West-first | West-first non miminal | Unidirec-tional XY | XY | XY | CG | XY |
| Scheduling Algorithm | Round Robin or Pricrity | Round Robin | Round Robin | Age Based | Round Robin | Round Robin | Round Robin |
| CRC | No | No | No | No | Yes | No | No |
| QoS | Yes | No | No | Yes | No | No | Yes |

Figure 2.11: Characteristics of the NoCs Developed by Faculdade de Informâtica PUCRS [27].

# CHAPTER 3

# IMPLEMENTATION OF NETWORK ON CHIP IN FPGA PLATFORM

In this thesis study, NoC which is performing wormhole flow control and shortest path across-first source routing is implemented in Field Programmable Gate Arrays (FPGA). The implemented NoC is composed of identical routers, which are connected in a definite topology. Routers also consist of sub-parts, which perform buffering, virtual channel allocation, switch allocation and switching functions. Topology, routers and sub-parts of router are implemented in different hierarchical blocks in FPGA.

In this chapter, these blocks will be examined in hierarchical order. By using block diagrams and state machine diagrams, their functions and operations will be clarified. But before the presentation of the FPGA blocks of NoC, flit structure that is recognized by the implemented NoC is given in the following section.

## 3.1 FLIT STRUCTURE

As was denoted in Chapter 2, wormhole flow control splits messages into packets and packets into flits. Packets are transmitted through the network in the form of single or multiple flits [11]. Flits are generated and injected into the network in the source node. They travel in the network via routers until they reach the destination node. In the destination node, flits are combined to form the packets again.

Flits include not the only data field, but also a control field [16]. Control field is necessary to express the characteristics of the flit to the network elements. As an example, routing is executed depending on the related information in the control field. This information needs to be in a definite place in the flits. Thus when a flit is received, by inspecting the bits in this place, the router can decide on how to route that flit.

The flit structure used in our implementation is given in Figure 3.1. Header flit consists of type, size, VCID, route data as the control field and payload as the data field. Body and tail flits have larger payload area instead of route data in header flits.

Type field consists of two bits, which are ‘start of frame’ (sof) and ‘end of frame’ (eof). These two bits identify four different types of flit. Header flit (b’10’) indicates the start of the packet and contains the routing information for the entire packet. A header flit is followed by a body or tail flit. Header flit can also indicate the end of packet in case of packet with a single flit (b’11’). It this case header flit is followed by the header flit of another packet. Body flits (b’00’) come between the header and tail flits. There can be one, multiple or no body flit in a packet. Body flit does not contain routing information and it is routed according to the routing information in the corresponding header flit. A body flit is followed by tail flit or another body flit. Tail flit indicates the end of packet. Tail flit is routed in the same way as a body flit. A tail flit is followed by the header flit of another packet. The length of the type field is 2 bits and does not change with the topology or the length of the rest of the flit (Figure 3.2).

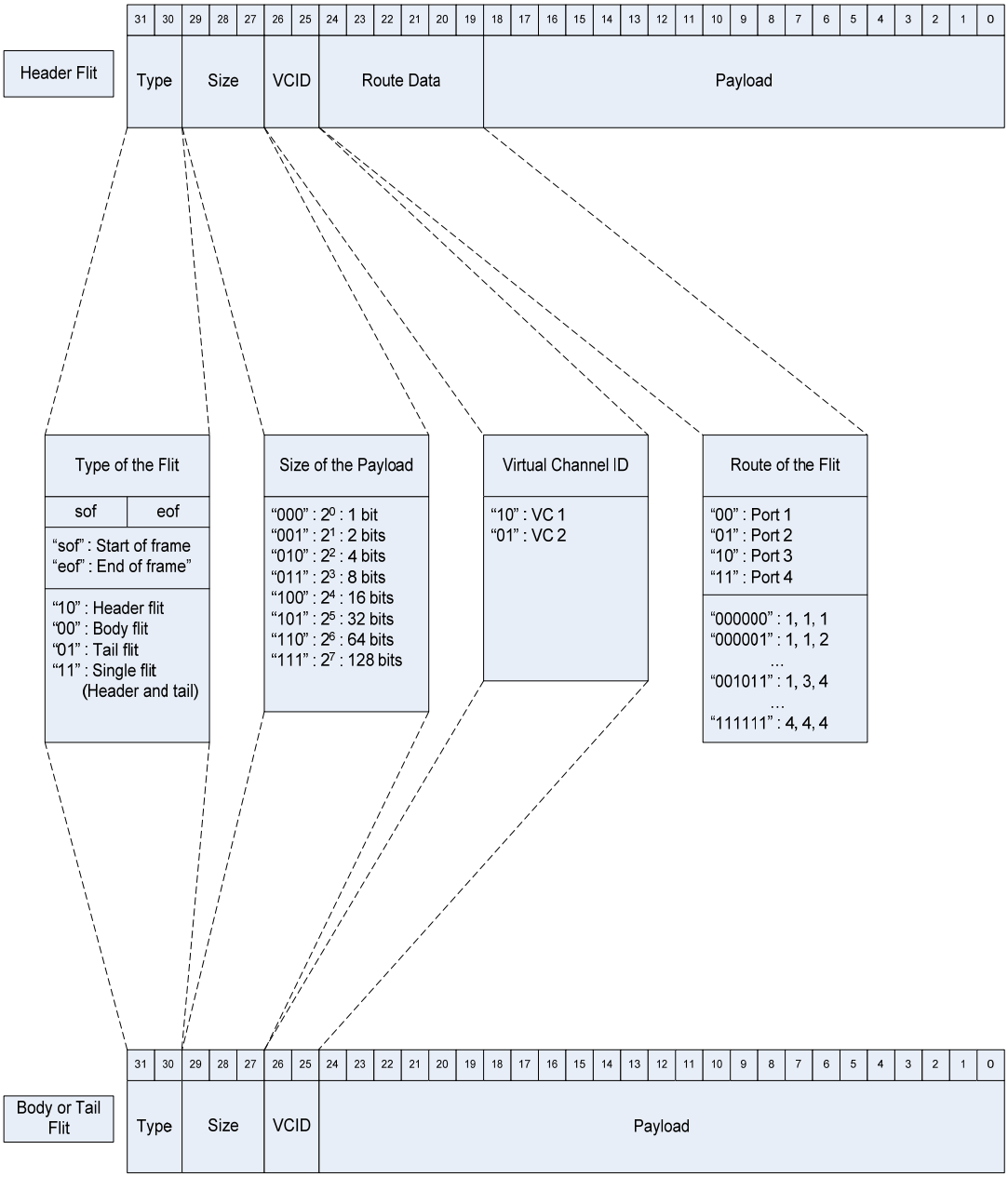


Figure 3.1: Flit Structure

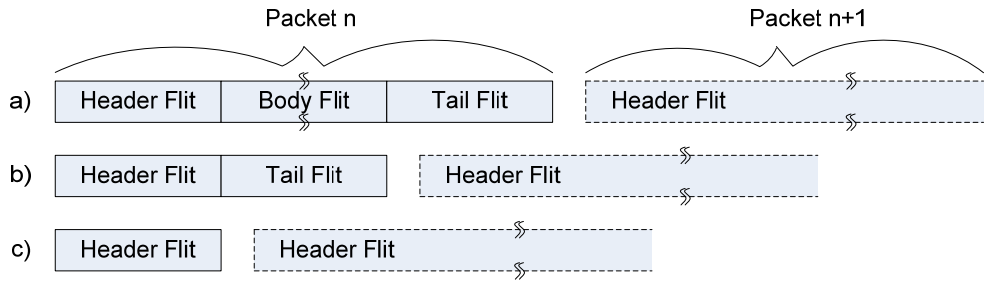


Figure 3.2: Flit Sequencing

Size field logarithmically shows the number of the meaningful bits in the payload area of that flit. Valid data in the payload area does not have to be equal to the whole length of the payload area. By using size field, destination node can understand the length of the valid bits. This information is not used by routers in the network. If the whole length of the payload area in a body or a tail flit is *n*, then the length of the size field is equal to $\lfloor \log_2 \lceil \log_2 (n) \rceil \rfloor + 1$. Since total lengths of all flits are equal and header flit contains routing information, whole payload area in a body or a tail flit is larger than the whole payload area in a header flit. So whole of the payload area in a body or tail flit is taken into consideration while calculating the length of the size field.

Virtual channel ID (VCID) field indicates virtual channel number to which the flit is going to be accepted in the next router. VCID field is assigned during the virtual channel allocation (va) for the header flit. Same VCID is also assigned to other flits of the same packet. So each flit of the same packet carries the same VCID while leaving the router. Since virtual channel allocation is going to be taken place again in the next router, assigned VCID may change in each router. The length of VCID field is equal to the number of virtual channels in input buffers. If there is no virtual channel, meaning a single buffer, VCID consists of only one bit.

Routing field includes output port numbers of all routers through which the flit is going to pass. This field is written into header flit by the source node according to shortest path across first routing. If *d* is the network diameter and *k* is the network degree for the applied topology, then the length of the routing field is equal to $(d+1)*\lceil \log_2(k+1) \rceil$. $(d+1)$ gives the maximum number of routers that a flit can pass through until reaching the destination node. $\lceil \log_2(k+1) \rceil$ gives the number of bits needed to represent ports of these routers. Each router uses first $\lceil \log_2(k+1) \rceil$ bits of this field to understand the output port for the incoming flit. While sending the flit to the next one, router rotates this field as shown in Figure 3.3.

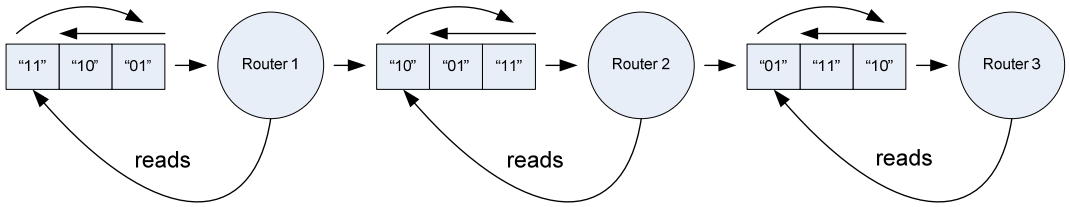


Figure 3.3: Rotation of Route Field for d = 2 and k = 3

Payload field contains the data to be transferred from source to destination. This field is not used and changed by routers during the transfer. Payload fields are combined to form packets at the destination node.

Number of virtual circuits and length of payload, which are two modifiable attributes in our design, determine the whole flit size. In Table 3.1, field lengths for body-tail flits are given for different virtual circuit numbers (*v*) and body-tail flit payload length (*n*).

Table 3.1: Flit Field Length vs. '*v*' and '*n*'

| **FLIT FIELD LENGTHS (*in bits*) vs. '*v*' and '*n*'** | ***v* = 2, *n* = 32** | ***v* = 2, *n* = 256** | ***v* = 4, *n* = 32** | ***v* = 4, *n* = 256** |
|---|---|---|---|---|
| **Type Field Length** | 2 | 2 | 2 | 2 |
| **Size Field Length ( =** $\lfloor \log_2 \lceil \log_2 (n) \rceil \rfloor + 1$ **)** | 3 | 4 | 3 | 4 |
| **VCID Field Length ( = *v*)** | 2 | 2 | 4 | 4 |
| **Payload Field Length ( = *n*)** | 32 | 256 | 32 | 256 |
| **Total Flit Length** | 39 | 264 | 41 | 266 |

On the other hand, topology determines the length of route field which also makes payload areas of header flits and body-tail flits differ from each other. This relationship between topology and route field will be explained in the following section in more detail.

## 3.2 TOPOLOGY BLOCK

Topology of a network describes how routers are connected to each other. Thus, topology block in our FPGA includes routers in network, interconnections among routers, interfaces between router and device nodes. In this section routers will be assumed as blackboxes and only connections defined by this block will be highlighted.

There exist four different topologies, which can be applied by our topology block. These are ring, spidergon, mesh and torus topologies, which was listed as direct

topologies in Chapter 2. Depending on the applied topology, some properties of the network, such as network degree and diameter, can be altered.

Node degree refers to number of edges connected to a router node where edge is defined as the connection between two router nodes. Network degree is the maximum node degree for all nodes in the network. The number of the ports in a router node is directly proportional to network degree. On the other hand, network diameter refers to the maximum number of routers on the shortest path route connecting any two network node. Network diameter depends not only to the applied topology but also to the network size. Network size is also defined by the topology block and can be changed via this block. Network size stands for the number of device nodes, which is also equal to number of router nodes in direct topologies. Network diameter affects the routing scheme together with network degree [8].

The relationship between topology and mentioned network properties is given in Table 3.2. As can be seen in this table, the number of the ports of routers is greater than network degree by one in all of the given topologies. Network diameter depends on network size ($N$), varying with topology.

Table 3.2: Network Properties vs. Topology

| | ***Ring*** | ***Spidergon*** | ***2-d m x n Mesh*** | ***2-d k x k Torus*** |
|---|---|---|---|---|
| **Network Degree ( = *k*)** | 2 | 3 | 4 | 4 |
| **Port Number ( = *k* + 1)** | 3 | 4 | 5 | 5 |
| **Network Diameter ( = *d*)** | $\lfloor N/2 \rfloor$ | $\lceil N/4 \rceil$ | $m+n-2$ | $2\lceil k/2 \rceil$ |

Since routing scheme changes with topology and network size, the length of route field in a header flit changes too. An incoming flit can be sent via any one of the $(k+1)$ ports. In order to find the outgoing port for a flit, $\lceil \log_2(k+1) \rceil$ bits are required by the router. In the implemented NoC, the shortest path across first route between source and destination is calculated and written to the header flit of the packet just before the flit is injected to the network. $(d+1)$ is the maximum number of routers that a flit can pass. As a result, $(d+1)*\lceil \log_2(k+1) \rceil$ bits are needed to represent the full route in a header flit. In Table 3.3 length of route field is given for ring and spidergon topologies for different network sizes (*N*).

Table 3.3: Network Properties vs. Topology and Network Size

| | ***Ring* (*N* = 8)** | ***Ring* (*N* = 12)** | ***Spidergon* (*N* = 8)** | ***Spidergon* (*N* = 12)** |
|---|---|---|---|---|
| **Network Degree ( = *k*)** | 2 | 2 | 3 | 3 |
| **Port Number ( = *k* + *1*)** | 3 | 3 | 4 | 4 |
| **Routing Bits Per Router** ( $=\lceil \log_2(k+1) \rceil$ ) | 2 | 2 | 2 | 2 |
| **Network Diameter ( = *d*)** | 4 | 6 | 2 | 3 |
| **Max. # Of Routers Passed ( = *d* + 1)** | 5 | 7 | 3 | 4 |
| **Length Of Route Field** ( = $(d+1)*\lceil \log_2(k+1) \rceil$ ) | 10 | 14 | 6 | 8 |

Connection diagram for the ring topology for $N = 8$ case is shown in the Figure 3.4. In this topology each router node has two edges adjacent to two router nodes. Third port of each router is connected to the related device node.

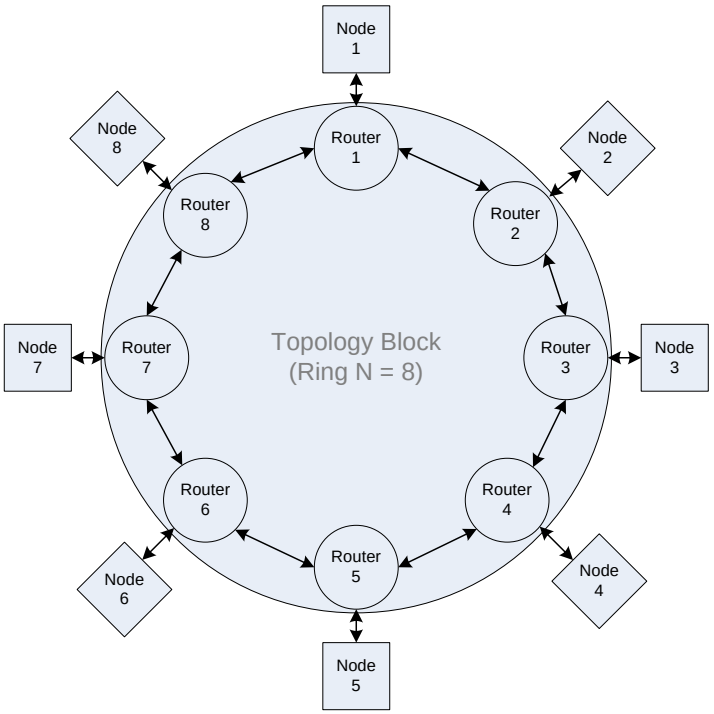


Figure 3.4: Ring Topology ($N = 8$)

In spidergon topology routers have three edges. The extra channels across the topology shorten the routes. Figure 3.5 presents the connection diagram for spidergon topology for $N = 8$ case. The only difference between this diagram and Figure 3.6 is the network size. Network size does not change the router node structure.

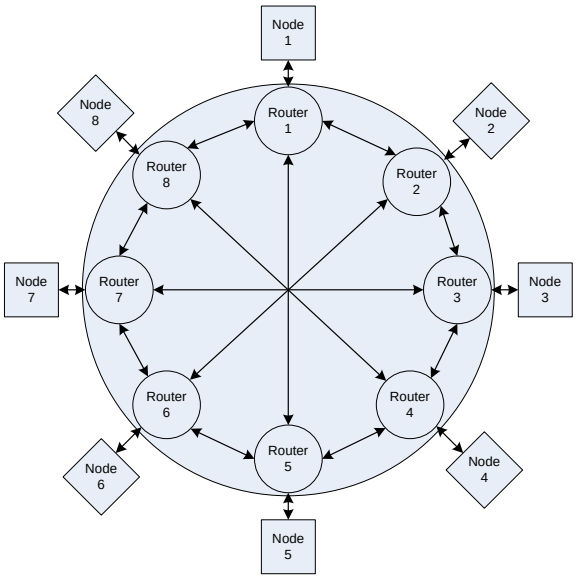


Figure 3.5: Spidergon Topology ($N = 8$)

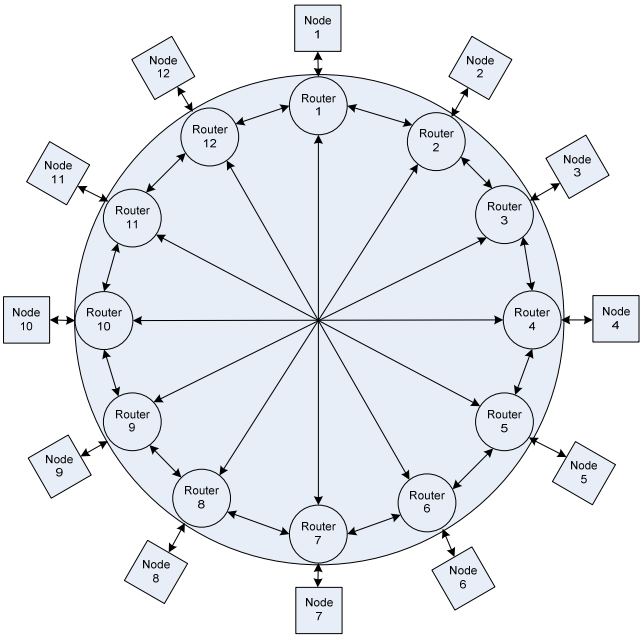


Figure 3.6: Spidergon Topology ($N = 12$)

Routers in mesh topology have 5 ports. Since node degree for routers at edges and corners is smaller than the network degree, some ports of these routers are unconnected. Sending flits to unconnected ports should be prevented by the routing scheme. A 4 x 3 mesh topology is shown in Figure 3.7. Corner routers (1, 4, 9 and 12) have 2 connections with other routers, while edge routers (2, 3, 5, 8, 10 and 11) have 3 connections. The remaining ones (6 and 7) have 4 connections, which is also the network degree.

