UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Contribution to the architecture and implementation of Bi-NoC routers for multi-synchronous GALS systems

Rajeev Kamal

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Enginyeria Electrònica

Contribution to the Architecture and Implementation of Bi-NoC Routers for Multi-Synchronous GALS Systems

Thesis submitted in partial fulfillment of the requirement for the PhD Degree issued by the Universitat Politècnica de Catalunya, in its Electronic Engineering Program.

Rajeev Kamal

Director: Juan Manuel Moreno Arostegui

Barcelona, Spain July, 2017

Declaration of Authorship

I, Rajeev Kamal, declare that this thesis titled, “Contribution To The Architecture And Implementation Of Bi-NOC Routers For Multi-Synchronous GALS Systems” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"*Take up one idea. Make that one idea your life; dream of it; think of it; live on that idea. Let the brain, the body, muscles, nerves, every part of your body be full of that idea, and just leave every other idea alone. This is the way to success, and this is the way great spiritual giants are produced.*"

Swami Vivekananda

Polytechnic University of Catalonia

Abstract

Department of Electronic Engineering (EEL)

Doctor of Philosophy

Contribution To The Architecture And Implementation Of Bi-NOC Routers For Multi-Synchronous GALS Systems

by Rajeev Kamal

Networks-on-Chip (NoC) is an emerging on-chip interconnection centric platform that influences the modern high speed communication infrastructure to improve on-chip communication challenges in the recent many core System-on-Chip (SoC) designs. Continuing shrinkage of feature dimensions of Nano-scale semiconductor devices has been raised serious concerns of the reliability, signal integrity, and quality of services (QoS) of traditional bus based on-chip interconnect infrastructure. NoC represents a major standard move to address these concerns by incorporating state-of-the-art of high-speed data network components (such as routers and switches) and packet-based routing protocols in novel on-chip network infrastructure. A NoC's aim is to provide a reliable on-chip communication platform to facilitate scalable gigascale SoC design.

A multi-synchronous bi-directional NoC's router architecture is proposed in this thesis to enhance the performance of available on-chip communication platform. Using parameterized RTL implementation, we first divide microarchitecture into six blocks as multi-synchronous FIFO, Arbiters, Route Computation, Switch Allocator, Virtual channel Allocator, and Network Interface. Overall architecture of the proposed NoC router consists of five bi-directional ports which supports data transfer between two clock domain of completely arbitrary phase and frequency; and best suited for the Distributed Scalable Predictable Interconnect Networks (DSPIN). In this router, each communication channel allows itself to be dynamically reconfigured to transmit flits in either direction. This added flexibility promises better bandwidth utilization, lower packet delivery latency, and higher packet consumption rate.

We first evaluated performances of each blocks in terms of power, area, and delay with optimizes these blocks to satisfy network key parameters, as well as the impact of allocation on overall network performance. Using structural modeling style and parametric Verilog HDL, all blocks are individually implemented, tested and verified. Finally, all individual blocks are combined to implement bi-directional router's architecture as a whole. Here, we vary the number of nodes for performance evaluation.

A multi-synchronous bi-directional router microarchitecture have been implemented in this thesis, is sufficient to provide throughput challenges, interconnect issues, low latency and high bandwidth in the future Globally Asynchronous Locally Synchronous Systems (GALS) system.

In concise, to enhance the performance of on-chip communications of GALS Systems, a dynamic reconfigurable multi-synchronous router architecture is proposed and implemented to increase the NoC efficiency with changing the path of the communication link in the runtime traffic situation. In order to address GALS issues and bandwidth requirements, the proposed multi-synchronous bidirectional NoC's router is developed and it gives reliable higher packet consumption rate, better bandwidth utilization with lower packet delivery latency. All the input/output ports of the proposed router behave as a bi-directional ports and communicate through a novel multi-synchronous first-in first-out (FIFO) buffer. The bidirectional port is

controlled by a dynamic channel control module which provides a dynamic reconfigurable channel to the router itself and associated with sub-modules.
This proposed multi-synchronous bidirectional router architecture is synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. The performance of the proposed architecture is evaluated in terms of power, area, and delay.

Resumen

Las redes en chip (NoC) constituyen una plataforma de interconexión en chip emergente que influye en la moderna infraestructura de comunicación de alta velocidad para mejorar los desafíos de comunicación en chip de los recientes diseños de sistemas en chip (SoC). La continua reducción de las dimensiones de los dispositivos semiconductores a escala nanométrica ha planteado serias preocupaciones en cuanto a la fiabilidad, la integridad de la señal y la calidad de los servicios (QoS) de la infraestructura de interconexión en chip basada en canal tradicional. NoC representa un paso estándar importante para abordar estas cuestiones incorporando tecnología moderna de componentes de red de datos de alta velocidad (como enrutadores y conmutadores) y protocolos de enrutamiento basados en paquetes en la nueva infraestructura de red en chip. El objetivo de NoC es proporcionar una plataforma de comunicación en chip fiable para facilitar el diseño escalable de SoC.

En esta tesis se propone una arquitectura de enrutador NoC bidireccional multi-síncrono para mejorar el rendimiento de la plataforma de comunicación en chip disponible. Utilizando una implementación RTL parametrizada, primero dividimos la microarquitectura en seis bloques como FIFO multi-síncrono, arbitradores, Cálculo de Rutas, Asignador de Conmutadores, Asignador de canales virtuales e Interfaz de Red. La arquitectura general del enrutador NoC propuesto consta de cinco puertos bidireccionales que soportan la transferencia de datos entre dos dominios de reloj de fase y frecuencia completamente arbitrarias; además, se muestra más adecuada para las Redes de Interconexión Predecibles Escalables y Distribuidas (DSPIN). En este enrutador, cada canal de comunicación permite ser reconfigurado dinámicamente para transmitir las unidades de control de flujo en cualquier dirección. Esta flexibilidad añadida promete una mejor utilización del ancho de banda, una menor latencia de entrega de paquetes y una mayor tasa de consumo de paquetes.

Primero evaluamos las prestaciones de cada bloque en términos de potencia, área y retraso, optimizando estos bloques para satisfacer los parámetros clave de la red, así como el impacto de la asignación en el rendimiento general de la red. Utilizando el estilo de modelado estructural y el Verilog HDL paramétrico, todos los bloques se implementan, prueban y verifican individualmente. Finalmente, todos los bloques individuales se combinan para implementar la arquitectura de enrutador bidireccional como un todo. Aquí, variamos el número de nodos para la evaluación del rendimiento.

En esta tesis se ha implementado una microarquitectura de enrutador bidireccional multi-síncrono, suficiente para atacar retos de rendimiento, problemas de interconexión, baja latencia y un gran ancho de banda en el futuro sistema de Sistemas Síncronos Locales Globalmente Asíncronos (GALS).

En forma concisa, para mejorar el rendimiento de las comunicaciones en chip de los sistemas GALS, se propone e implementa una arquitectura de enrutador multi-síncrono reconfigurable dinámico para aumentar la eficiencia de NoC con el cambio de la ruta del enlace de comunicación en la situación de tráfico en tiempo de ejecución. Con el fin de abordar los problemas de GALS y los requisitos de la banda ancha, el enrutador de NoC bidireccional multi-síncrono propuesto se desarrolla y proporciona una mayor tasa de consumo de paquetes, una mejor utilización de la banda ancha con menor latencia de entrega de paquetes. Todos los puertos de entrada / salida del enrutador propuesto se comportan como puertos bidireccionales y se comunican a través de un nuevo búfer multi-síncrono de tipo FIFO (primera

entrada primera salida). El puerto bidireccional es controlado por un módulo de control de canal dinámico que proporciona un canal reconfigurable dinámico al propio enrutador y asociado con sub-módulos.
Esta propuesta arquitectura de enrutador bidireccional multi-síncrono se sintetiza utilizando Xilinx ISE 14.7 y el dispositivo FPGA Virtex 6 XC6VLX760 se considera como la tecnología objetivo. El rendimiento de la arquitectura propuesta se evalúa en términos de potencia, área y retraso.

Acknowledgements

I would like to express my deep sense of worship and indebtedness to my supervisor Professor Juan Manuel Moreno Arostegui, Electronics Engineering Department, Polytechnic University of Catalonia for his inestimable guidance, directive instructions, continuous encouragement, constructive comments and inspirations throughout the course of my research. He has not only helped me in bringing the thesis to this shape, and also stretched his helping hands whenever I was in need. I would also like to thank the Professor Xavier Aragones Coordinator of the PhD Program in Electronic Engineering, UPC for opening the doors of this university, and give me the opportunity of professional growth during the development of my doctoral studies.

I gratefully acknowledge , all the faculty members, technical staff and administrative staff members of the Electronics Engineering Department for their care and warmth shown towards me during my research period in this University and their active support and encouragement always helped me during my difficult situations.

I would like to express my deep sense of gratitude to Dr. Arti Noor, Director, School of Electronics, CDAC, Noida India for her support, precious suggestions and timely encouragement.

I am especially grateful to my friend Vikas Nehra, Assistant Professor, DCRUSTM Murthal India, who supported me emotionally and financially. I always knew that you believed in me.