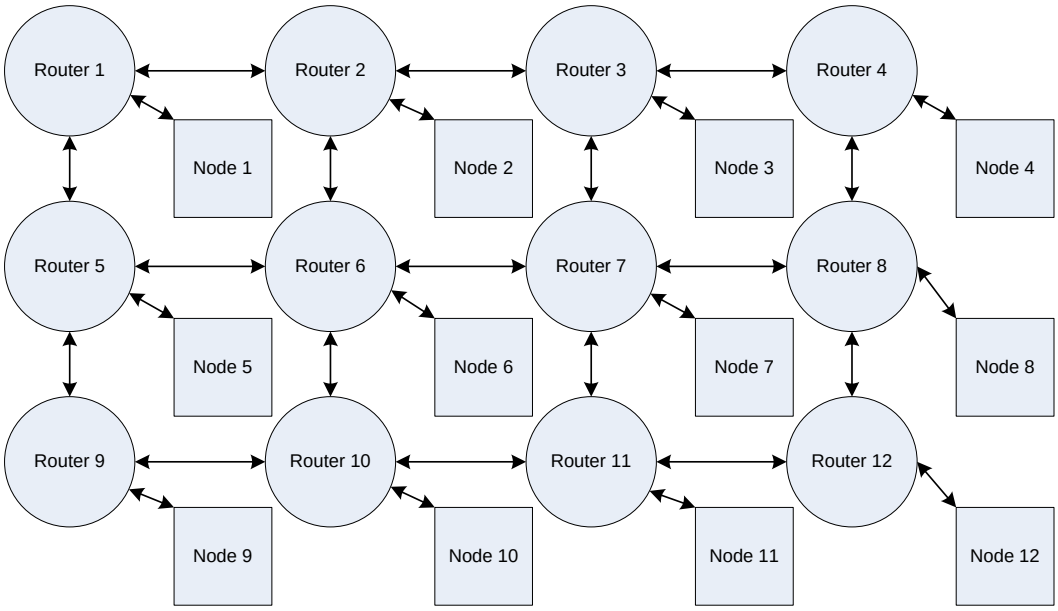


Figure 3.7: Mesh Topology ($N$ = 4 x 3)

In Figure 3.8, 4 x 4 torus topology is given. In this topology routers have 5 ports similar to the ones in mesh topology. Because edge and corner routers have across connections to other edge and corner routers, node degree for all routers is 4 meaning that each router node is connected with four other router nodes. Hence this condition shortens the routing as in the case of ring and spidergon.

As stated in the beginning of this section, the topology block in our implementation uses routers as components and defines connections between the routers. It also connects one port of each router node to outside as an interface to device nodes. This block does not contain any combinational or sequential logic. All the logical operations and decisions are taken place inside the router blocks which are going to be explained in the following section.

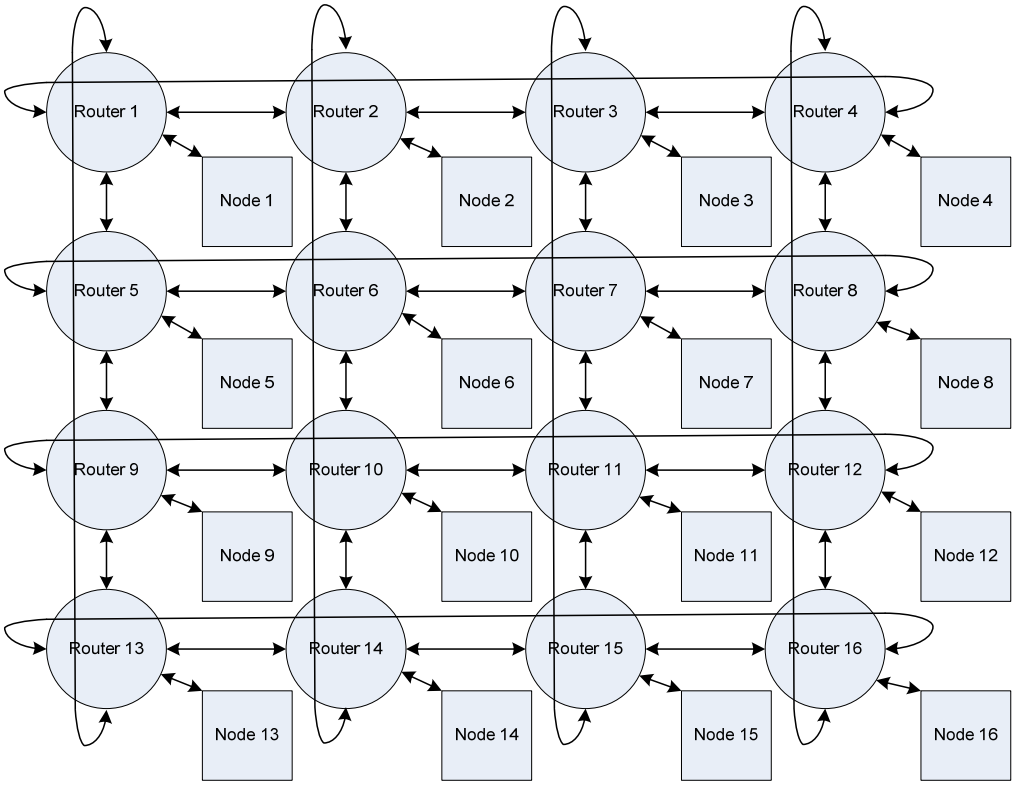


Figure 3.8: Torus Topology ($N = 4$ x 4)

## 3.3 ROUTER BLOCK

Router blocks are identical building blocks of the network. In order to create a network, routers are connected to each other through their identical ports. These ports are composed of two physical channels, i.e., incoming and outgoing channels. Each channel is composed of data bits whose count is equal to the length of whole flit and flow control bits which are requests and credits. Apart from the ones in the flits, these control bits are also required to ensure successful transfer of flits between nodes. Pinout diagram for a 4 port router can be seen in Figure 3.9.

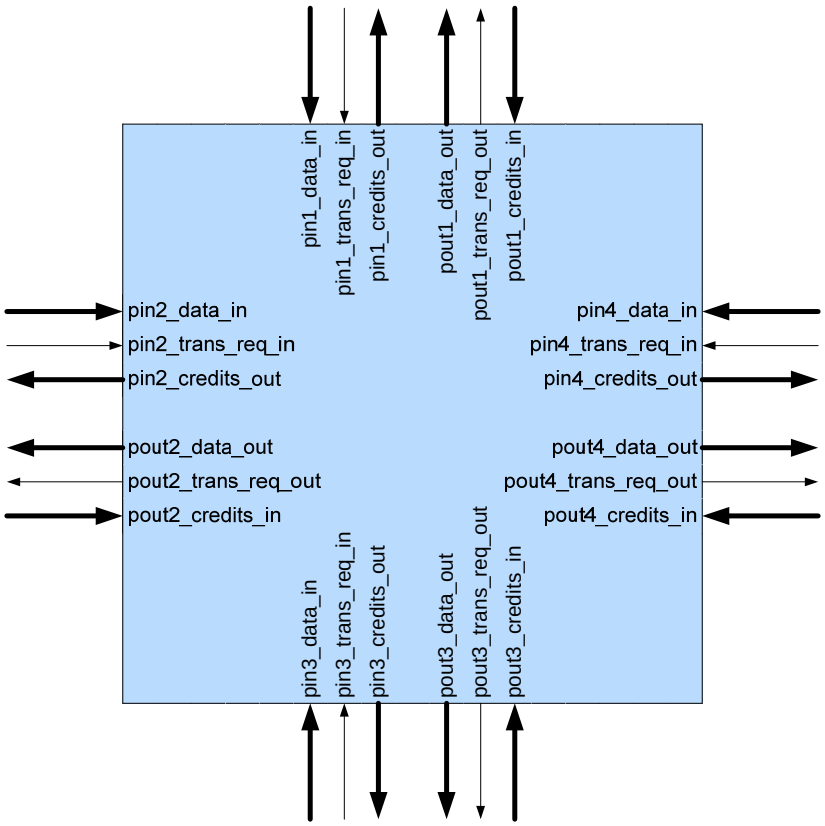


Figure 3.9: Pinout Diagram for a Four-Port Router

Function of a router is to direct the incoming flits from any one of the input ports to any one of the output ports excluding the input one. In order to achieve this, several mechanisms operate in a router, such as buffering, arbitration, allocation and switching. Mostly, routers encounter cases where multiple flits should be routed simultaneously. In these cases, if each flit is to be routed to different output ports, only a switching mechanism would be sufficient for routing. However, there exist cases where two or more flits are to be routed to the same output port. In such a situation, the permission to use the desired output should be given to one of the flits and other flits should be stored in the router. Here, an important problem arises: which flit should be routed and which one should be stored? Buffering, arbitration and allocation mechanisms are required to overcome this problem. In Figure 3.10, block diagram for a router with 4 ports and 2 virtual channels is given. In this figure different instances of each functional block are used. For example vc_1, vc_2, va_1 sa_1, sw etc… Only *va_req_out_x* and *va_ack_in* signals of vc_1, *sw_data_out* and *sw_dir_out_x* signals of vc_5, *sa_req_out_x* signals of vc_7 and vc_8, and *sa_ack_in* signal of vc_7 are shown in the figure. Please note that other blocks also have the same signals although not shown for simplicity.

**Pre_vc** blocks demultiplex the incoming data and request to **vc** blocks according to VCID field in the data (Shown as ***1*** on Figure 3.10). **Vc** blocks perform buffering and manage the entire operation. Initially these blocks generate request signals for output virtual channel allocation (*va_req_out_x*) (***2***). A **vc** block can only generate requests for output virtual channels of other ports, not for the port it belongs to. In this example, 2 *va_req_out_x* signals of each **vc** block are left unconnected and 6 *va_req_out_x* signals go to **va** blocks from each **vc** block. Similarly, each **va** block has 6 request inputs. Depending on the round-robin arbitration method, **va** blocks generate acknowledge signal for one of the requests and grant the corresponding output virtual channel for that request until the end of the packet. After receiving output virtual channel acknowledgment (*va_ack_in*) (***3***), **vc** blocks generate requests

for the switch allocation (*sa_req_out_x*) (***4***). Because of the same reason mentioned for *va_req_out_x* signals, one of these request signals is left unconnected and other three requests are connected to **sa** blocks. **Sa** blocks perform round-robin arbitration like **va** blocks and acknowledge the winner request. Unlike **va** blocks, which grant output virtual channel on packet basis, they grant the switch usage on a flit basis. When **vc** blocks get switch allocation acknowledgment (*sa_ack_in*) (***5***) and observe available buffer space (c*redit_in_x*) (***6***) on the next node, they transfer the data (*sw_data_out*) to the switch (**sw**) which multiplexes the data to one of the output ports according to select signals (*sw_dir_out_x*) (***7***).

These sub-blocks are explained in detail in the following sub-sections.

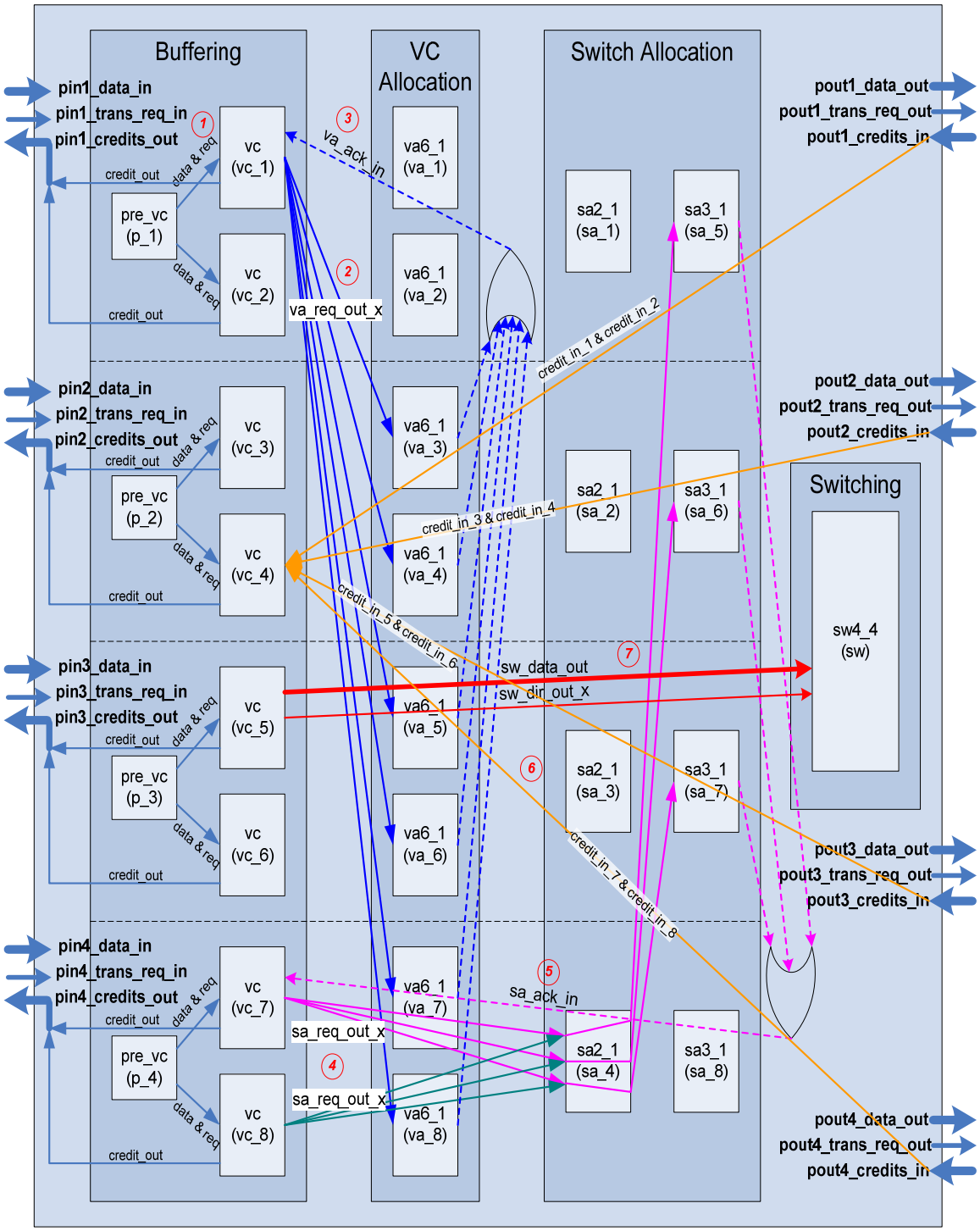


Figure 3.10: Block Diagram for Router with 4 Ports and 2 VCs

### 3.3.1 Pre_vc Blocks

Pre_vc blocks are the first units which an incoming flit crosses through. Their function is to direct the incoming flits into the correct virtual channel according to the VCID field of the flits. There is only one data input and only one request input for each physical channel. However there can be many virtual channels on these physical channels. Pre_vc block demultiplexes data and generates request inputs for virtual channels. Since it is composed of combinational logic only, a pre_vc block does not cause any delay except gate delays.

The number of pre_vc blocks in a router is equal to the number of ports. In Figure 3.11, circuit diagram for the pre_vc blocks of a router with two virtual channels is illustrated. As the number of the virtual channels increases, number of the outputs of the pre_vc blocks increases too. Logic elements also increase proportionally to multiplex the flits between more outputs.

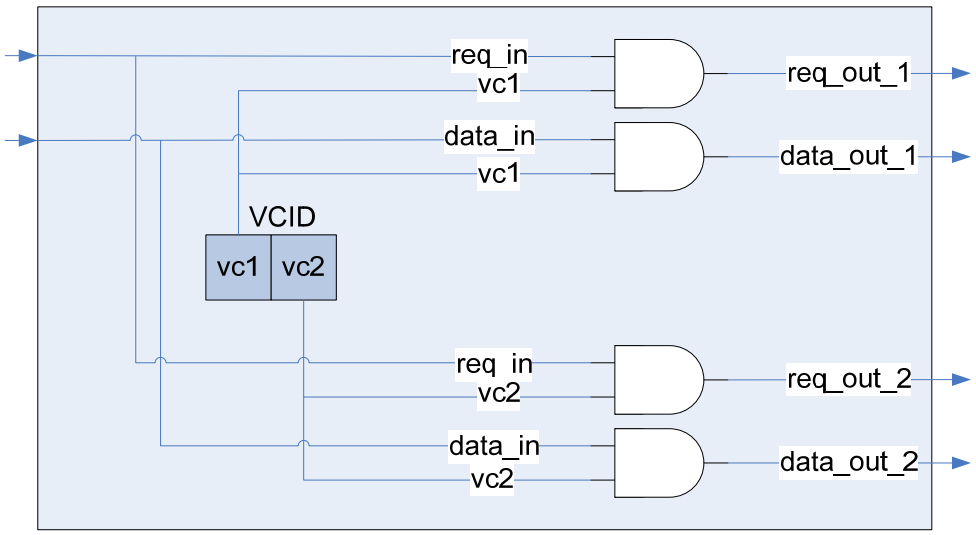


Figure 3.11: Circuit Diagram for Pre_vc Blocks of a Router with 2 VCs

### 3.3.2 Vc Blocks

Vc block performs a multiple stage functionality inside the router [7, 28, 29]. The first stage of its function is to buffer the incoming flits until they leave the router. If the stored flit is a header flit, in the next stage vc block decodes the routing field, understands the output port for that flit and its trailing flits and chooses the appropriate output virtual channel on this route. Depending on the output port and output virtual channel, vc block sends a request for virtual channel allocation and waits until the acknowledge is received. Unless the stored flit is a header flit, this stage is bypassed because virtual channel allocation is executed on a packet basis. Once acknowledgment is received for the header flit, it is valid for the rest of the packet. In the third stage, vc block sends a request for switch allocation. The buffered flit is multiplexed in the switch immediately after acknowledgment for the switch allocation is received. Since switch allocation is executed on a flit basis, every flit passes through this stage. In the implemented NoC, these stages are sequentially connected. Each stage takes at least one clock period. Under best conditions (buffer is not full, acknowledgments are received immediately) a header flit leaves the router in three clock periods while other flits leave in two. Distribution of the sub-functions in the form of stages is shown in Figure 3.12.

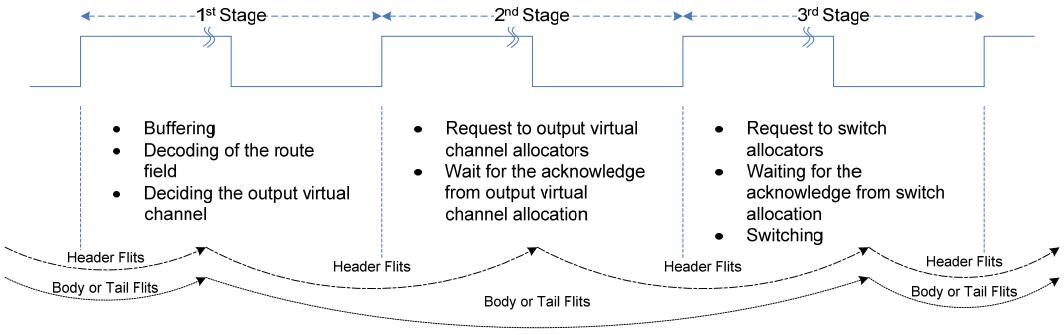


Figure 3.12: Distribution of the Sub-functions in the Form of Stages

The number of the vc blocks in a router is equal to the number of ports times the number of virtual channels. In Figure 3.13, pinout diagram for the vc blocks of a router with 4 ports and 2 virtual channels is demonstrated. Functions of the I/O ports are explained in Table 3.4 and Table 3.5.

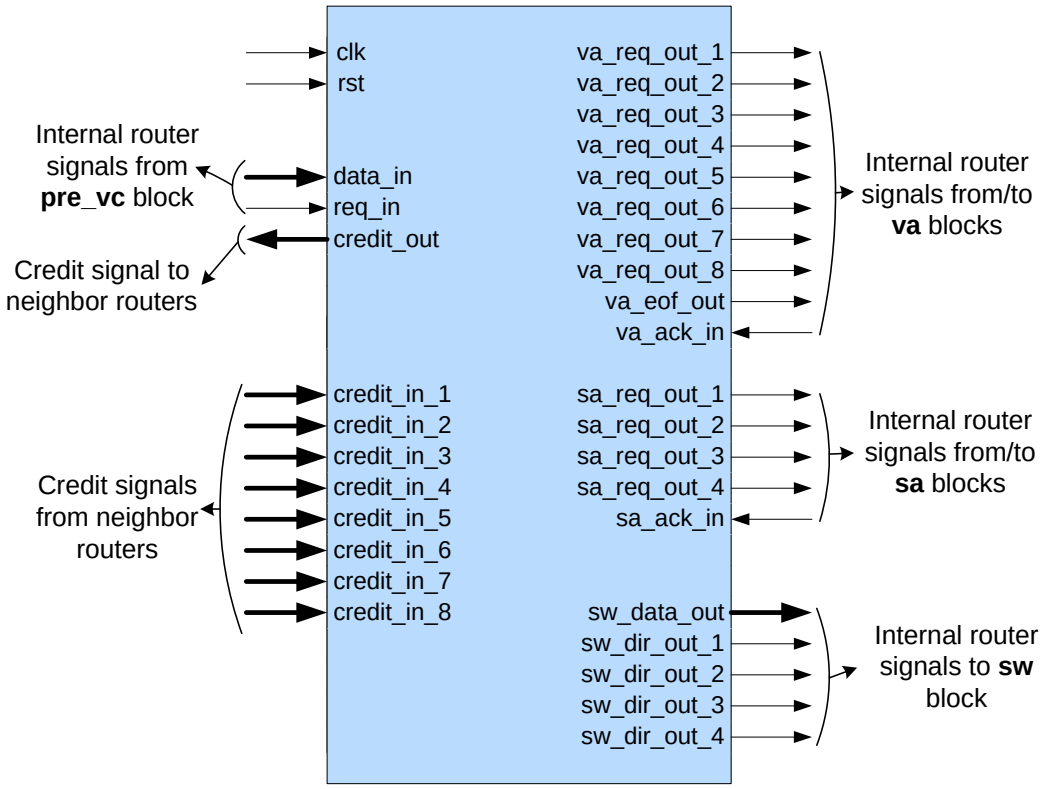


Figure 3.13: Pinout Diagram for Vc Blocks of a Router with 4 Ports and 2 VCs

Table 3.4: Functions of I/O Ports of Vc Blocks Part 1 of 2

| ***Port*** | ***Function*** |
|---|---|
| **clk** | Reference clock input for sequential processes. |
| **rst** | Asyncronous reset input. |
| **data_in** | Data input connected to one of the data outputs of the related pre_vc block. Takes flits into block. |
| **req_in** | Request input connected to one of the request outputs of the related pre_vc block. Indicates availability of valid flit in *data_in* input. |
| **credit_out** | Credit output connected to *credit_in* inputs of the vc blocks in neighbor nodes. Indicates available buffer size. Credit_out signals of vc blocks belonging to the same port are concatenated and they form *pinx_credit_out* outputs of the router. |
| **credit_in_*x* (1 - 8)** | Credit inputs connected to *credit_out* outputs of 8 vc blocks in neighbor nodes. Carries available buffer size information of vc blocks in neighbor nodes. These signals are formed from *poutx_credits_in* inputs of the router.. |
| **va_eof_out** | Eof bit output connected to the va_eof_in input of the va blocks to indicate the end of packet. |
| **va_req_out_*x* (1 - 8)** | Request outputs connected to 8 discrete virtual channel allocators. Generated according to decoded route field and decided output virtual channel. Two of these signals are left unconnected. |
| **va_ack_in** | Acknowledge input indicating whether the output virtual channel is granted or not. |
| **sa_req_out_*x* (1 - 4)** | Request outputs connected to switch arbitrators. Generated according to decoded route field. One of these signals is left unconnected. |
| **sa_ack_in** | Acknowledge input indicating the granted switch access. |

Table 3.5: Functions of I/O Ports of Vc Blocks Part 2 of 2

| ***Port*** | ***Function*** |
|---|---|
| **sw_data_out** | Data output connected to switch. Takes flits out of block. |
| **sw_dir_out_*x*<br>(1 – 4)** | Direction and request outputs connected to switch. Indicates availability of valid flit in *sw_data_out* and its direction. |

Besides input and output ports, internal signals are also employed in the operation of the vc block. These signals and their functions are given in Table 3.6. They will be used in the following algorithmic state machine diagrams that describe the operation of vc block.

Table 3.6: Functions of the Internal Signals of Vc Blocks

| ***Signal*** | ***Function*** |
|---|---|
| **in_cnt** | Counter in VC_IN process for incoming flits in modulo *buffer size.* |
| **out_cnt** | Counter in VC_OUT process for outgoing flits in modulo *buffer size.* |
| **buf_ind** | Used capacity of the buffer. Expresses zero or positive values and equals to (*in_cnt - out_cnt*) or (*buffer size + in_cnt - out_cnt)* depending on overflow of *in_cnt* and *out_cnt* counters. |
| **vc_buffer(*x*)** | $x^{th}$ cell of the circular FIFO buffer with size *buffer size*. |
| **vc_route** | Decoded routing info depending on related bits in *vc_buffer(out_cnt)* and *credit_in* inputs. |
| **vc_eof** | Eof bit of the flit in *vc_buffer(out_cnt).* |
| **va_buffer** | Latched *vc_buffer(out_cnt).* |
| **va_route** | Latched *vc_route.* |
| **sa_buffer** | Latched *va_buffer* on the instant that *va_ack_in* signal is received. |
| **sa_route** | Latched *va_route* on the instant that *va_ack_in* signal is received. |
| **sa_route_buf** | Latched *sa_route* used for restoring *sa_route* signal. |
| **next_ok** | Signal indicating the availabilty of the next router for the current flit. |

Vc block makes possible the flow of flits in pipelined mode, i.e., while a flit is being buffered into *vc_buffer*, another flit can be switched to the next node through sw block. Two processes running in parallel in vc block are designed to enable this

2-stage pipelined operation. Operation of these processes will be explained in the following algorithmic state machines.

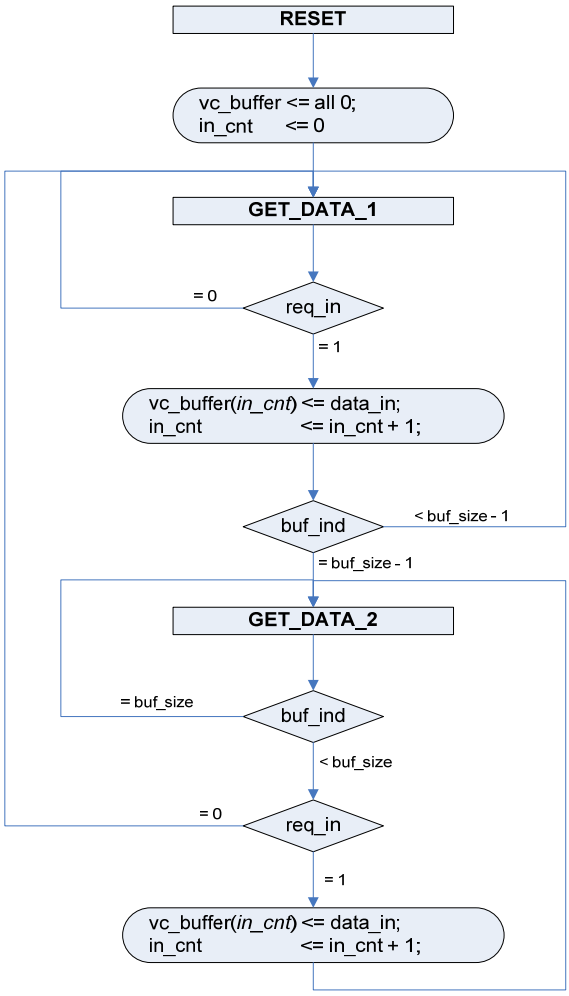


Figure 3.14: ASM of VC_IN Process

First process named VC_IN deals with storing incoming flits into buffers. In Figure 3.14 algorithmic state machine of this process is given. Each state is clarified in detail in Table 3.7.

Table 3.7: Description of States of VC_IN Process

| ***State*** | ***Description*** |
|---|---|
| **RESET** | • After asynchronous reset VC_IN process stays in RESET state for one clock period.<br>• In this state *vc_buffer* and *in_cnt* signals are reset to zero, then process goes to GET_DATA_1 state. |
| **GET_DATA_1** | • ***Buffering is performed in this state if more than one buffer space is available.***<br>• To understand whether there is a new flit or not, VC_IN process checks *req_in* signal at every rising edge of the applied clock signal.<br>• In the case that an incoming flit exists, it is stored into the next empty place indicated by *in_cnt* counter and *in_cnt* counter is incremented by 1.<br>• VC_IN process stays in this state until *buf_ind* signal indicating the used capacity of the buffer reaches *buffer size*.<br>• If *buf_ind* reaches *buffer size*, VC_IN process goes to GET_DATA_2 state. |
| **GET_DATA_2** | • ***Buffering is performed in this state if only one buffer space is available or halted if no buffer space is available.***<br>• While an outgoing flit is leaving the buffer, an incoming flit can be stored simultaneously. GET_DATA_2 state makes this possible even if a single buffer area is available.<br>• Process checks *buf_ind* signal first. Even if there is a request for buffer write, incoming flit is not stored until *buf_ind* signal indicates empty area.<br>• If there is an empty area but no request, then process returns to GET_DATA_1 state.<br>• If there is both empty area and a request, incoming flit is stored into next empty place indicated by *in_cnt* counter and *in_cnt* counter is incremented by 1. |

Second process named VC_OUT deals with reading flits out of the buffer, timing of allocation stages and directing flits to the switch. In Figure 3.15, Figure 3.16 and Figure 3.17 the algorithmic state machine of this process is illustrated. Also states of VC_OUT process are defined and explained in Table 3.8, Table 3.9, Table 3.10, Table 3.11 and Table 3.12. Different micro-operations are applied to flits in different stages. The specified tables also denote which micro-operation belongs to which stage of the Figure 3.12.

Table 3.8: Description of States of VC_OUT Process Part 1 of 5

| ***State*** | ***Description*** |
|---|---|
| **RESET** | • After asynchronous reset VC_OUT process stays in RESET state for one clock period.<br>• In this state *va_buffer, sa_buffer, va_route, sa_route, sa_route_buf, va_eof_out* and *out_cnt* signals are reset to zero, then process goes to OUT_DATA_1 state. |
| **OUT_DATA_1** | • ***Generation of output virtual channel request for header flit is handled in this state if the vc_buffer is initially empty.***<br>• To understand whether there is a new flit or not, VC_OUT process checks *buf_ind* signal at every rising edge of applied clock signal until *buf_ind* signal indicates that *vc_buffer* is not empty. (1st Stage)<br>• In the case that *vc_buffer* is not empty; the cell indicated by *out_cnt* of the *vc_buffer* is read out and latched into *va_buffer*. Also *vc_route* is latched into *va_route* and *vc_eof* is output via *va_eof_out*. Then *out_cnt* counter is incremented by 1. (1st Stage)<br>• Meanwhile a combinational logic circuit generates corresponding *va_req_out* signal depending on *va_route* signal. (1st Stage)<br>• Finally, next state is decided using *vc_eof* signal indicating whether the following flit will be a header flit or not. (1st Stage)<br>• If next flit is a header flit (*vc_eof* = 1), VC_OUT process goes to OUT_DATA_2 state, otherwise (*vc_eof* = 0) process goes to OUT_DATA_4 state. (1st Stage) |

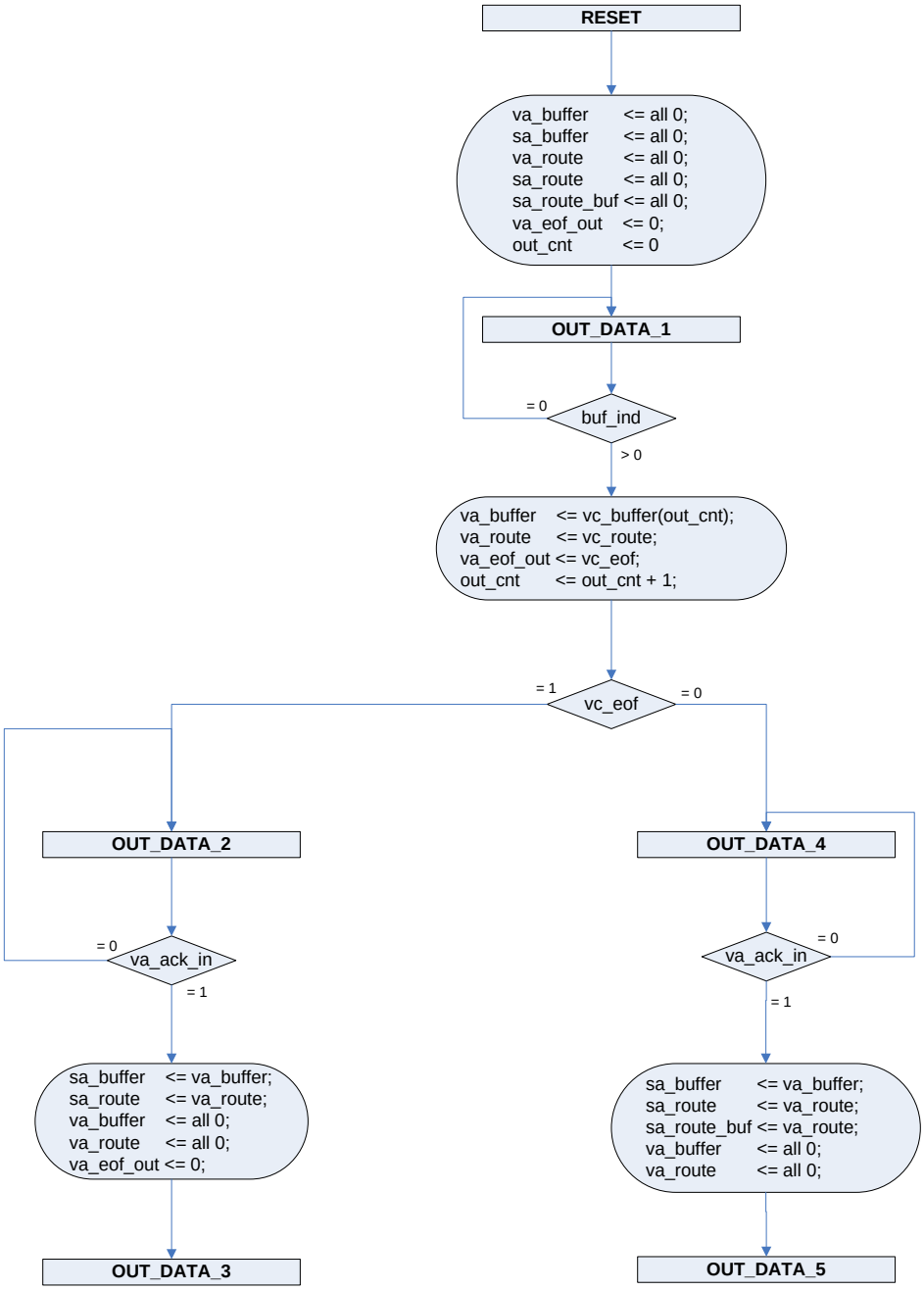


Figure 3.15: ASM of VC_OUT Process Part 1 of 3

Table 3.9: Description of States of VC_OUT Process Part 2 of 5

| ***State*** | ***Description*** |
|---|---|
| **OUT_DATA_2** | • ***Generation of switch request for header flit is handled in this state if the packet is composed of header flit only (single flit).***<br>• Process checks *va_ack_in* signal at every rising edge to understand whether the desired output virtual channel is granted or not. ($2^{nd}$ Stage)<br>• If the *va_ack_in* signal is detected, *va_buffer* signal is latched into *sa_buffer* and *va_route* is latched into *sa_route*. Furthermore *va_buffer*, *va_route* and *va_eof_out* are reset to zero. ($2^{nd}$ Stage)<br>• Meanwhile combinational logic circuit generates corresponding *sa_req_out* signal depending on *sa_route* signal. ($2^{nd}$ Stage)<br>• Then, process goes to OUT_DATA_3 state. |

Table 3.10: Description of States of VC_OUT Process Part 3 of 5

| ***State*** | ***Description*** |
|---|---|
| **OUT_DATA_3** | • ***Transmission of single flit to the sw block is handled in this state. Also generation of output virtual channel request for new incoming header flit is handled in parallel in this state.***<br>• Process checks *sa_ack_in* and *next_ok* signals at every rising edge to understand whether the switch is allocated or not and whether the next node have enough buffer space or not, respectively. (3rd Stage)<br>• If these two signals are detected, meaning flit is transmitted to next node successfully, *sa_buffer* and *sa_route* signals are reset to zero to cancel the switch allocation request. (2nd Stage)<br>• Simultaneously *buf_ind* signal is checked to understand whether *vc_buffer* is empty or not. (1st Stage)<br>• If *vc_buffer* is empty, process goes to OUT_DATA_1 stare. (1st Stage)<br>• If *vc_buffer* is not empty, *vc_buffer(out_cnt)* is latched into *va_buffer*, *vc_route* is latched into *va_route*, *vc_eof* is output via *va_eof_out* and *out_cnt* counter is incremented by 1 and *va_req_out* is generated. (1st Stage)<br>• Depending on *vc_eof* signal, process goes to OUT_DATA_2 or OUT_DATA_4 state. (1st Stage) |

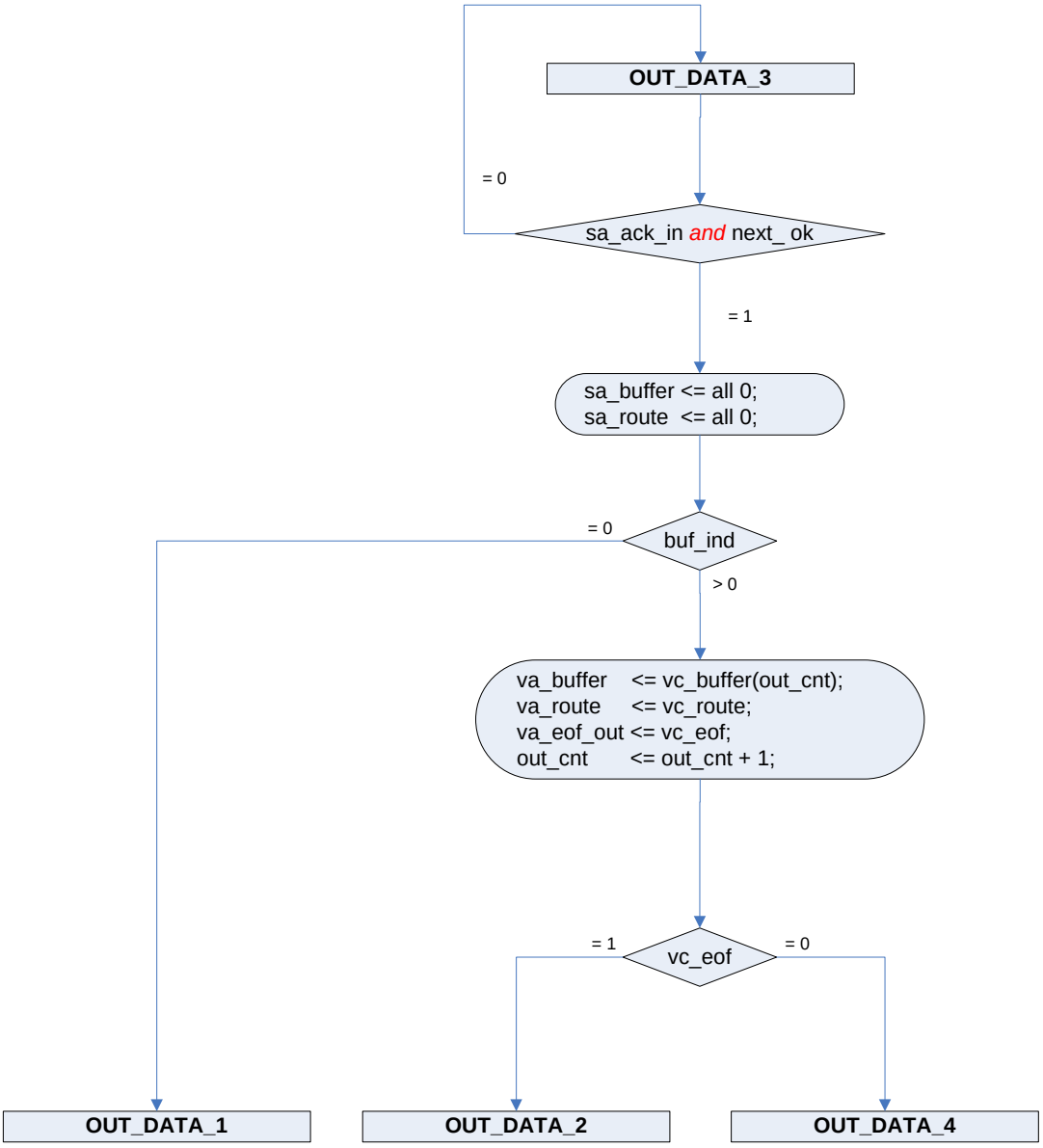


Figure 3.16: ASM of VC_OUT Process Part 2 of 3

Table 3.11: Description of States of VC_OUT Process Part 4 of 5

| ***State*** | ***Description*** |
|---|---|
| **OUT_DATA_4** | • ***Generation of switch request for header flit is handled in this state if the packet is composed of multiple flits.***<br>• Process checks *va_ack_in* signal at every rising edge. (2nd Stage)<br>• If the *va_ack_in* signal is detected, *va_buffer* signal is latched into *sa_buffer, va_route* is latched both into *sa_route* and *sa_route_buf* which will be used for restoring *sa_route* in following states, *va_buffer* and *va_route* are reset to zero. (2nd Stage)<br>• S*a_req_out* signal is generated by combinational logic depending on *sa_route* signal. (2nd Stage)<br>• Then, process goes to OUT_DATA_5 state. |
| **OUT_DATA_5** | • ***Transmission of header and body flits of packets composed of multiple flits to the sw block is handled in this state. Also generation of switch request for new incoming body flits is handled in parallel in this state.***<br>• Process checks *sa_ack_in* and *next_ok* signals at every rising edge. (3rd Stage)<br>• If these two signals are detected, process checks *buf_ind* signal. (1st Stage)<br>• If *vc_buffer* is empty, *sa_buffer* and *sa_route* signals are reset to zero to cancel switch allocation request. Process goes to OUT_DATA_6 state. (1st Stage)<br>• If *vc_buffer* is not empty, *vc_buffer(out_cnt)* is latched into *sa_buffer*, *vc_eof* is output via *va_eof_out*, *out_cnt* counter is incremented by 1 and *sa_req_out* is generated. (1st Stage)<br>• Depending on *vc_eof* signal, process stays in OUT_DATA_5 state or goes to OUT_DATA_7 state. (1st Stage) |