I am grateful to Dr. Neeraj Kumar Shukla, Associate Professor, The NorthCap University, Gurgaon India for his selfless support and innovative suggestions during this research work.

I wish to acknowledge my student Gaurav Manocha, MTS MentorGraphic Noida India for his supports, precious suggestion and timely encouragement for simulation and Verification of my design.

I wish to acknowledge my friend and researcher Dr. Abhishek Basu, Head of Department, RCC Institute of Technology, Kolkata India, for his support, precious suggestions and timely encouragement.

I would never forget all the chats and beautiful moments I shared with some of my friends. They were fundamental in supporting me during these stressful and difficult moments. I would like to mention some friend here : Vikas, Rupesh, Tarun, Mandeep, Lluís , Saoni, and Saeed.

I am indebted to my wife Aarti, for her immense love and unconditional support, thanks for always being there for me and my little doraemon Vishu, for the games that we will play, and for the life that you will teach me to discover.

I would like to express my sincere gratitude to my in-laws, brothers, sisters, friends and family members for their constant encouragement and support rendered during my course of research study.

Finally and most importantly, I thank my family especially my father Mr. Sheo Kumar Pathak and mother Mrs. Shail Devi for their unconditional support, their unwavering confidence in my abilities and their encouragement in all of my endeavors. You made me live most unique, magic, and carefree childhood that has made me who I am now! Without you, this journey

would not have been possible.

Needless to say, without all the above help and support the writing and production of this thesis would not have been possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AMBA	Advanced Microcontroller Bus Architecture
ASPIN	Asynchronous Scalable Predictable Interconnect Network
BE	Best-effort
CMP	Chip Multi-Processor
DPA	Dynamic Priority Arbiter
DSPIN	Distributed Scalable Predictable Interconnect Network
FIFO	First In First out
FPA	Fixed Priority Arbiter
GALS	Globally Asynchronous Locally Synchronous
GT	Guaranteed Throughput
HOL	Head of Line
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductor
MPSoC	Multi-Processor SoC
MTBR	Mean Time Between Synchronization Failures
NI	Network Interface
NIC	Network Interface Controller
NIU	Network Interface Unit
NoC	Network On Chip
OCP	Open Core Protocol
PDF	Probability Density Function
QoS	Quality of Service
RC	Route Computation
RRA	Round Robin Arbiter
RTL	Register Transfer Logic
SoC	System On Chip
VCI	Virtual Component Interface
VCT	Virtual Cut Through
XPA	Xpower Analyzer
XPE	Xpower Estimator

Dedicated to my wife Arti
my little Doremon Vishu
and
Indian Philosophy & Way of life
of which
my parents are
an integral part

Chapter 1

Introduction

Network-on-Chip (NoC) is a general-purpose on-chip interconnection network that offers great promises to moderate the ever increasing communication complexity of modern many-core system-on-chip (SoC) designs. Specifically, a city-block style tiled NoC architecture proposed in [1–3] has gained high popularity due to its simplicity and flexibility. NoC is an emerging on-chip interconnection centric platform that influences modern high speed communication infrastructure to improve ever increasing on-chip communication challenges of recent many-core SoC designs. Continuing shrinkage of feature dimensions of nano-scale semiconductor devices has raised serious concerns of the reliability, signal integrity, and quality of services (QoS) of traditional bus-based on-chip interconnect network. NoC represents a major standard move to address these concerns by incorporating state-of-art high-speed data network components (such as routers and switches) and packet-based routing protocols in novel on-chip network structure. A NoC provides a reliable on-chip communication platform to facilitate scalable giga-scale SoC design.

1.1 Motivation and context

Traditionally, a SoC or an MPSoC system interconnects intellectual property (IP) components by using a bus-based interconnect system. When the number of participating components is more than ten, then the bus system will have a performance bottleneck problem [3]. In order to solve the performance bottleneck problem, a fully crossbar interconnect can be used. However, this approach will imply a wiring complexity in the circuit, in which wires could be more dominant than the logic parts, especially when the number of the interconnected components is very high. Another problem in the fully crossbar interconnect is the effect of electromagnetic interference that can disturb the interconnect functionality. A Point-to-point interconnects (dedicated wires) is also another alternative solution to the performance bottleneck problem and to the wiring complexity problem. However, this approach is not flexible. Instead of connecting the top-level components by routing the dedicated wires, an on-chip interconnection network can be implemented and interconnect the interacting components by routing packets through the network [4].
Since interconnect technology affects more profoundly on chip performance and power usage, improving on-chip communication technology has become increasingly important to researchers and processor manufacturers [5]. A high-throughput communication infrastructure is required to meet the bandwidth requirement of each data communication flows generated due to interacting processors in the MPSoC systems. This issue can be potentially handled by a communication infrastructure based on the network-on-chip (NoC), which has better scalability to provide sufficient communication bandwidth.
On-chip network infrastructure also enables advanced intellectual property (IP) communication concepts for MPSoC. In embedded MPSoC systems, NoCs can provide a flexible communication infrastructure, in which several components such as microprocessor cores, MCU, DSP, GPU, memories and other intellectual property (IP) components can be interconnected