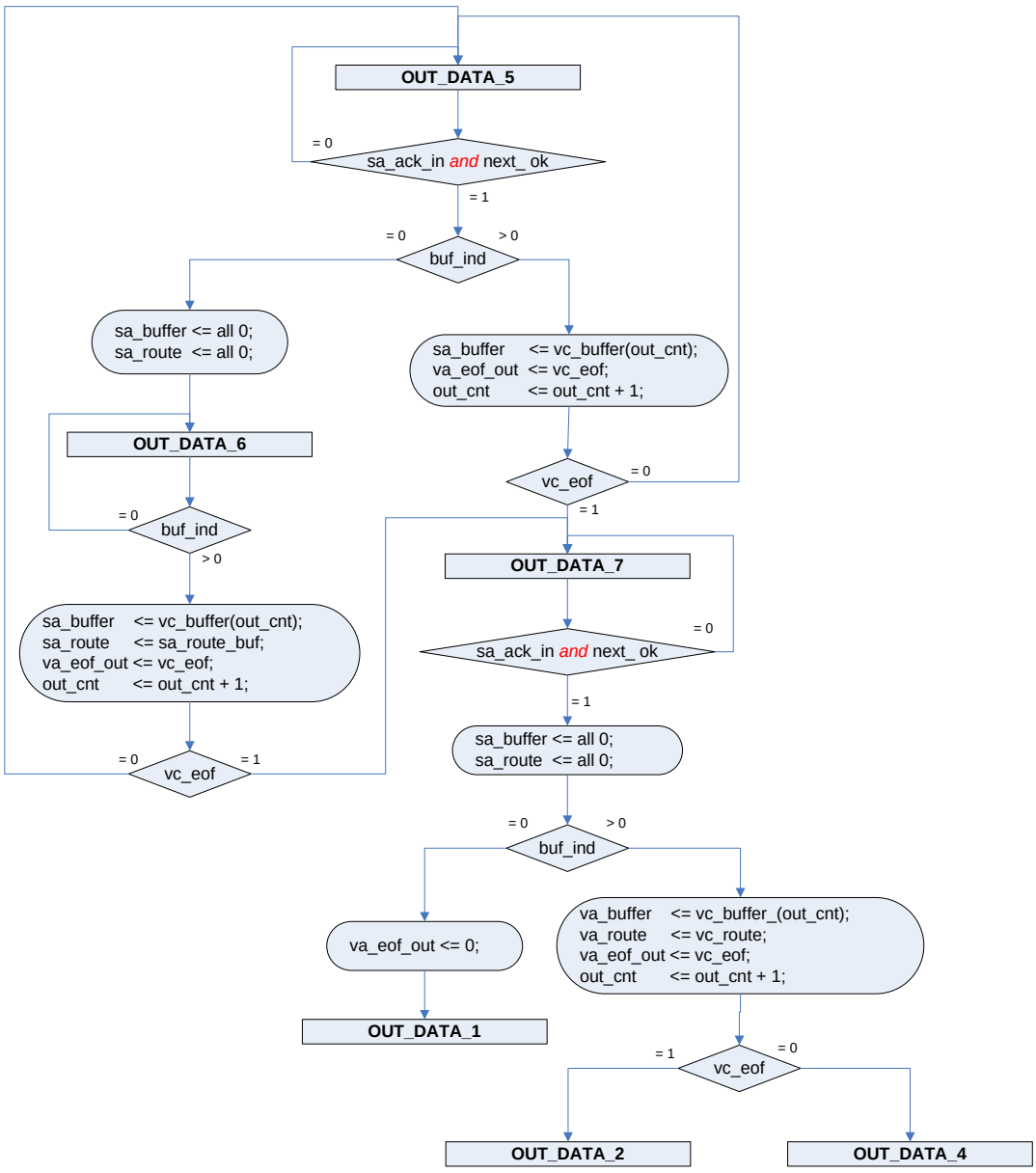


Figure 3.17: ASM of VC_OUT Process Part 3 of 3

Table 3.12: Description of States of VC_OUT Process Part 5 of 5

| ***State*** | ***Description*** |
|---|---|
| **OUT_DATA_6** | • ***Generation of switch request for new incoming body flits is handled in this state if the vc_buffer is initially empty.***<br>• Process checks *buf_ind* signal at every rising edge of applied clock. (1st Stage)<br>• In an instant that *vc_buffer* is not empty, *vc_buffer(out_cnt)* is latched into *sa_buffer*, *sa_route* is restored from *sa_route_buf*, vc_eof is output via *va_eof_out, out_cnt* counter is incremented by 1 and *sa_req_out* is generated. (1st Stage)<br>• Depending on *vc_eof* signal, process goes to OUT_DATA_5 state or goes to OUT_DATA_7 state. (1st Stage) |
| **OUT_DATA_7** | • ***Transmission of tail flits of packets composed of multiple flits to the sw block is handled in this state. Also generation of output virtual channel request for new incoming header flits is handled in parallel in this state.***<br>• Process checks *sa_ack_in* and *next_ok* signals at every rising edge. (3rd Stage)<br>• If these two signals are detected, process reset *sa_buffer* and *sa_route* signals to zero (2nd) and checks *buf_ind* signal. (1st Stage)<br>• If *vc_buffer* is empty, *va_eof_out* is reset to zero. Then, process goes to OUT_DATA_1 state. (1st Stage)<br>• If *vc_buffer* is not empty, *vc_buffer(out_cnt)* is latched into *va_buffer*, *vc_route* is latched into *va_route*, *vc_eof* is output via *va_eof_out, out_cnt* counter is incremented by 1 and *va_req_out* is generated. (1st Stage)<br>• Depending on *vc_eof* signal, process goes to OUT_DATA_2 state or goes to OUT_DATA_4 state. (1st Stage) |

State transition diagram of the VC_OUT process is given in Figure 3.18. State transitions depend on several conditions. Different conditions are checked for flits in different stages of Figure 3.12. For instance, ‘empty’, ‘tail’ and ‘not_tail’ conditions are checked for the flits in the 1st stage, ‘va’ condition is checked for the flits in the 2nd stage and ‘sa’ condition is checked for the flits in the 3rd stage.

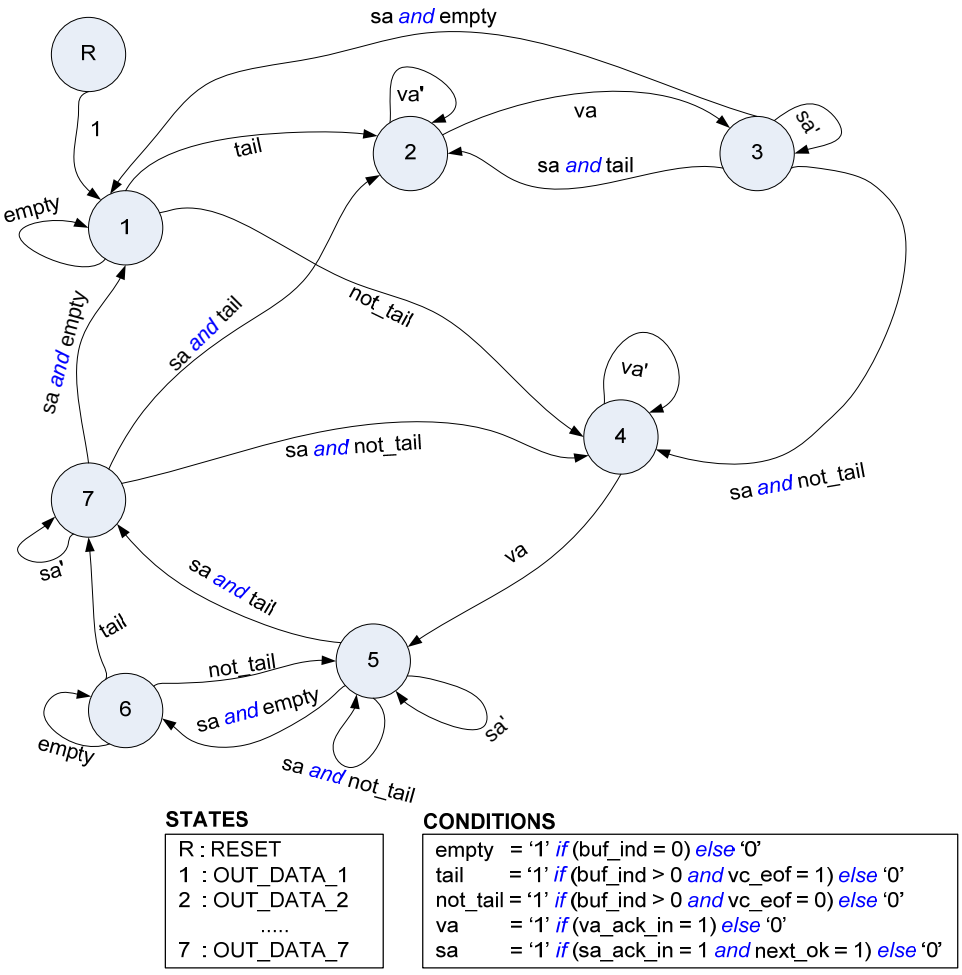


Figure 3.18: State Transition Diagram of the VC_OUT Process

### 3.3.3 Va Blocks

Va blocks implement output virtual channel allocation hardware. Depending on the arbitration method applied in va blocks an output virtual channel is allocated to one of the vc blocks requesting it. The number of va blocks in a router is equal to *number of ports* * *number of virtual channels.* Each of these blocks have (*number of ports* – 1) * *number of virtual-channels* request inputs and corresponding number of acknowledge outputs. This is because flits can not be routed to the port from which they are inserted. So vc blocks can only send a request to virtual channels of other ports. As an example, for a router with 4 ports and 2 VCs, there are 8 va blocks with 6 request and 6 acknowledge outputs. Output virtual channels of port 1 can not be requested by its own vc blocks. They can be requested by vc blocks of port 2, 3 and 4. Since each port has 2 vc blocks, arbitration is performed among 6 requests. The pinout diagram for va6_1 block, va block of the router with 4 ports and 2 VCs, is given in the Figure 3.19.

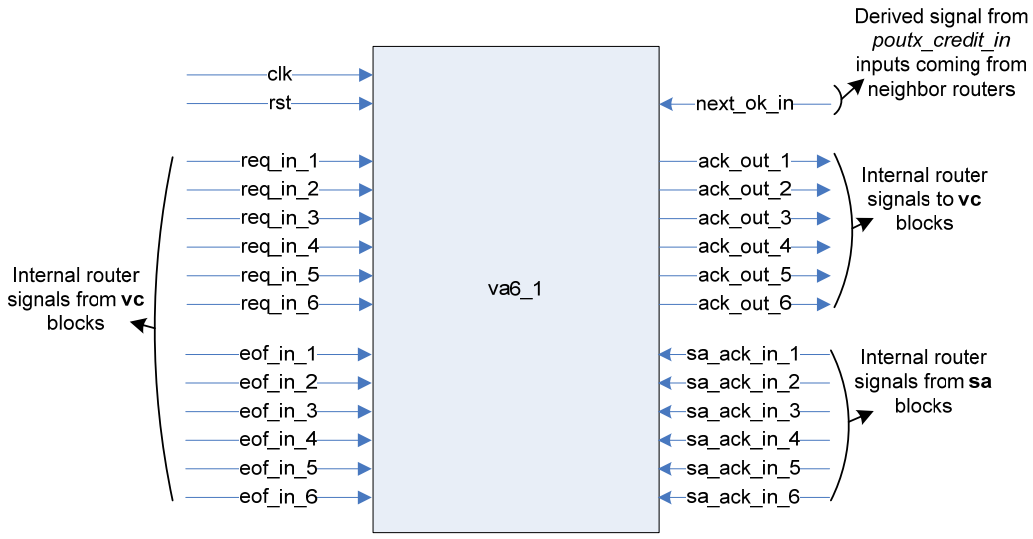


Figure 3.19: Pinout Diagram for va6_1 Block

A vc block can request only one output virtual channel at a moment. However several vc blocks can request the same output virtual channel at the same time. To solve this contention, an arbitration mechanism is executed in va blocks. This mechanism chooses one of the requests and sends acknowledge indicating that the output virtual channel is allocated to vc block to whom the winner request belongs. Round-robin type arbitration mechanism is applied in our implemented NoC. State of this mechanism is held in the VA process running independently in each va block. For the sake of simplicity, one sixth of ASM of this process is illustrated in Figure 3.20 and Figure 3.21. Only the transitions of GET_HEADER_1, GET_HEADER_ACK_1 and GET_PAYLOAD_1 states are given in these figures. Entire ASM is composed of eighteen states. Connection between these all states is expressed through Figure 3.22. Mechanism is same for other states except checking order of requests which corresponds to priority. For example, in GET_HEADER_4 state *req_in_4* is the most prior request and *req_in_3* is least prior one while in GET_HEADER_1 state *req_in_1* is the most prior request and *req_in_6* is least prior one. In the case that there exist one or more requests, the most prior one is accepted and acknowledged. According to *eof_in_1* input which is coming from vc blocks to indicate the end of packet, next state is decided. If *eof_in_1* is active, then process goes to GET_HEADER_ACK_1 state. Unlike GET_HEADER_1, these states control additional inputs *next_ok_in*, generated to indicate the free space availability in the next router, and *sa_ack_in_1,* related switch allocation acknowledgment, at first to understand whether the transmission of last flit is successful or not. Beside this difference, remaining logic is the same as GET_HEADER_1. If *eof_in_1* is not active, then process goes to GET_PAYLOAD_1 state. This state is needed to hold acknowledgment for the entire packet composed of multi flits until the tail flit of packet is transmitted successfully.

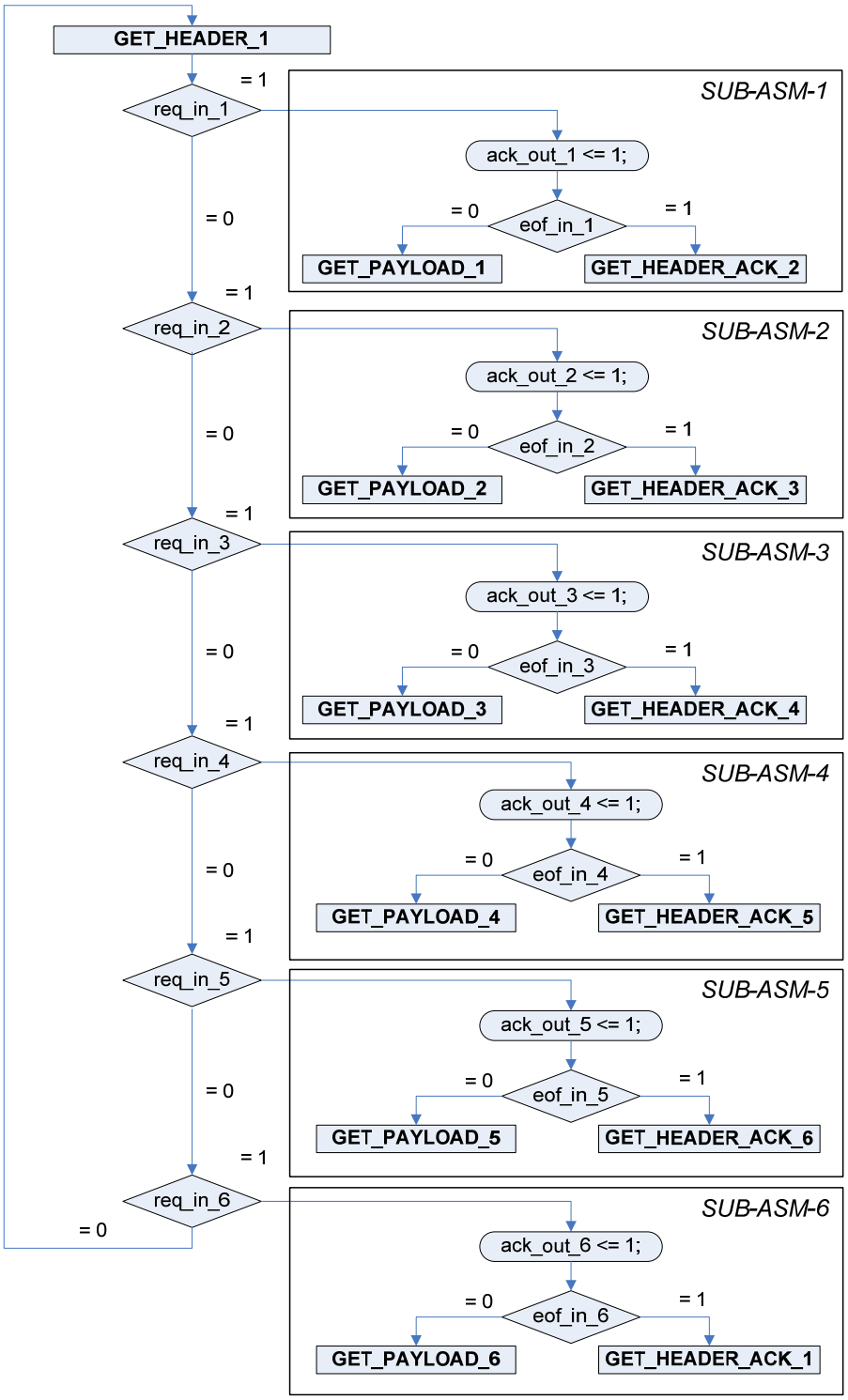


Figure 3.20: ASM of VA Process Part 1 of 3

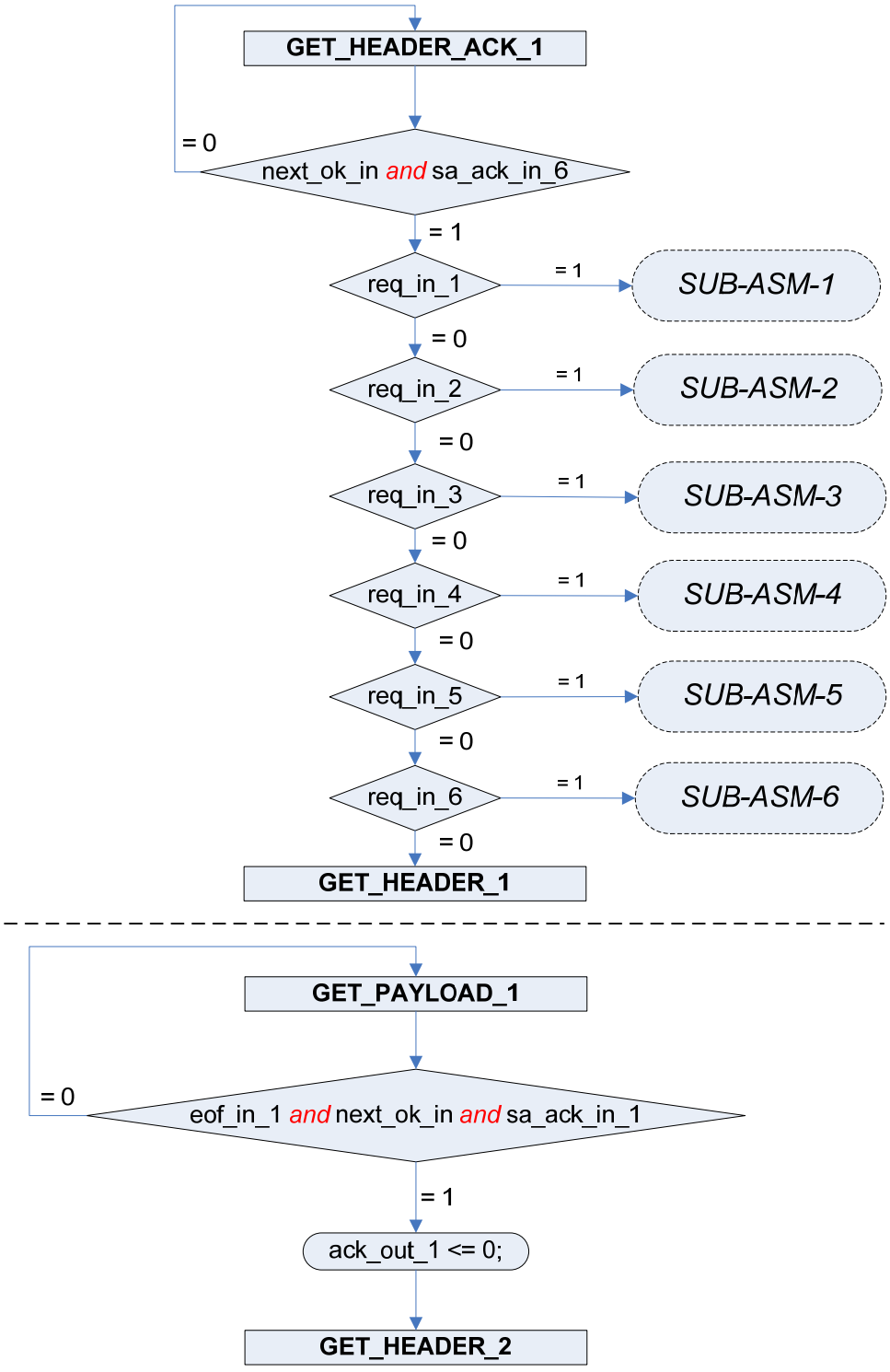


Figure 3.21: ASM of VA Process Part 2 of 3

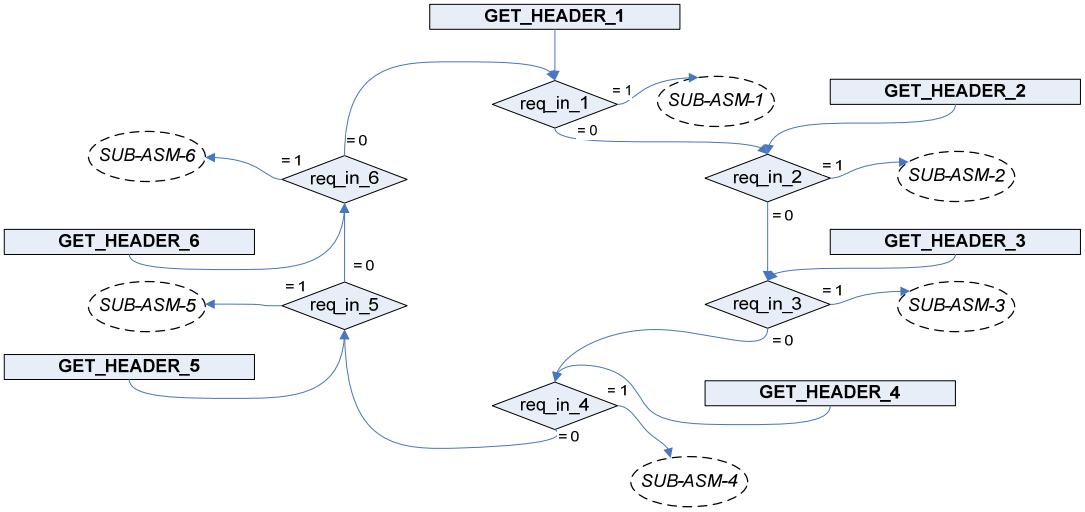


Figure 3.22: ASM of VA Process Part 3 of 3

As was denoted in previous sub-section there are 8 virtual channel request outputs in each vc block of a router with 4 ports and 2 VCs. Only 6 of these requests are connected to the va blocks. Corresponding acknowledge signals of requests passes through an OR gate and a single acknowledgment reaches the vc block as shown in Figure 3.23.

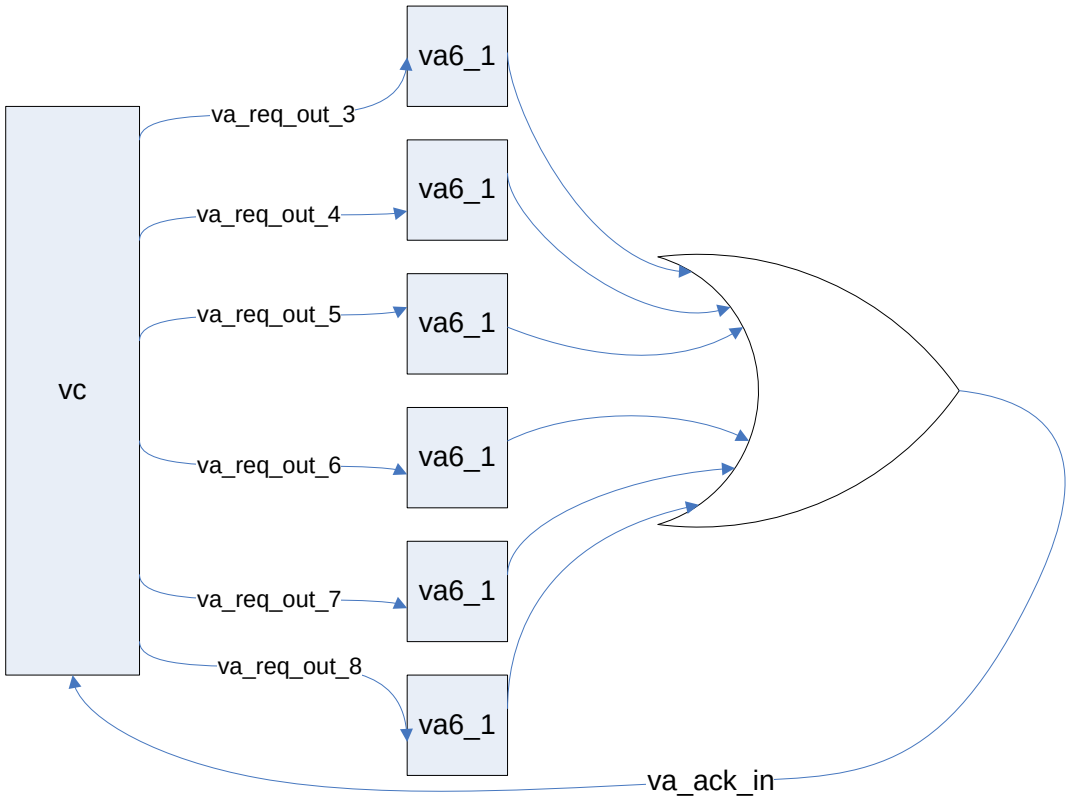


Figure 3.23: Acknowledge Signals Passing Through the OR Gate

### 3.3.4 Sa Blocks

Sa blocks implement switch allocation hardware. Desired port of the switch is allocated to one of the vc blocks requesting that port depending on the arbitration method applied inside sa blocks. Allocation is performed on flit a basis and takes place in two stages. In the first stage, only one vc block is chosen among the ones which belong to same physical port. Even if vc blocks request different ports of the switch, only one of them can be elected for the next step. Number of sa blocks operating in this stage (sa2_1) is equal to the number of ports and number of request inputs of each sa block is equal to the number of virtual channels. *Sa_req_out* outputs of each vc block passes through OR gates and resulting signals enter sa

blocks in first stage. By using outputs of these sa blocks and *sa_req_out* outputs of vc blocks, request signals for next stage are generated through some combinational logic. Second stage carries out the elimination among the elected vc blocks requesting the same output port. The number of sa blocks operating in the 2nd stage (sa3_1) is also equal to the number of ports. Since vc blocks cannot request their own port, these blocks have (*port number* – 1) request inputs and acknowledge outputs. The elected vc blocks which pass both of the stages successfully are informed by the signals generated from acknowledgments of sa blocks in both stages. Block diagram for switch allocation mechanism performed in a router with 4 ports and 2 VCs is given in Figure 3.24. For the sake of simplicity, only the requesting path of vc blocks of input port 1 for the allocation of output port 2 is shown in the figure.

Inside the sa blocks of both stages, round robin type arbitration mechanisms are taken place as in the va blocks. Unlike va blocks, sa block allocates the switch access for one flit period not for whole packet period. ASM for SA process performing in sa2_1 block is illustrated in Figure 3.25. Same flow is also applicable for sa3_1 block with difference in the number of states. Output acknowledge signals of sa blocks are generated by inside combinational logic according to the state in which the SA process is and the incoming requests.

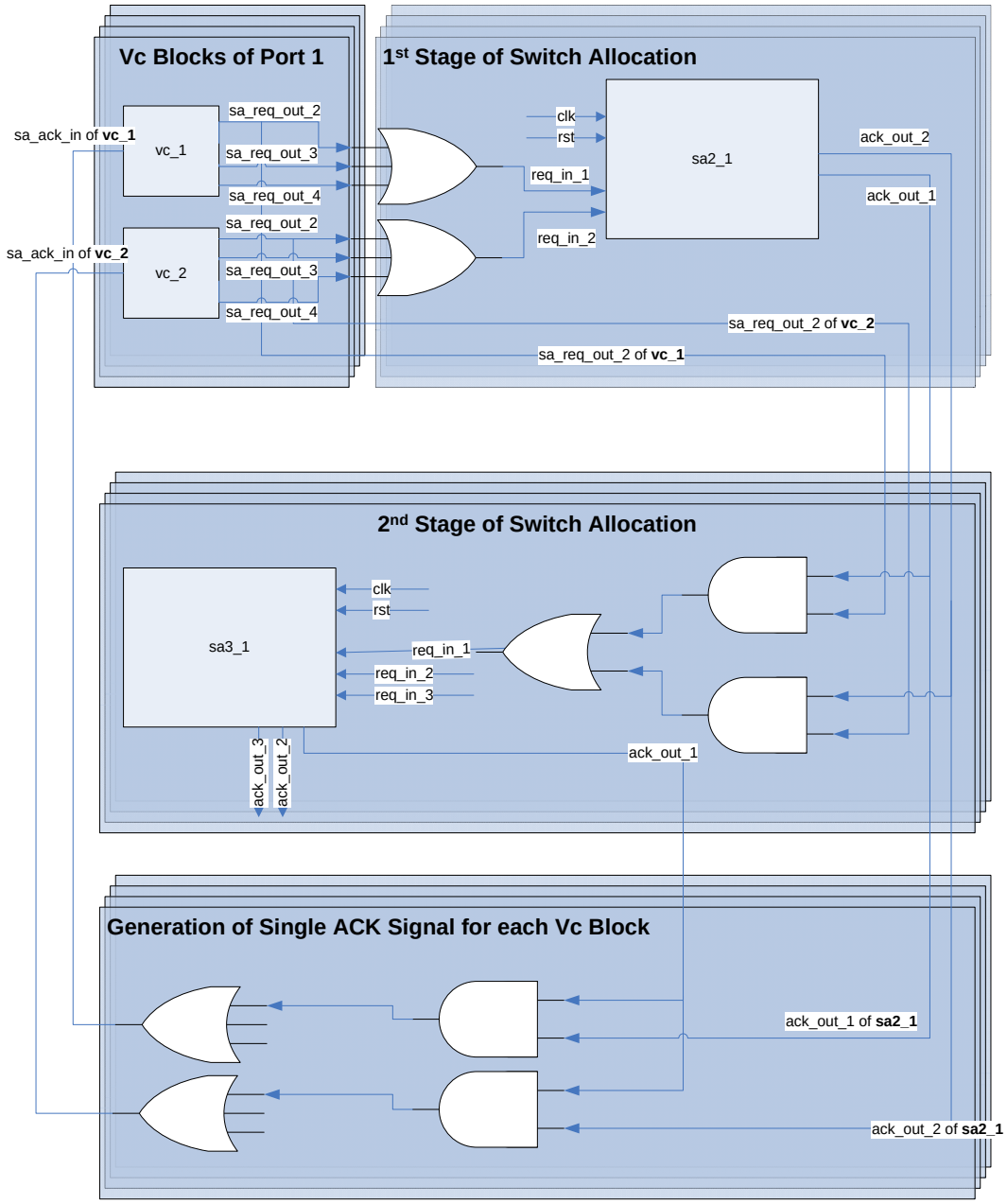


Figure 3.24: Block Diagram for Sa Block

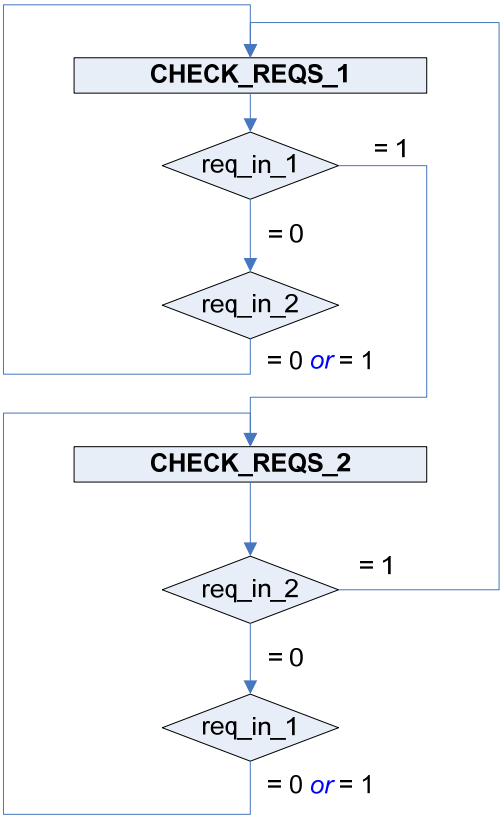


Figure 3.25: ASM of SA Process

### 3.3.5 Sw Blocks

Sw blocks switch the flits on the *data_in_x* inputs to the *data_out_x* outputs according to *req_in_x* signals and generate *trans_req_out_x* signals. *Sw_data_out* outputs of each vc block are connected *to data_in_x* inputs and *sw_dir_out_x* outputs of each vc block are connected to *req_in_x* inputs of sw block inside the router block. *Data_out_x* and *trans_req_out_x* outputs of this block are connected to the *poutx_data_out* and *poutx_trans_req_out* outputs of the router block respectively.

There exists only one sw block in each router block, which is composed of combinational logic only. Numbers of input and output ports of sw blocks are also depended on the number of ports and virtual channels as other blocks inside the router. In Figure 3.26, block diagram of the sw block inside a router with 4 ports and 2 VCs is shown. 8 data inputs connected to vc blocks are multiplexed depending on 24 bits direction information. Each vc block generates 4 bits direction information just after output virtual channel allocation and switch allocation stage. Each bit corresponds to one of the ports. One of these bits is useless due to the fact that vc blocks cannot send flits to their own port. Other 3 direction bits of each vc block are connected to select inputs of the multiplexers inside the sw block. Since allocation stages prevent any possible conflict, only one of the select inputs can be active at a time. Inside the multiplexer, one of the data inputs which is pointed by the active select input is switched to the output.

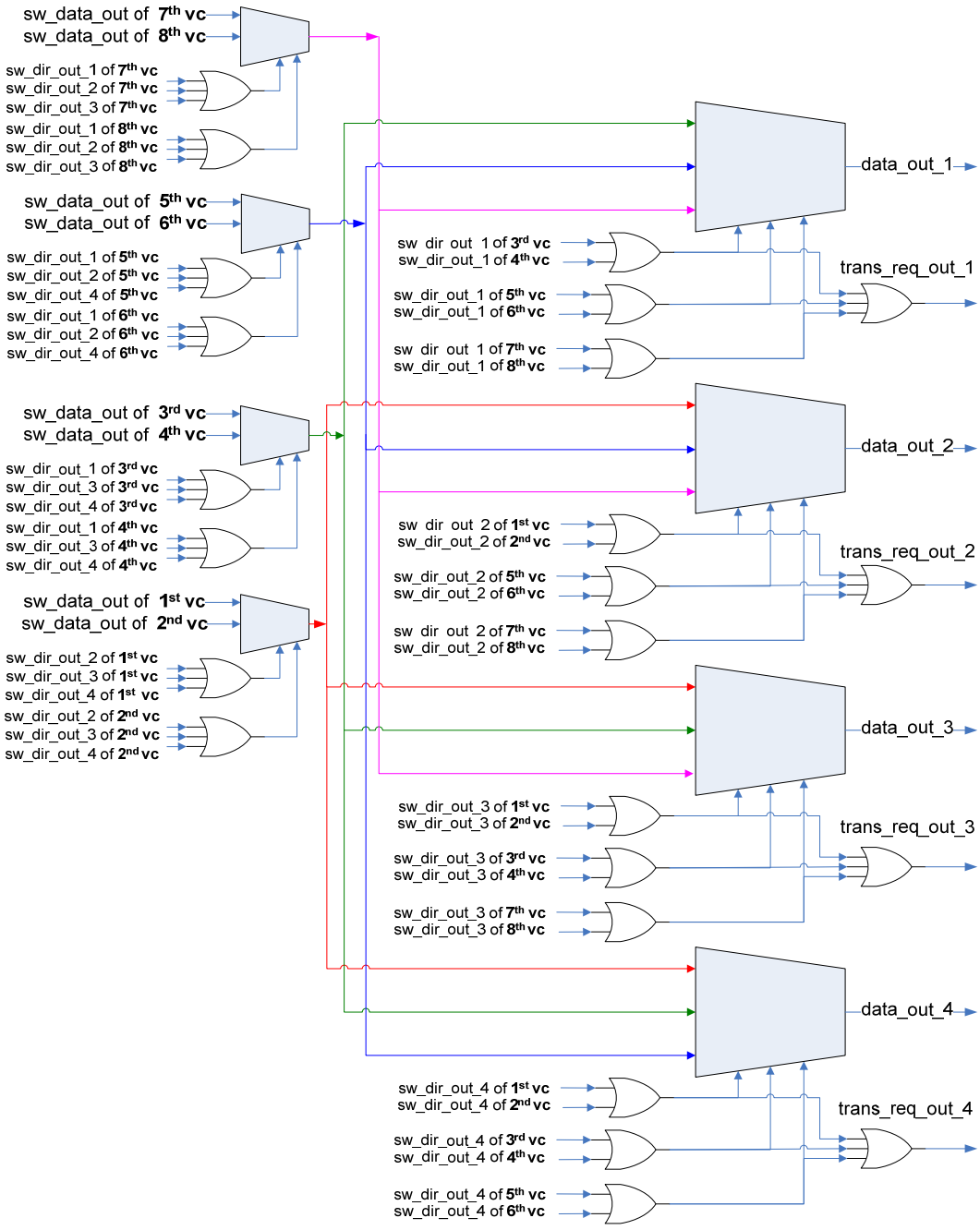


Figure 3.26: Block Diagram of Sw Block

# CHAPTER 4

## SIMULATION RESULTS AND REAL-TIME IMPLEMENTATION

The NoC design described in Chapter 3 is implemented in VHDL language. Before its real-time implementation on an FPGA platform, verifications of the VHDL source codes and also the design are performed using MODELSIM, the simulation tool produced by MENTOR GRAPHICS. After verifying the source codes, source nodes creating dummy flits are designed for real-time verification. These nodes communicate with a user interface via RS-232 to receive traffic characteristics to be be used. The designed NoC connected to these source nodes is synthesized and embedded onto our FPGA.