by using reusable NoC routers via general modular interfaces. The MPSoC systems can also be reconfigured for a certain embedded computing application and can be customized to improve the communication performance in the application. Hence, the NoC-based systems combine performance with design modularity [6]. The innovation of a flexible NoC communication infrastructure will enable accordingly the IP vendors to sell not only their IP components but also a system architecture [7].
IP blocks communicate over the NoC using a three layered communication scheme, referred to as the Transaction, Transport, and Physical layers as shown in figure 1.1.
The Transaction layer defines the communication primitives available to interconnected IP

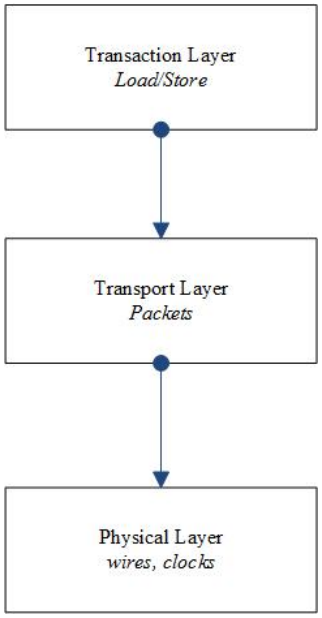


FIGURE 1.1: Three layer communication scheme

blocks. The transaction layer defines how information is exchanged between NoC Interface Units (NIUs) to implement a particular transaction. For example, a NoC transaction is typically made of a request from a master NIU to a slave NIU, and a response from the slave to the master.
The Transport layer defines rules that apply as packets are routed through the switch fabric. Very little of the information contained within the packet is needed to actually transport the packet. The packet format is very flexible and easily accommodates changes at transaction level without impacting transport level.
The Physical layer defines how packets are physically transmitted over an interface, much like Ethernet defines 10Mb/s, 1Gb/s, etc. NoC links between switches can be optimized with respect to bandwidth, cost, data integrity, and even off-chip capabilities, without impacting the transport and transaction layers.
A NoC channel could be either unidirectional or bidirectional. A bidirectional channel NoC architecture enhances the performance of on-chip communication. In this, each communication channel allows to be dynamically self-reconfigured to transmit flits in either direction. This added flexibility promises better bandwidth utilization, lower packet delivery latency, and higher packet consumption rate. Novel on-chip router architecture is developed to support dynamic self-reconfiguration of the bidirectional traffic flow. The area-efficient NoC router delivers better performance and requires smaller buffer size than that of a conventional NoC. This thesis is motivated to provide an architecture and implementation of bidirectional NoC's router for multi-synchronous GALS (Globally Asynchronous Locally Synchronous) infrastructure.

1.2 State of the art

Networks-on-Chip (NoCs) are widely accepted as a promising approach for addressing the communication challenges associated with future Multi-Processors System-on-Chip (MP-SoCs) in the face of further increases in integration density [7-8]. In the recent multi-core systems, e.g. Google Brain project, NoC routers require to transfer a large amount of communication data among processors [9-11]. Networks-on-Chips (NoCs) [12] represent a widely accepted solution to connect multiple cores, due to its scalability, flexibility and high bandwidth properties. Several regular patterns for NoC architectures have been proposed and implemented in [13]. These NoC architectures have gained popularity due to their scalability, simplicity and flexibility. In such NoC architectures, main switching component (router) behaves like a synchronous island. These routers are connected via two unidirectional links with the neighboring router. A typical 2-D mesh 5-stage pipelined virtual-channel based NoC router is shown in figure 1.2.

A NoC router is responsible for forwarding the incoming flits (large network packets are