This chapter is composed of four sections. At first, simulation results will be presented for the NoC in spidergon topology created by 8 routers with 4 ports and 2 VCs and supporting 32-bit flits. Functions of the source node blocks and serial interface blocks will be explained in the second section. Then, user interface communicating with the source nodes will be defined and finally results of our real-time implementation obtained via this user interface will be presented.

## 4.1 SIMULATION RESULTS

For the verification of the designed NoC, several scenarios are created by using test blocks. Signals generated by the blocks of the NoC according to the applied scenarios are simulated via MODELSIM.

***Scenario 1:*** A single flit is injected to second port of the router of which buffers and other ports are empty.

Simulation result in Figure 4.1 is obtained. At the first rising edge (labelled as 1 in Figure 4.1), the flit whose value is 0xED201201 (type: bx11, size: bx101, VCID: bx10, route: bx100100, payload: bx0000001001000000001) appears at the input of the second port of the router referenced as r_1 with the request. Flit and request are demultiplexed by the pre_vc (p_2). At the second rising edge (labelled as 2 in Figure 4.1) flit is stored into the buffer in vc_3. Since there is no other flit in the buffer, flit is latched into the va_buffer and request for output virtual channel is created in the next edge. This request can be detected by va block at the fourth edge and is acknowledged immediately since no other request exists. Acknowledge can be sensed by vc block at the fifth edge and flit is latched into sa_buffer. Request for switch allocation is also created and va_buffer is reset since vc_buffer becomes empty. At the sixth edge acknowledgment for switch allocation and next_in signal indicating the availability of the next node are checked. Flit is stored into the buffers of the next node.

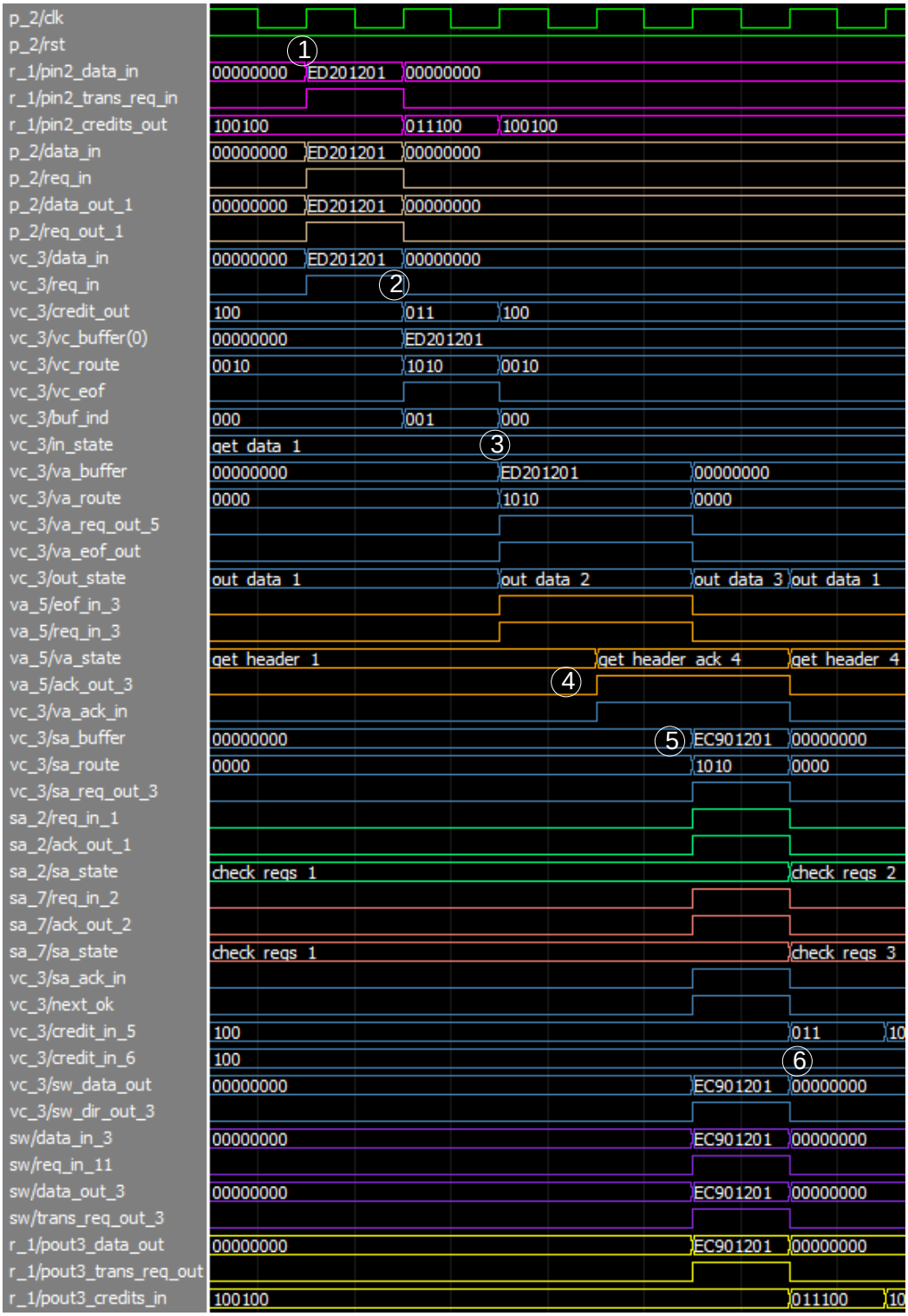


Figure 4.1: Simulation of Single Flit inside the Router

***Scenario 2:*** Consecutive flits carrying the same route information are injected through the same physical port, but via different virtual channels which is selected depending on *credit_out* signal of the router.

The obtained waveform is given in Figure 4.2. First two flits are injected from the first virtual channel and next two flits are injected from the second. First flit appears at the output port after 4 clock periods. After 2 more clock periods third flit appears at the output before the second flit since second one has to wait in *vc_buffer* until first flit is switched to output. Flits can be switched to output port consecutively as in the case of fourth and fifth flits. However, generally output port can not be used in full capacity due to the delay in virtual channel allocation stage. As the number of virtual channels using the same physical channel increases, throughput of physical channel usage is expected to increase.

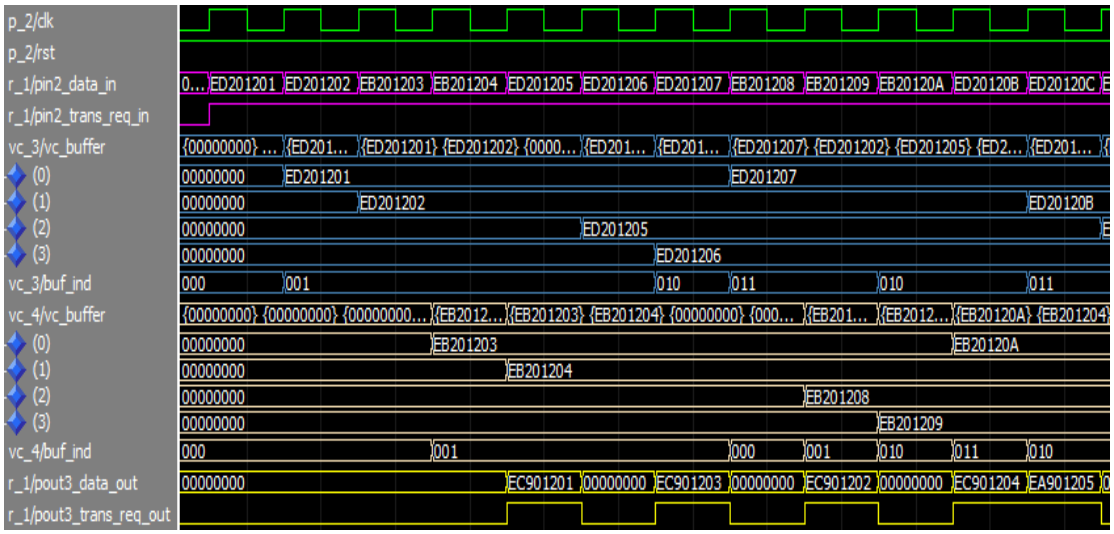


Figure 4.2: Simulation of Consecutive Single Flits Arriving to Same Port

***Scenario 3:*** Consecutive flits carrying different route information are injected through the same port with different virtual channels.

Result is illustrated in Figure 4.3. As in the previous scenario, first flit can be observed at the desired output after 4 clock periods. Moreover the flit which will be transmitted to eighth node is switched to output port before the flit which will be transmitted to fifth.

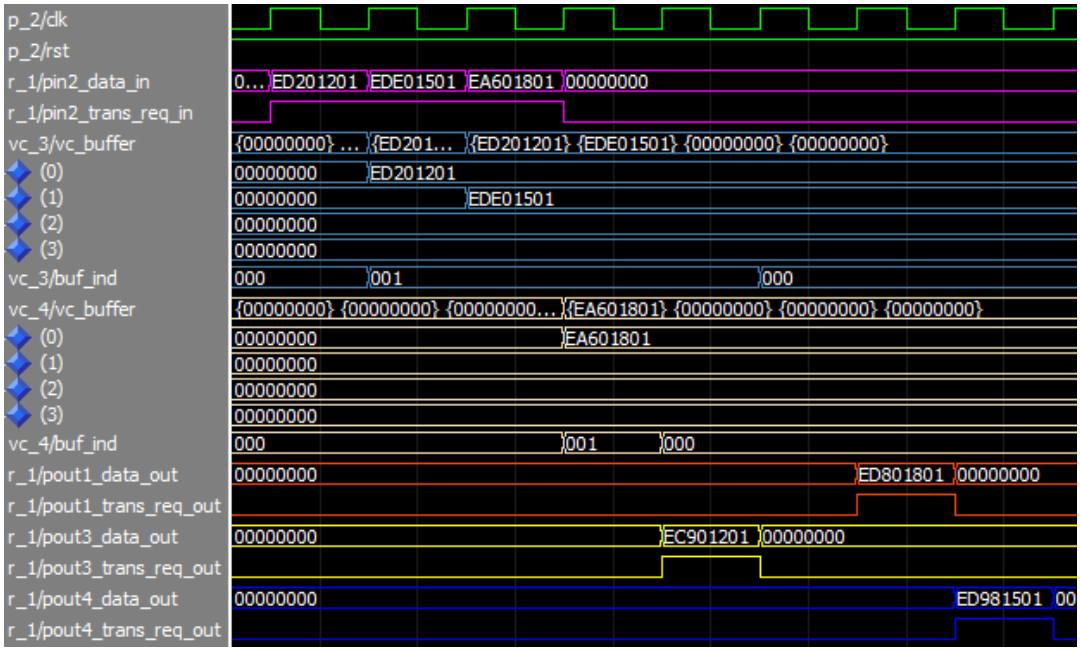


Figure 4.3: Simulation of Consecutive Single Flits Arriving to the Same Port with Different Routes

***Scenario 4:*** If the rate of injected flits is greater than the rate of ejected ones, buffers inside the block start to fill.

This case is simulated in Figure 4.4. *Buf_ind* signals increase until they reach the *buffer size.* VC_IN processes of both vc blocks (vc_3 and vc_4) go to GET_DATA_2 state. By checking *credit_out* signal, previous node understands that ejected flits cannot be stored and continues to hold the last flit in the link. On the other hand, tenth and twelfth flits can not be transmitted from r_1 and stay in the *sa_buffers* of vc blocks. Vc blocks continue requesting the switch. Flits are tried to be transmitted on a flit basis.

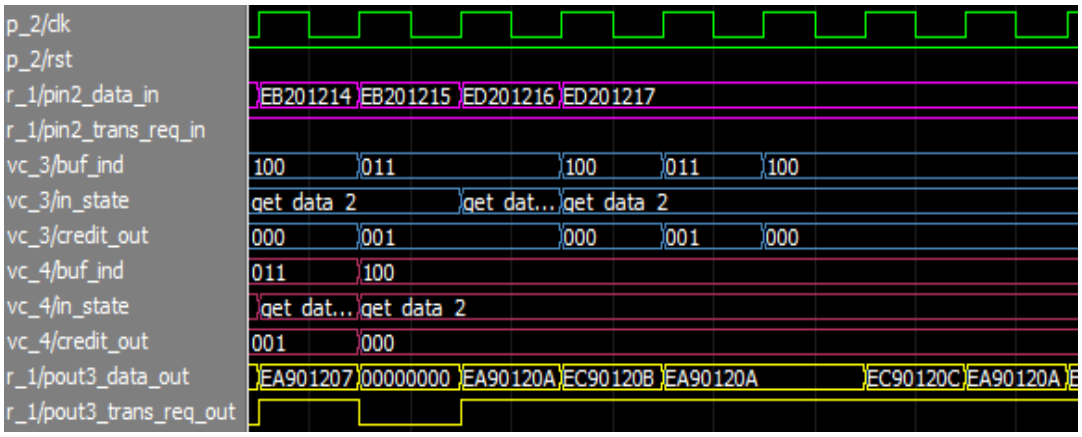


Figure 4.4: Simulation of the Case that Buffer is Full

***Scenario 5:*** Three flits requesting the same output are injected through different ports.

The obtained waveform is given in Figure 4.5. Request signals generated for the same output virtual channel reach va block (vc_3) at the same time. Only one of them can be acknowledged at any time instance. Other vc blocks continue to hold request signal. Acknowledged request by the va block generates request for the switch allocation stage. Flits are ejected in the order determined by these two allocation stages.

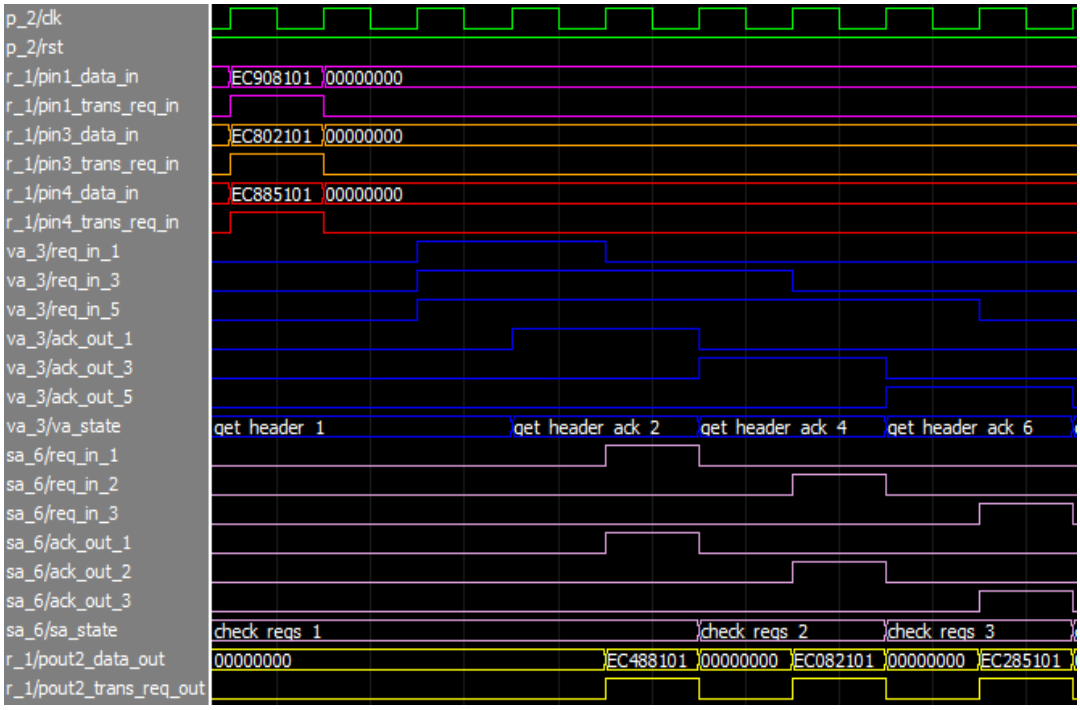


Figure 4.5: Simulation of Simultaneous Flits Requesting the Same Output

***Scenario 6:*** Simulation of a packet composed of four flits is illustrated in Figure 4.6.

Only header flit of the packet includes routing information. This information is stored in vc block and is also used for body and tail flits of the packet. Since header flit allocates the output virtual channel for the entire packet, other flits passes to switch allocation stage directly. Va block does not carry out new arbitration and holds acknowledge signal until the end of packet. Sa block carries out arbitration mechanism on flit basis. Since flits except the header are not exposed to the delay in virtual channel allocation stage, flits can be switched to output consecutively. Thus, output port can be used in full capacity even if there is only 1 virtual channel. If there exists any flit requesting the same output but different output virtual channel, flits of different virtual channels will appear on this same output port. This situation

is simulated in the Figure 4.7. While vc block is receiving a packet from port 2, another packet requesting the same output port is injected from port 1 after 5 clock periods. Different VCIDs are assigned to these packets by the vc blocks. At the output port, header flit of the second packet appears between fifth and sixth flits of the first packet. Next node distinguishes the flits through the assigned VCIDs.

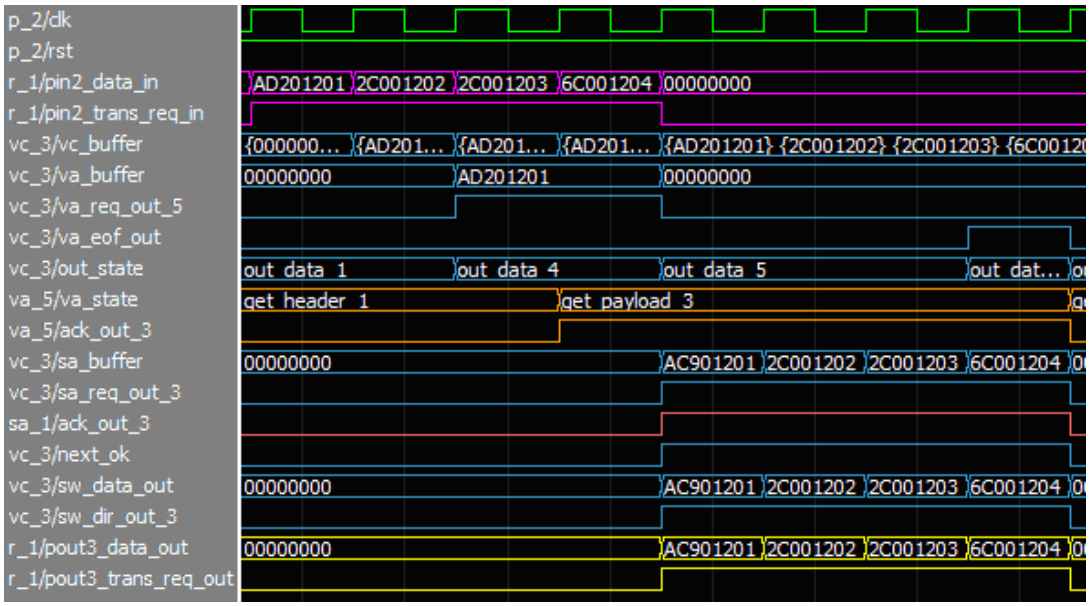


Figure 4.6: Simulation of Packet Composed of Multiple Flits

***Scenario 7:*** Flits are routed by multiple router nodes until they reach the destination.

In Figure 4.8, a packet composed of 5 flits is injected by the source node which is connected to port 2 of router 1 (r_1). Through routers r_1, r_5 and r_6, flits reach their destination node which is connected to port 4 of router 6 (r_6). Because of having 4 clock delay in each router, flits appear in the input of the destination node after 12 clocks.

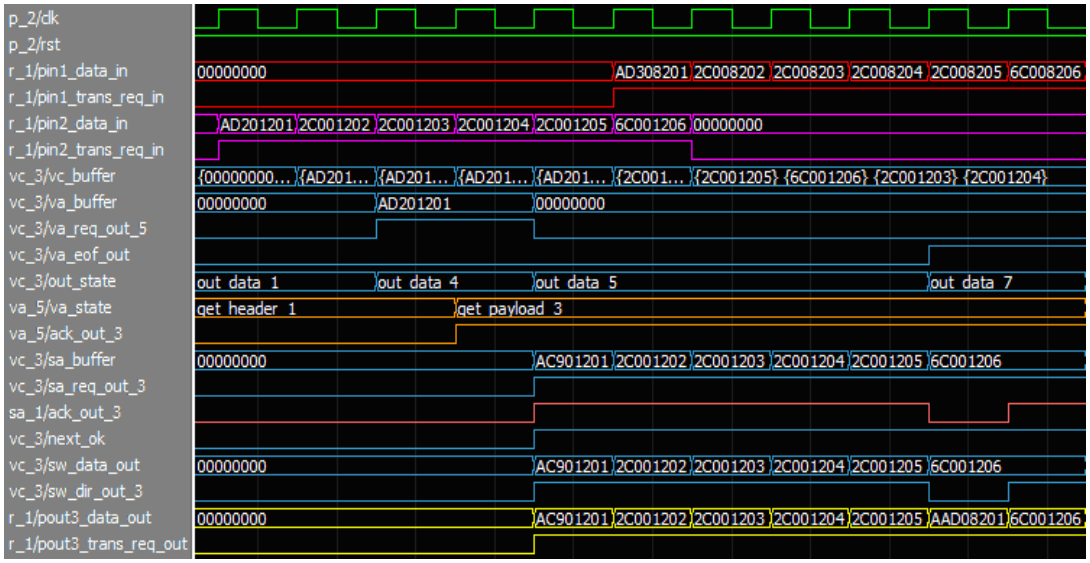


Figure 4.7: Simulation of Packets Requesting the Same Output

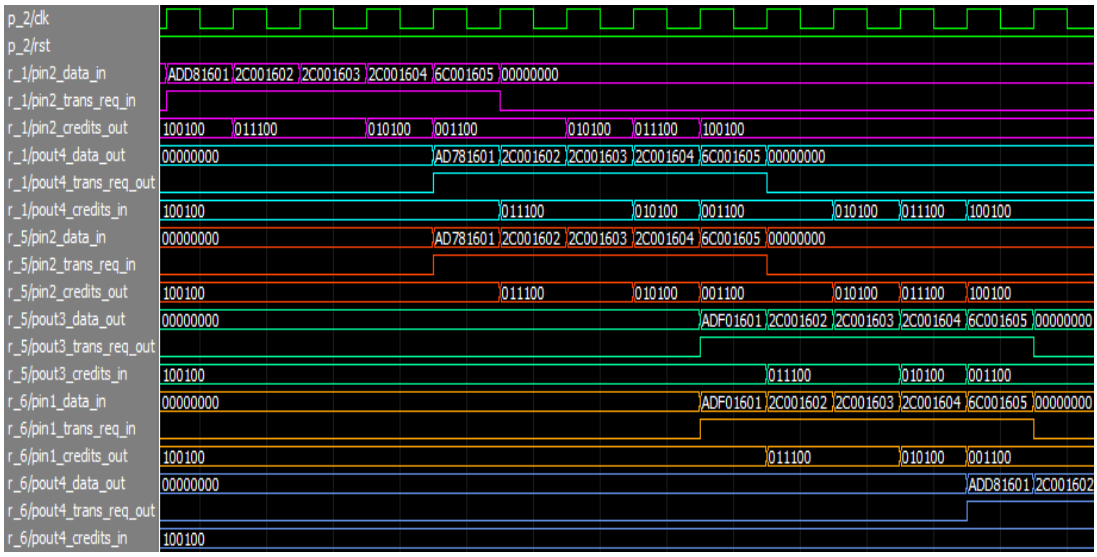


Figure 4.8: Simulation of a Packet throughout the Whole Network

## 4.2 TESTING BLOCKS

In order to test the implemented NoC in real time, an infrastructure generating dummy flits is required. Our infrastructure contains source nodes, which can transmit flits to other 7 nodes through the NoC and can receive flits from them. Our test infrastructure also communicates with a user interface executing on the computer via RS-232. Thus, outgoing traffic generated by each source node can be changed and incoming traffic received by each node can be monitored. In addition to source nodes, additional blocks are needed to maintain RS-232 communication. These blocks perform different tasks such as encoding, decoding, serializing and deserializing. Block diagram for our overall testing infrastructure is given in Figure 4.9. All blocks existing in the infrastructure will be explained briefly in the following sub-sections.

### 4.2.1 Source Nodes

Source nodes have two main functions. First function is to generate packets composed of one or more flits. Number of flits in packets is determined according to *outgoing_cnt_x* inputs. Packets are generated for the nodes whose corresponding *outgoing_cnt_x* inputs are not zero. Generation is also related with *wait_cnt* input defining the idle period of the node, in which no flit is generated. As an example if the *outgoing_cnt_1, outgoing_cnt_2* and *wait_cnt* inputs of node 1 are equal to 5, 20 and 10 respectively and other inputs are equal to zero, node 1 generates a packet composed of 5 flits to be transmitted to node 2, generates a packet composed of 20 flits to node 3 and waits for 10 clock periods without generating any flits in each cycle.

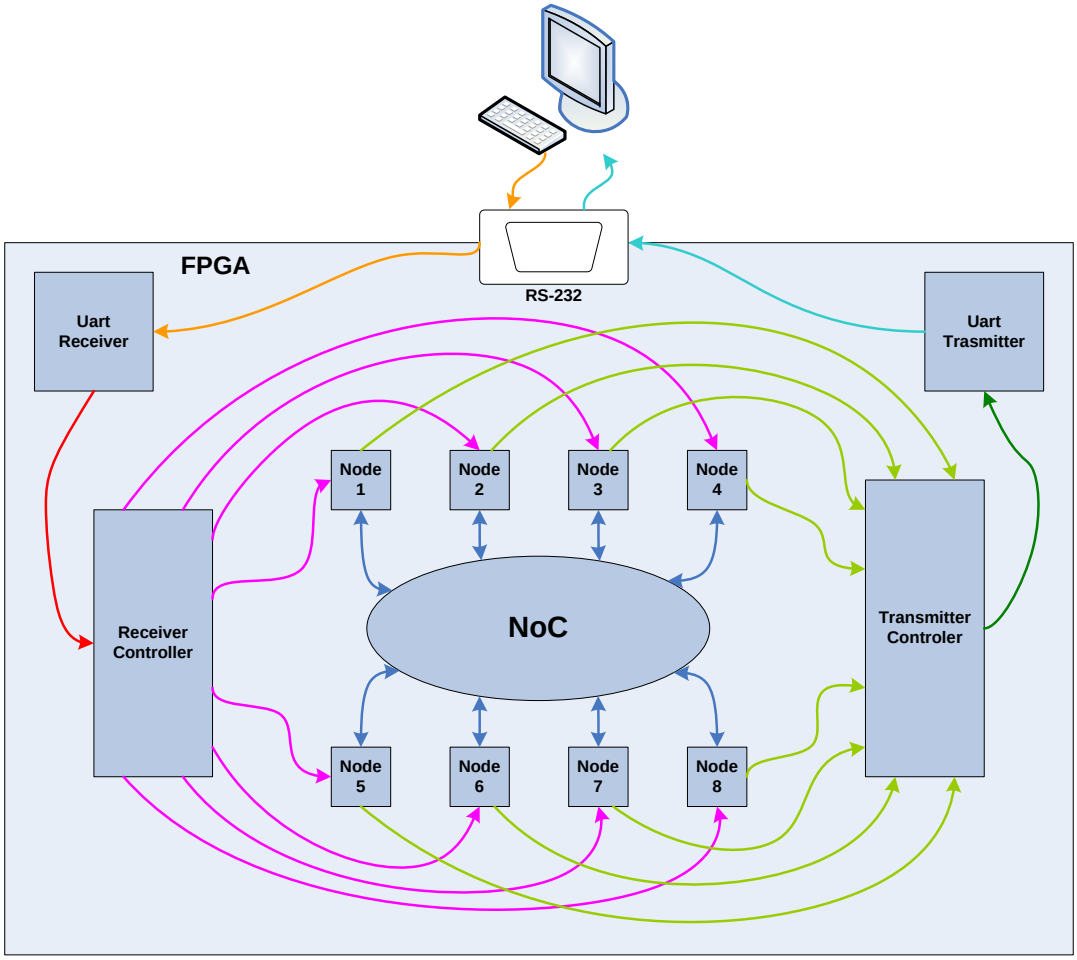


Figure 4.9: Block Diagram for the Testing Infrastructure

Route fields of generated header flits are determined using a look-up table which implements across first routing. Packets are injected from the virtual channel which is determined according to *pout_ack_in* signal carrying the credit information of the router, which the source node is connected to. Payload fields of the flits include source and destination addresses. These addresses are used in destination nodes to verify the routing applied inside the network. Payload fields also contain flit counter. Structure of the generated dummy flits is given in Figure 4.10.

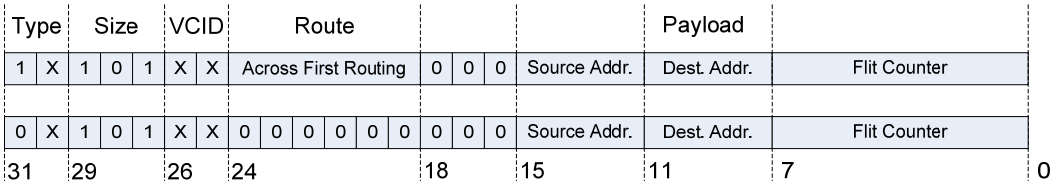


Figure 4.10: Structure of Dummy Flits

Second function of the source nodes is to monitor the incoming flits. Related *incoming_cnt_x* output is incremented according to the source address. Also destination address is checked to understand whether the flit is routed to correct destination or not. In case that destination address does not belong to that node, *error_cnt* output is incremented instead of *incoming_cnt_x* outputs.

Pinout diagram for the source nodes is given in Figure 4.11. In addition to the mentioned inputs and outputs, source nodes have a router interface to be connected to network and also *activate* input to start or stop flit generation instantly.

### 4.2.2 Uart Receiver and Transmitter

These blocks perform serializing and deserializing tasks. Receiver block deserializes the bit stream coming from the computer, outputs parallel data with *data_valid* signal indicating the availability of meaningful data. On other hand, transmitter serializes the parallel data when the *transmit* signal is triggered and also signals the state of transmission from the *transmitting* output. Pinout diagram of these blocks and waveform of the serial signals are given in Figure 4.12. 1 bit start and 1 bit stop bit are used to transmit 8 bits data and baudrate of the serial signal is 115200 bits/sec.

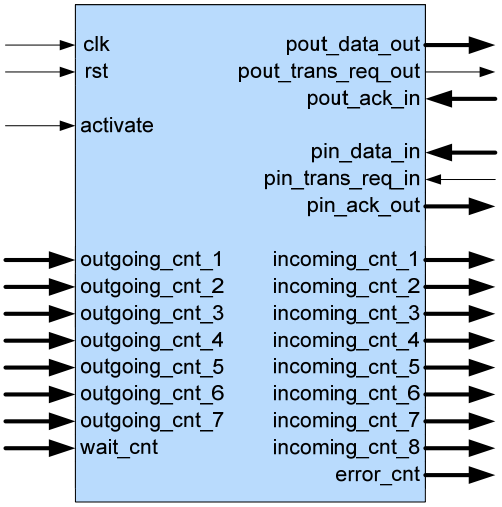


Figure 4.11: Pinout Diagram for the Source Nodes

### 4.2.3 Receiver and Transmitter Controllers

Receiver controller decodes messages coming from the user interface. *Activate*, *outgoing_cnt_x* and *wait_cnt* inputs of 8 source nodes are connected to receiver controller. Depending on the received messages, this block modifies the values assigned to these signals. Format of the received message is given in Figure 4.13. Value in data field is assigned to the signal to which value in address fields is mapped. In order to confirm the message, calculated checksum from first three bytes is compared with the received one.

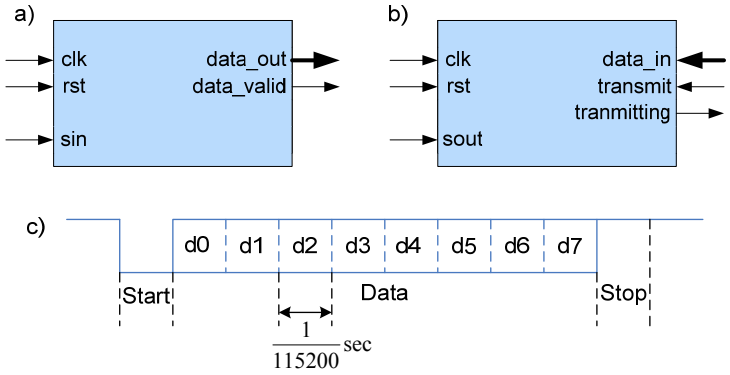


Figure 4.12: Pinout Diagram for Receiver (a) and Transmitter Blocks (b) and Waveform for the Serial Signal (c)

In order to maintain real time monitoring on the computer side, unlike receiver controller, transmitter performs its task using address and data. This block puts the information gathered from *incoming_cnt_x* and *error_cnt* outputs of the source nodes in the order recognised by the user interface and transmits bytes continuously in this order. Transmit and clock counters that are kept inside the controller are also added to the message. Values of these counters are used in rate calculations performed on the computer side. Header and checksum fields are also available in this message.

| Header | Address | Data | Checksum |
|---|---|---|---|
| 0xC5 | 0xXX | 0xYY | 0x00 – (0xC5 + 0xXX + 0xYY) |

Figure 4.13: Format of the Received Messages from User Interface

## 4.3 NOC MONITOR

NoC Monitor is a graphical user interface running on the computer to create dummy traffic for the on-chip network composed of 8 routers (4 ports, 2 VCs) in spidergon topology and to monitor created traffic on real time.

A screenshot of the NoC Monitor is given in Figure 4.14. Generated traffic is controlled by the checkboxes and the table in the division marked as 1. Each checkbox activates or deactivates generation of packets in the related source node. Each cell on the table corresponds to one of the *outgoing_cnt_x* (*TO NODEx*) or *wait_cnt* (*IDLE*) inputs of the source nodes. User interface calculates the address field according to edited cell and transmits the serial message whose format is given in Figure 4.13.

Each serial message sent by the transmitter controller block on FPGA includes 1 byte packet counter. This counter and the difference between packet counters of two consecutive messages fetched by UI are displayed on the table in the 2$^{nd}$ division of Figure 4.14. Time difference between two recent messages is also presented on this table in terms of seconds.

Table in the 3$^{rd}$ division indicates the total number of the received flits by each source node. Values of *incoming_cnt_x* and *error_cnt* outputs of source nodes are printed on the table according to source-destination pairs. Rows correspond to sources and columns to destinations. The value of a clock counter is also received from the transmitter controller on FPGA. This value is used to calculate rates of the received flits. Differences between previous and current values of each cell are divided by the difference between previous and current value of clock counter. To find the rate in terms of Mbit/sec result divisions are multiplied with 0,000032 since flit size of the tested NoC is 32. Total received and transmitted traffic by each node

can also be observed from the table. Intersection of these column and rows gives the rate of the whole traffic on NoC.

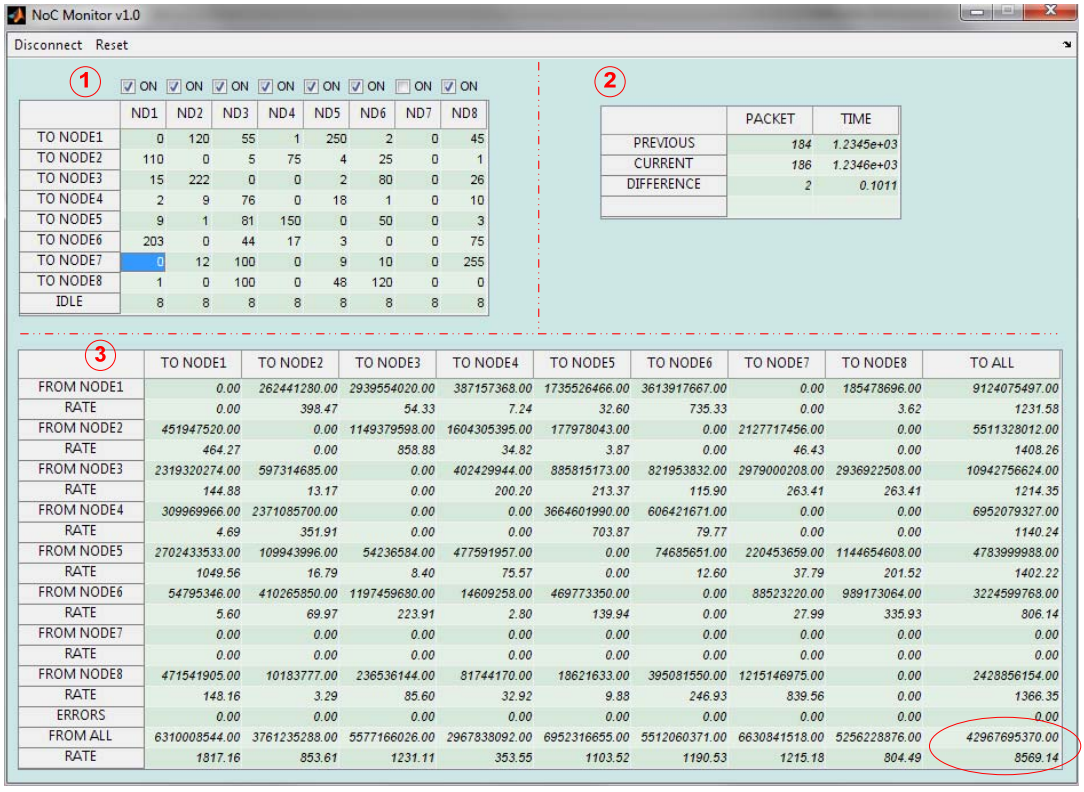


Figure 4.14 : NoC Monitor

## 4.4 NOC GENERATOR

NoC Generator is another graphical user interface designed within the scope of this thesis to generate diverse NoCs with different characteristics. This tool creates necessary VHDL source codes for the specified NoC via a user interface.

In Figure 4.15, a screenshot of the NoC Generator is illustrated. First, user chooses the topology of the network among 4 choices which are ring, spidergon, mesh and

torus. Depending on the chosen topology network size is specified as one dimensional (ring, spidergon) or two dimensional (mesh, torus). Node degree is displayed by the user interface according to the topology.

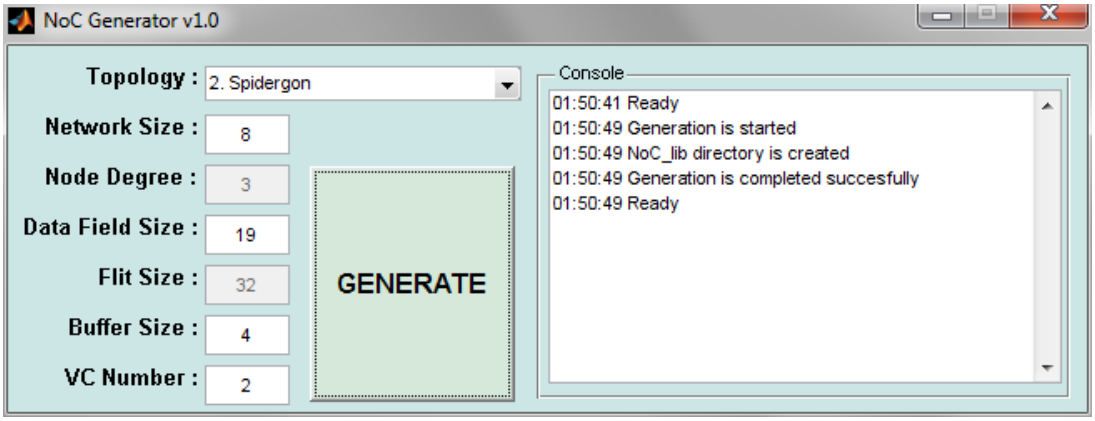


Figure 4.15 : NoC Generator

Then, data field size in the header flit is assigned. Together with topology, network size and VC number, data flit size determines the whole flit size which is calculated and displayed by our UI. The depth of the input buffers in vc blocks is specified by Buffer Size tab. Finally virtual channel number is entered and Generate button is pressed to start the generation. 'NoC_lib' directory is created and VHDL source codes which are ready to be synthesized are generated under this directory.