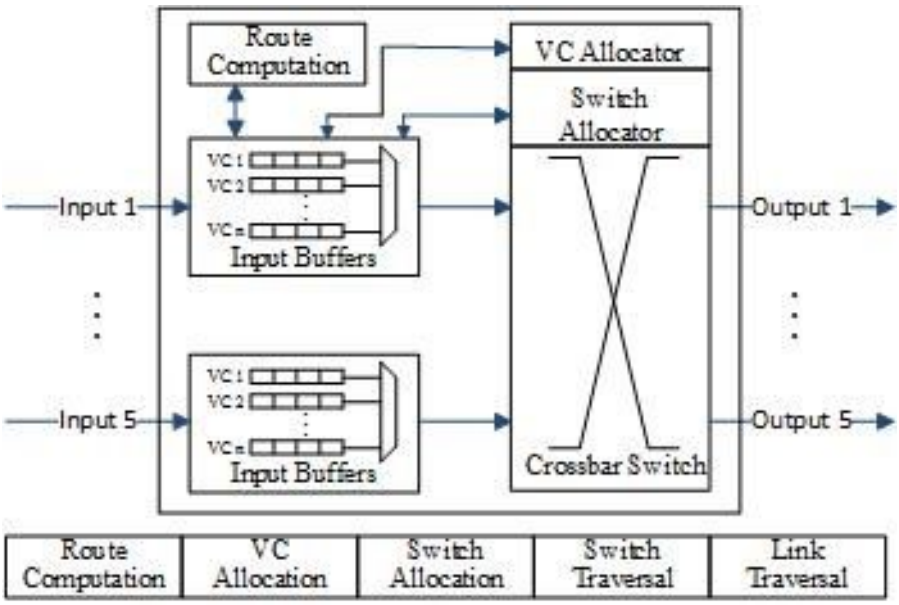


FIGURE 1.2: A typical 2-D mesh 5-stage pipelined virtual-channel based NoC router

broken into small pieces called flits) from the input buffers, with its proper arbitration and routing algorithms towards outputs ports [1-3]. The router makes a decision based on information collected from the network and the decision may be centralized or distributed [14]. Router can implement various functionality from simple switching to smart routing. From the switching point of view, in a circuit-switching network, routers may be designed without buffers but in a packet-switching network, buffering is needed to support heavy data transfer [13].

1.2.1 Network-on-Chip

In the recent years, researchers have proposed various router designs on the basis of arbitration strategy, switching techniques, flow control, buffer management, power consideration, etc. Dally and Towles proposed a packet based switching router [2], whereas Wolkotte et. al. circuit-switching based router architecture[15]. Wormhole based packet forwarding design is proposed by Albenes [16]. Switching techniques are again a parameter for designing

the NoC router. After initial classification, it can be classified upon network characteristics, packet switching are further classified as Wormhole, Store and Forward (S & F) and Virtual Cut Through (VCT). After combination of different switching techniques adhoc switching techniques can be developed. MANGO and ETHEREAL uses such techniques [17 - 20].
The flow control mechanism determines network parameter e.g. channel bandwidth, buffer capacity and control state are allocated to a flit traversing the network. It may be buffer less or buffered. The buffered flow control is further classified as credit based, handshaking, ACK/NACK etc. In SoCIN NOC implementation, handshaking flow control is used [22], Whereas credit based flow control is used in QNOC[21 - 22] and ACK/NACK protocol is used in the implementation of XPIPES[23].
Another important aspect of NoC is virtual channels, which split single channel into many channels that provides path for the packet to be routed [24]. This concept removes the problem of HOL [25] and provides the low latency path for high priority data flits [26]. Bjerregaard and Sparsø have proposed the design and architecture of a virtual channel router [19 - 20]. A large number of virtual channels required higher buffer capacity but that will reduce network contention thereby reducing latency. Saastamoinen, Zimmer, and Bolotin proposed the buffer architecture and their implementation has been described [27 - 29]. In the recent years shared buffer architecture has been also described [30].
International Technology Roadmap for Semiconductor (ITRS) [31] states that "Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification and test[32]". Faults in a NoC architecture can be categorized as hard fault and soft fault NoC [32 - 33]. Fault tolerant routing protocol with their implementation and error detection and correction schemes in data for NoC link has been proposed in [34 - 37].
The Network interface (NI), implements the interface to the IP module and is responsible for the packetization/depacketization of data traffic. Radulescu et. al. provides an efficient on network interface that offers guaranteed services, shared-memory abstraction and flexible network configuration [38]. Bhojwani and Mahapatra compared software and hardware NI implementation [39].
A NoC is capable of supporting different classes of service level such as Guaranteed Throughput (GT) and Best-Effort (BE) [40]. A guaranteed-throughput (GT) router guarantees uncorrupted, loss-less, and ordered data transfer, and both latency and throughput over a finite time interval. Best-effort delivery describes a network service in which the network does not provide any guarantees that data is delivered or that a user is given a guaranteed quality of service level or a certain priority. In a best-effort network all users obtain best-effort service, meaning that they obtain unspecified variable bit rate and delivery time, depending on the current traffic load. The GT and BE need to support an arbitration technique. NoCs have implemented various different techniques e.g. Round Robin, First Come First Serve, Priority Based, and Priority Based Round Robin. The Round Robin technique is implemented in the SPIN and RASoC [41], whereas QNOC, XPIPES and Philips NoC adopted Priority Based Round Robin.
A NoC system architecture is further categorized as homogeneous and heterogeneous architecture, most of NoCs support homogeneous whereas XPIPES support heterogeneous architecture. In the recent years research article, authors adopt Globally Asynchronous and Locally Synchronous (GALS) clocking scheme to reduce power and energy consumption in CMPs (Chip Multiprocessors) or SoCs [42]. Asynchronous circuit can reduce power dissipation by eliminating a global clock signal on a single chip entirely [43 - 44]. In the asynchronous NoC, multiple communications are simultaneously done over asynchronous routers and link between processing elements [45 - 53]. Qnizawa et. al. proposed a high-throughput and compact delay-intensive asynchronous router [54].
According to Sheibanyard et. al. [55], the GALS system is further categorized into DSPIN