## 4.5 IMPLEMENTATION ON FPGA PLATFORM

The designed NoC composed of 8 routers (4 ports, 2VCs) in spidergon topology is also executed together with our testing blocks on Xilinx Virtex 6 Evaluation Board which is illustrated in Figure 4.16. This board includes XC6VLX240T FPGA of Xilinx which contains 241.152 logic cells and 37.680 slices each of which

composes of 4 look-up-tables (LUTs) and 8 flip-flops. The design is synthesized and implemented (translate, map, place & route) by using Integrated Software Environment (ISE) tool of Xilinx. At the end of these processes, programming file needed to configure FPGA is obtained and design issues such as maximum delay, maximum applicable clock frequency, I/O pad assignments and resource usage etc. are reported. According to these reports, maximum delay observed on the paths is 5,675 ns. So maximum clock frequency that can be applied to the blocks in the design is 176,214MHz. Our design uses 14.626 of 301.440 flip-flops (registers) and 44.335 of 150.720 LUTs. This information is stated on Table 4.1.

Several NoCs with different parameters are also generated using our NoC Generator. These NoCs are synthesized by using ISE. Similar information for all these NoCs is stated on Table 4.1. Table 4.1 shows that network size, flit size and buffer depth do not change maximum delay significantly. But these parameters dramatically affect resource usage. Therefore a network can be expanded without changing its timing performance. On the other hand, since number of virtual channels also has an influence on control logic, it affects both resource usage and timing performance. Topology type also changes both resource usage and timing performance.

For comparison, HERMES [25-27] NoC in mesh (4x2) topology with 2 VCs, 32 bit flit size and buffers of depth 4 is synthesized by using ISE and synthesis results are also stated on Table 4.1. When we compare HERMES NoC and our NoC with the same parameters, it is observed that our proposed NoC outperforms HERMES in timing performance in return for an increase in resource usage.

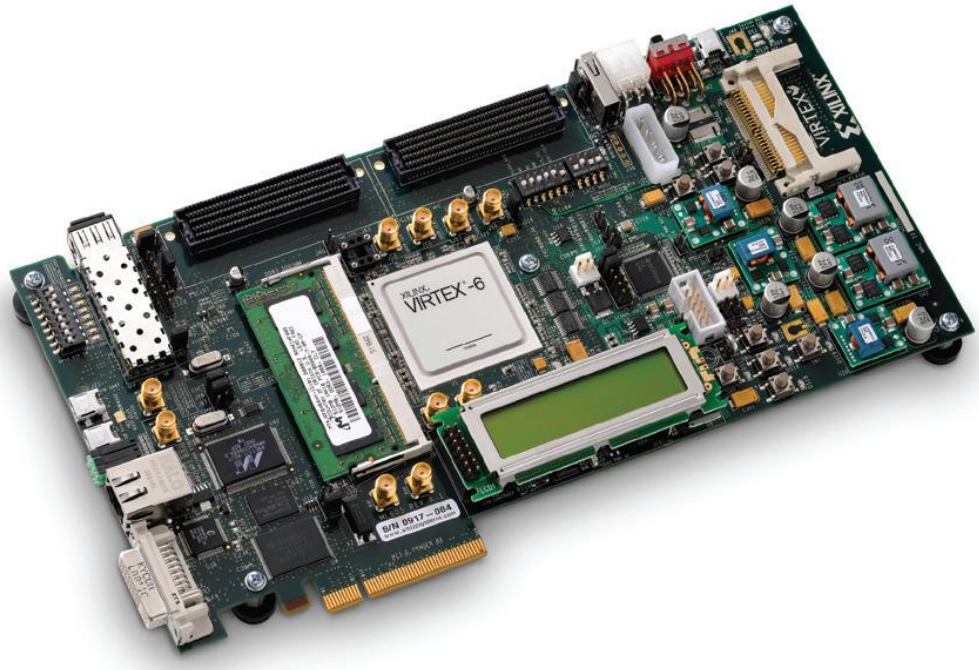


Figure 4.16: Xilinx Virtex 6 Evaluation Board

Table 4.1: Generated NoCs vs. Maximum Delay and Resource Usages

| **Generated NoCs vs. Maximum Delay and Resource Usages** | ***Maximum Delay*** | ***Registers*** | ***LUTs*** |
|---|---|---|---|
| **Spidergon (8), VC = 2, F = 32, B = 4** | 5,675 ns | 14.626 | 44.335 |
| **Spidergon (8), VC = 2, F = 32, B = 2** | 5,528 ns | 10.663 | 33.205 |
| **Spidergon (8), VC = 2, F = 64, B = 4** | 5,519 ns | 26.914 | 50.065 |
| **Spidergon (8), VC = 4, F = 32, B = 4** | 7,076 ns | 31.778 | 115.137 |
| **Spidergon (16), VC = 2, F = 32, B = 4** | 5,586 ns | 29.250 | 65.073 |
| **Ring (8), VC = 2, F = 32, B = 4** | 4,473 ns | 10.676 | 31.315 |
| **Mesh (4x2), VC = 2, F = 32, B = 4** | 6,162 ns | 14.277 | 32.710 |
| **Torus (3x3), VC = 2, F = 32, B = 4** | 6,450 ns | 21.788 | 57.646 |
| **HERMES Mesh (4x2), VC = 2, F = 32, B = 4** | 8,319 ns | 3.370 | 11.034 |

## 4.6 PERFORMANCE

When operated with 66 MHz clock frequency, a total rate of over 14 Gb/s can be obtained on the network with a 32-bit flit size as shown in Figure 4.17. Applied network traffic can also be seen on this figure. At this frequency, total capacity of 8 point-to-point links of 32 bit width is 16,896 Gb/s. In our implementation, more than 82% of this capacity can be reached. If a bus structure was implemented with the same flit size and was operated with same clock frequency, maximum 2,112

Gb/s rate would be obtained, even if arbitration periods are assumed as zero. Total rate obtained on the network can be increased by operating the network under higher clock frequencies. Maximum delay obtained by synthesis tools (as shown in Table 4.1) limits the maximum operating frequency.

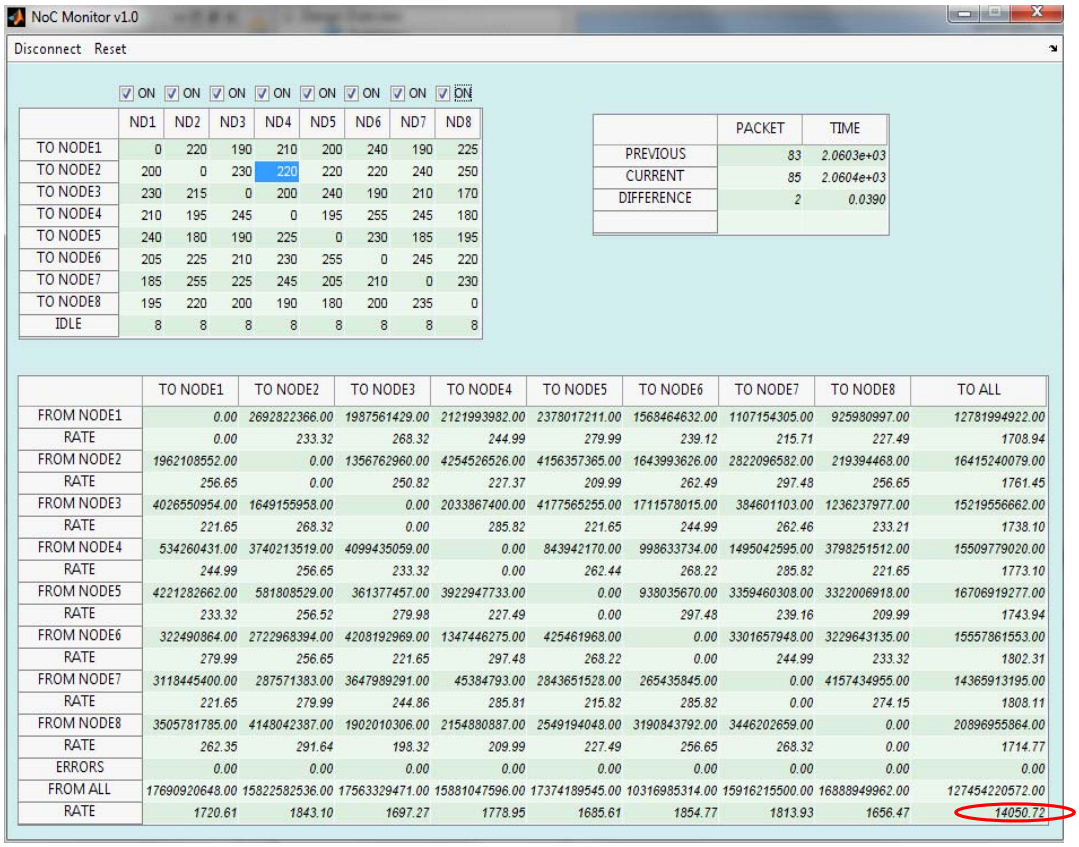


Figure 4.17: Performance of the NoC Operated with 66 MHz Clock Frequency

Also, we observe that, through our allocation mechanisms, fairness is provided. In the case that all source nodes generate identical traffic, observed incoming and outgoing traffic rates for each source node are observed to be identical. This case is

created and monitored by using our NoC Monitor as shown in Figure 4.18 which demonstrates that our NoC does not cause any privilege to any node in the network.

NoC Monitor v1.0

Disconnect Reset

ON ON ON ON ON ON ON ON

| | ND1 | ND2 | ND3 | ND4 | ND5 | ND6 | ND7 | ND8 |
|---|---|---|---|---|---|---|---|---|
| TO NODE1 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| TO NODE2 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 |
| TO NODE3 | 10 | 10 | 0 | 10 | 10 | 10 | 10 | 10 |
| TO NODE4 | 10 | 10 | 10 | 0 | 10 | 10 | 10 | 10 |
| TO NODE5 | 10 | 10 | 10 | 10 | 0 | 10 | 10 | 10 |
| TO NODE6 | 10 | 10 | 10 | 10 | 10 | 0 | 10 | 10 |
| TO NODE7 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 10 |
| TO NODE8 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 |
| IDLE | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

| | PACKET | TIME |
|---|---|---|
| PREVIOUS | 139 | 202.3426 |
| CURRENT | 141 | 202.3914 |
| DIFFERENCE | 2 | 0.0488 |

| | TO NODE1 | TO NODE2 | TO NODE3 | TO NODE4 | TO NODE5 | TO NODE6 | TO NODE7 | TO NODE8 | TO ALL |
|---|---|---|---|---|---|---|---|---|---|
| FROM NODE1 | 0.00 | 4163965752.00 | 4141767460.00 | 4128780982.00 | 4118711254.00 | 4109244096.00 | 4102289861.00 | 4145143958.00 | 28909903363.00 |
| RATE | 0.00 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 1319.99 |
| FROM NODE2 | 4087178985.00 | 0.00 | 4058101187.00 | 4044241947.00 | 4044390732.00 | 4039444306.00 | 4035925890.00 | 4030831122.00 | 28340114169.00 |
| RATE | 188.57 | 0.00 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 1320.00 |
| FROM NODE3 | 3844496532.00 | 3819413892.00 | 0.00 | 3800274222.00 | 3791098116.00 | 3785889732.00 | 3781501102.00 | 3775561032.00 | 26598234628.00 |
| RATE | 188.57 | 188.57 | 0.00 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 1319.99 |
| FROM NODE4 | 3786505502.00 | 3768759072.00 | 3758237552.00 | 0.00 | 3744206662.00 | 3738300382.00 | 3733777932.00 | 3730403818.00 | 26260190920.00 |
| RATE | 188.57 | 188.57 | 188.57 | 0.00 | 188.57 | 188.57 | 188.57 | 188.58 | 1319.99 |
| FROM NODE5 | 3735182972.00 | 3721251592.00 | 3711490462.00 | 3698134052.00 | 0.00 | 3684373952.00 | 3679474402.00 | 3674140782.00 | 25904048214.00 |
| RATE | 188.57 | 188.58 | 188.57 | 188.57 | 0.00 | 188.57 | 188.57 | 188.57 | 1319.99 |
| FROM NODE6 | 3687819462.00 | 3676888122.00 | 3668269662.00 | 3659718231.00 | 3636442452.00 | 0.00 | 3625833702.00 | 3620226322.00 | 25575197953.00 |
| RATE | 188.57 | 188.57 | 188.57 | 188.58 | 188.57 | 0.00 | 188.57 | 188.57 | 1319.99 |
| FROM NODE7 | 3618443342.00 | 3608918344.00 | 3601535692.00 | 3595252632.00 | 3588665762.00 | 3579548242.00 | 0.00 | 3573706622.00 | 25166070636.00 |
| RATE | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 0.00 | 188.57 | 1319.99 |
| FROM NODE8 | 3569127132.00 | 3559008242.00 | 3553838592.00 | 3548414802.00 | 3544088322.00 | 3540468539.00 | 3535320872.00 | 0.00 | 24850266501.00 |
| RATE | 188.57 | 188.57 | 188.57 | 188.57 | 188.57 | 188.58 | 188.57 | 0.00 | 1319.99 |
| ERRORS | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| FROM ALL | 26328753927.00 | 26318205016.00 | 26493240607.00 | 26474816868.00 | 26467603300.00 | 26477269249.00 | 26494123761.00 | 26550013656.00 | 211604026384.00 |
| RATE | 1319.99 | 1320.00 | 1319.99 | 1320.00 | 1319.99 | 1319.99 | 1319.99 | 1320.00 | 10559.95 |

Figure 4.18: Provided Fairness

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

## 5.1 CONCLUSIONS

In order to meet increasing needs and demands of the contemporary world, technology development gained acceleration. For instance, past black and white televisions have left their places to high definition 3D televisions, and gramophone records have been replaced by lossless audio media. Hence, there has been a demand for increased processing capacity to support these, which could be realized by increasing processing frequency and/or the number of processors. As the number of processors increased, the communication between these processors gained significant importance. In addition to the existing communication structures such as on chip buses and crossbars, developers have recently directed their attention to another solution; that is on-chip network which is composed of shared point-to-point links. So far, several NoC examples have been proposed and implemented; however, a standard such as the ones found for on-chip buses has not been developed yet.

In this thesis an example NoC that performs wormhole flow control and source routing is implemented. Firstly, a NoC composed of 8 routers, each having 4 ports and 2 VCs in spidergon topology is designed. Then, this design is described using VHDL and source codes are created. These source codes were simulated on computer environment. Afterwards, to verify the design in real-time, blocks generating dummy traffic and serial interface blocks that communicate with user

interface on the computer and provide control and monitoring of the generated dummy traffic are also designed. Furthermore, together with these testing blocks, the designed NoC is synthesized to be implemented on an FPGA. Thus, using the user interface, i.e., our NoC Monitor, running on the computer, we verified our design in real-time. We observed that created dummy packets were successfully transferred from sources to destinations through the network. Eventually, a total rate of over 14 Gb/s is obtained on the network with a 32-bit flit size operating with 66 MHz clock frequency. If a bus structure was implemented with the same flit size and clock frequency, 2,112 Gb/s rate would be obtained. So we conclude that using network structures on chips, almost a sevenfold performance increase is obtained.

Our VHDL source codes are generalized and parameterized to span ring, spidergon, mesh and torus topologies with diverse buffer sizes, flit sizes and virtual channel numbers. Thus, to be used under various network traffics, it is now possible to generate different NoCs using our NoC Generator.

## 5.2 FUTURE WORK

The NoCs obtained throughout this study can be varied and further developed with new properties. Our tools, the NoC Generator and the NoC Monitor, created in this study can be improved.

The designed network does not provide any bandwidth guarantee. Bandwidth provided to a source node changes according to the traffic produced by other source nodes. In the future, Quality of service (QoS) can be implemented to ensure the guaranteed traffic.

Furthermore, in our current solution, dead lock avoidance is achieved by using a proper routing method. In future studies, it can be managed by router blocks on the network.

Also, instead of network interface units, dummy test blocks are used in the design. Network interface units can be designed and included in the NoC Generator. In these network interface units, various other routing methods can be applied.

Although they are not preferred to be used in on-chip networks, different flow control methods such as circuit-switching, store-and-forward, and virtual-cut-through can be added to the NoC Generator.

Moreover, the NoC Monitor can be developed to create other traffic types to observe different qualities of the network.

## REFERENCES


[1] R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, pp. 3-21, Jan 2009.

[2] P. Bhojwani and R. Mahapatra, "Interfacing cores with on-chip packet-switched networks," *16th International Conference on VLSI Design, Proceedings,* pp. 382-387, 2003.

[3] C. A. Zeferino, F. G. M. E. Santo, and A. A. Susin, "ParIS: A parameterizable interconnect switch for networks-on-chip," in *Proc. 17th Symposium on Integrated Circuits and Systems Design, SBCCI2004* pp. 204-209, 2004.

[4] S. Pasricha and N. Dutt, *On-chip communication architectures : system on chip interconnect*. Amsterdam ; Boston: Elsevier / Morgan Kaufmann Publishers, 2008.

[5] S. Murali, *Designing reliable and efficient networks on chips*. Dordrecht: Springer, 2009.

[6] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys,* vol. 38, pp. 1-51, Mar 2006.

[7] N. D. Enright Jerger and L.-S. Peh. (2009). *On-chip networks*. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00209ED1V01Y200907CAC008, last visited on 23.06.2011

[8] M. Coppola. (2009). *Design of cost-efficient interconnect processing units Spidergon STNoC*. Available: http://marc.crcnetbase.com/isbn/9781420044720, last visited on 23.06.2011

[9] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. San Francisco: Morgan Kaufmann Publishers, 2003.

[10] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers,* vol. 54, pp. 1025-1040, Aug 2005.

[11] P. Tvrdik. *Routing algorithms and switching techniques*. Available: http://pages.cs.wisc.edu/~tvrdik/7/html/Section7.html#Wormhole, last visited on 23.06.2011

[12] A. Agarval, C. Iskander, and R. Shankar, "Survey of NoC architecture and contributions," *Journal of engineering, computing, and architecture*, 2009

[13] R. Gindin, I. Cidon, and I. Keidar, "NoC-based FPGA: Architecture and routing," in *Proc. First International Symposium on Networks-on-Chip, NOCS 2007* pp. 253-262, 2007.

[14] M. Atagoziyev, "Routing algorithms for on chip networks," M. Sc. Thesis Middle East Technical University, 2007.

[15] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *Proc. 19th IEEE International Conference on Parallel and Distributed Processing Symposium* 2005, pp. 155-163.

[16] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Automation Conference* pp. 684-689, 2001.

[17] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. Design, Automation and Test in Europe Conference and Exhibition, Vols 1 and 2, Proceedings,* pp. 1226-1231, 2005.

[18] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proc. 31st Annual International Symposium on Computer Architecture,* pp. 188-197, 2004.

[19] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: a scalable, packet switched, on-chip micro-network," *Designers Forum: Design, Automation and Test in Europe Conference and Exhibition,* pp. 70-73, 2003.

[20] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture,* vol. 50, pp. 105-128, Feb 2004.

[21] R. R. Dobkin, R. Ginosar, and I. Cidon, "QNoC asynchronous router with dynamic virtual channel allocation," in *Proc. First International Symposium on Networks-on-Chip, NOCS 2007* pp. 218-218, 2007.

[22] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone - a communication protocol stack for networks on chip," in *Proc. 17th International Conference on VLSI Design,* pp. 693-696, 2004.

[23] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: Concepts, architectures, and implementations," *IEEE Design & Test of Computers,* vol. 22, pp. 414-421, Sep-Oct 2005.

[24] M. Dall'Osso, C. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," in *Proc. 21st International Conference on Computer Design,* pp. 536-539, 2003.

[25] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration-the VLSI Journal,* vol. 38, pp. 69-93, Oct 2004.

[26] L. Ost, A. Mello, J. Palma, F. Moraes, N. Calazans, and C. Postal, "MAIA - A framework for networks on chip generation and verification," in *Proc. Asia and South Pacific Design Automation Conference, Vols 1 and 2, Asp-Dac 2005* pp. 49-52, 2005.

[27] (23.06.2011). *Atlas - Atlas Wiki - Welcome to GAPH Projects*. Available: https://corfu.pucrs.br/redmine/projects/atlas/wiki

[28] N. Hou, D. L. Zhang, G. M. Du, Y. K. Song, and H. H. Wen, "Design and performance evaluation of virtual-channel based NoC," in *Proc. 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communication,* pp. 294-298, 2009.

[29] L.-S. Peh, "Flow control and micro-architectural mechanisms for extending the performance of interconnection networks," Thesis (PhD), Stanford University, 2001.