(Distributed Scalable Predictable Interconnect Network), which has a multiple synchronous network and ASPIN (Asynchronous Scalable Predictable Interconnect Network) is fully asynchronous. CHAIN, ANOC, MANGO, QNOC, and QOS are examples of ASPIN. Whereas Tushar et. al. and Ivan et. al. [56] proposed DSPIN Network-on Chip. The Difficulty of synchronization in DSPIN is resolved by bi-synchronous router whereas in ASPIN by Synchronous $\leftrightarrow$ Asynchronous converter.
After analyzing DSPIN and ASPIN, researchers found both networks are scalable, but the asynchronous approach shows a better saturation threshold than the synchronous one. Regarding the silicon area, both implementations have similar foot-prints, if long wire buffers are taken into account. In systems containing large clusters, the energy dissipated to transmit a packet is higher in the asynchronous approach than in the synchronous approach, which is a major drawback of the ASPIN. Of course packet latency is better in ASPIN with respect to DSPIN but this problem is overcome by shared buffered architecture.
Classification of NoC architecture on the basis of clock is shown in figure 1.3. The choice of NoC architecture is determined by one or more design criteria, such as requirements for performance (latency and throughput), power consumption, QoS, reliability, scalability, and implementation cost.A large number of NoC architectures have been published such as Dally's NoC , AETHEREAL, XPIPES and NOSTRUM which have synchronous architecture. some proposed asynchronous NoCs are CHAIN, MANGO, QNOC, ANOC and QoS. We describe a few of these NoC architectures below.

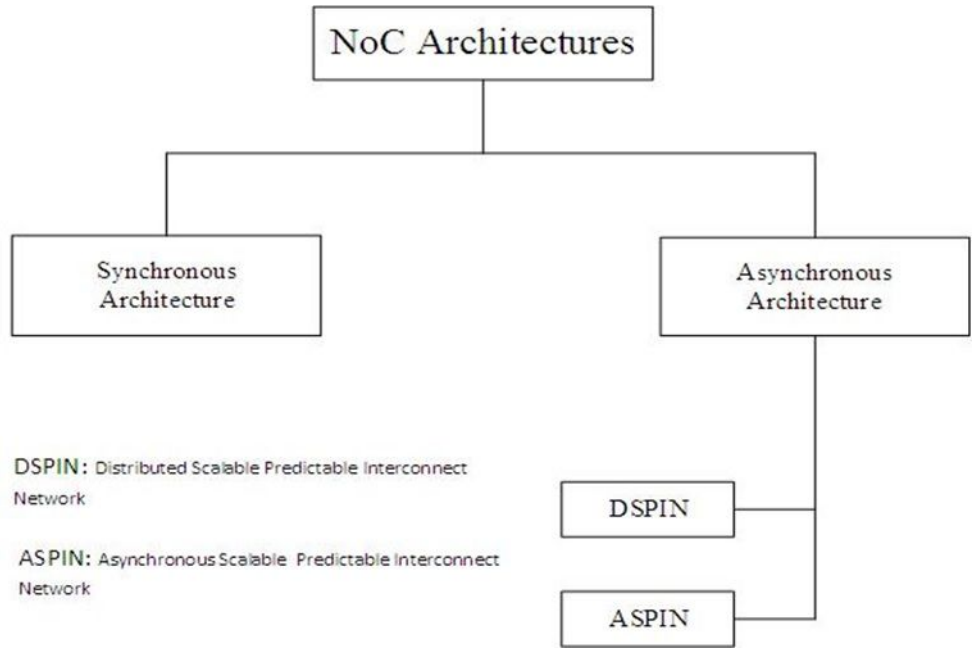


FIGURE 1.3: Classification of NoC architecture on the basis of clock

- Synchronous Architectures
  - AEthereal [18]
    - Developed by Philips.
    - Synchronous indirect network (but also supports irregular topologies).
    - Wormhole switching.
    - Contention-free source routing based on TDM.
    - GT as well as BE QoS.

    - GT slots can be allocated statically at initialization phase, or dynamically at runtime.
    - BE traffic makes use of non-reserved slots, and any unused reserved slots.
    - Link-to-link credit-based flow control scheme between BE buffers.

  - Xpipes [23]
    - Developed by the University of Bologna and Stanford University.
    - Source-based routing, WH switching.
    - Supports OCP standard for interfacing nodes with NoC.
    - Supports design of heterogeneous, customized (possibly irregular) network topologies.
    - Go-back-N retransmission strategy for link level error control.
    - Errors detected by a CRC (cycle redundancy check) block running concurrently with the switch operation.
    - XpipesCompiler and NetChip compilers.
    - Tools to tune parameters such as flit size, address space of cores, max. number of hops between any two network nodes, etc..
    - Generate various topologies such as mesh, torus, hypercube, Clos, and butterfly.

  - Nostrum [57]
    - Developed at KTH in Stockholm.
    - Direct network with a 2-D mesh topology.
    - Store and Forward switching with hot potato (or deflective) routing.
    - Support for switch/router load distribution, guaranteed bandwidth (GB), and Multicasting.
    - GB is realized using looped containers.
    - implemented by VCs using a TDM mechanism.
    - container is a special type of packet which loops around VC.
    - Multicast: simply have container loop around on VC having recipients.
    - Switch load distribution requires each switch to indicate its current load by sending a stress value to its neighbors.

- Asynchronous Architectures

  - MANGO [20]
    - Message-passing Asynchronous Network-on-chip providing GS over open core protocol (OCP) interfaces.
    - Developed at the Technical University of Denmark.
    - Clockless NoC that provides BE as well as GS services.
    - NIs (or adapters) convert between the synchronous OCP domain and asynchronous domain.
    - Routers allocate separate physical buffers for VCs for simplicity, when ensuring GS.
    - BE connections are source routed.
    - BE router uses credit-based buffers to handle flow control.
    - Length of a BE path is limited to five hops.

    - Static scheduler gives link access to higher priority channels.
    - Admission controller ensures low priority channels do not starve.

  - QNoC [21]
    - Developed at Technion in Israel.
    - Direct network with an irregular mesh topology.
    - WH switching with an XY minimal routing scheme.
    - Link-to-link credit-based flow control.
    - Traffic is divided into four different service classes.
      - signaling, real-time, read/write, and block-transfer.
      - signaling has highest priority and block transfers lowest priority.
      - every service level has its own small buffer (few flits) at switch input.
    - Packet forwarding is interleaved according to QoS rules. High priority packets able to preempt low priority packets.
    - Hard guarantees not possible due to absence of circuit switching. Instead statistical guarantees are provided.

- Other Popular NoC's Architectures

  - HERMES [58]
    - Developed at the Faculdade de Informática PUCRS,Brazil.
    - Direct network.
    - 2-D mesh topology.
    - WH switching with minimal XY routing algorithm.
    - 8 bit flit size; first 2 flits of packet contain header.
    - Header has target address and number of flits in the packet.
    - Parameterizable input queuing to reduce the number of switches affected by a blocked packet.
    - Connectionless: cannot provide any form of bandwidth or latency GS.

  - Octagon [59]
    - Developed by STMicroelectronics.
    - Direct network with an octagonal topology.
    - 8 nodes and 12 bidirectional links.
    - Any node can reach any other node with a max of 2 hops.
    - Can operate in packet switched or circuit switched mode.
    - Nodes route a packet in packet switched mode according to its destination field. Node calculates a relative address and then packet is routed either left, right, across, or into the node
    - Can be scaled if more than 8 nodes are required i.e. Spidergon.

  - SOCBUS [60]
    - Developed at Linköping University.
    - Mesochronous clocking with signal retiming is used.
    - Circuit switched, direct network with 2-D mesh topology.
    - Minimum path length routing scheme is used.
    - Circuit switched scheme:

        - deadlock free.
        - requires simple routing hardware.
        - very little buffering (only for the request phase).
        - results in low latency.
      - Hard guarantees are difficult to give because it takes a long time to set up a connection.

- SPIN [61]
  - Scalable programmable integrated network (SPIN).
  - Fat-tree topology, with two one-way 32-bit link data paths.
  - WH switching, and deflection routing.
  - Virtual socket interface alliance (VSIA) virtual component interface (VCI) protocol to interface between PEs.
  - Flits of size 4 bytes.
  - First flit of packet is header
    - first byte has destination address (max. 256 nodes)
    - last byte has checksum
  - Link level flow control. GS is not supported.

1.2.2 Bi-NoC Router Architecture

In above discussed NoC infrastructure, adjacent routers are connected with two unidirectional communication links and these communication links are hardwired to handle communication data into only one direction at a time. It's frequently seen in the simulation result that one channel can be collapsed by a huge number of packets in one route direction while the reverse route direction remains unused. This would tend to reduce throughput and therefore result in an unproductive use of communication channel. A straight forward approach to ease such an issue is to have an opposite passageway. An opposite passageway supports dynamic opposite path transfer. By means of electronic signals, an opposite passageway can be reversed to support additional capability to the passage having high volume of traffic.
Foruque et. al. [62] presented a unique configurable data channel called 2X-link. This can change its maintained bandwidth at runtime on demand. 2X-Links use tri-state logic to support bidirectional transmission for an adaptive on-chip communication infrastructure. In a BiNoC [63], a dynamic reconfigurable architecture has been suggested to change of the path of the links in the runtime traffic situation. To accommodate uneven distributed traffic in the core of the network, BiNoC architecture was initially introduced and further optimized in FSNoC [64] and BiLink [65]. Main difference between these architectures and 2X-Link is how the link direction control is implemented. In BiNoC, FSNoC and BiLink channel direction decision is made using a distributed channel direction control protocol, whereas in 2X-Link, the proposed direction decision mechanism is centralized and each link is configured in advance based on application bandwidth requirements. Cho et. al. [66] proposed a pressure based channel direction control protocol. This is different from the acquisition based channel control protocol used in BiNoC. In this thesis, we implement an acquisition based dynamic channel control protocol that supports multi-synchronous GALS infrastructure. Table 1 shows the comparison of different bi-direction NoC on the basis of channel direction control.

Dynamic reconfigurable router architectures have been suggested to enhance NoC proficiency by changing the path of the links in the runtime traffic situation, since normal traffic

TABLE 1.1: Comparison of different bi-direction NoC

Name	*Channel Direction Control (CDC)*
2X –Link	Centralized CDC
BiNoC	Distributive CDC
BiLink	Aggressive bidirectional Link
FSNoC	Acquisition based CDC
Cho et. al.	Pressure based CDC

pattern is commonly unevenly distributed in the core of the network. To accommodate for such kind of traffic pattern, a bi-directional NoC ((BiNoC) [63] architecture was presented and further optimized as FSNoC [64] and BiLink [65]. However, the greater part of the accentuation on the current reconfigurable NoC architecture has been concentrating on upgrading the design of the switch itself. The optimization of the interconnection between two adjacent routers is hardly satisfied. On the other side, network coding was introduced in communication domain to maximize channel bandwidth to accomplish noteworthy improvement in the system throughput. Over a single physical channel, an extra coding unit was inserted between each pair of routers to enable the data transmission from both ends at a time. BiLink [65] uses an additional link stage amongst routers and transmits two flits in one link for every clock cycle using stage pipelining if both routers require utilizing the current link. To increase additional effective bandwidth, each link path can be arranged in every clock cycle to supply different traffic loads from both sides.

1.3 Objective

Above discussed bidirectional NoCs are not suitable for the globally asynchronous, locally synchronous (GALS) architecture and it is believed that GALS approach is best suited for the MPSoC(Multi-Processor SoC) and packet based NoC. Initially Dally and Towles [2] introduced the concept of packet switching within the on-chip networks communication structure and same concept followed by the Benini and Micheli in [3]. In the research articles [67], different types of NoC and GALS based SoC are explained.
In this thesis, we will contribute to the architecture and implemented a bidirectional NoC router for the multi-synchronous GALS infrastructure. A better approach to design a bidirectional NoC for the multi-synchronous GALS system is shown in figure 1.3. Its consists of five bidirectional ports which support data transfer across two clock domains of arbitrary phase and frequency and it is best suited for Distributed Scalable Predictable Interconnect Networks (DSPIN) . We know that clock cross domain (CCD) and synchronization are the main requirements for the GALS infrastructure and these concepts greatly influence the design of NoC architectures. The main issue in GALS architectures is the possibility of synchronization failure (metastability) between two adjacent routers within a mutli-synchronous NoC. The proposed NoC architecture uses a novel multi-synchronous FIFO that implements the necessary functionality to remove clock cross domain issues.

In order to address GALS synchronization issue and bandwidth requirement, the proposed multi-synchronous bi-directional NoC's router is developed as shown in figure 1.4. It consists of five modules which are situated at the east, west, north, south and northwest directions. The communication between local module and processing element (PE) is performed using the network interface controller (NIC). All the input/output ports of different modules behave as a bi-directional port and communicate through a novel multi-synchronous FIFO.

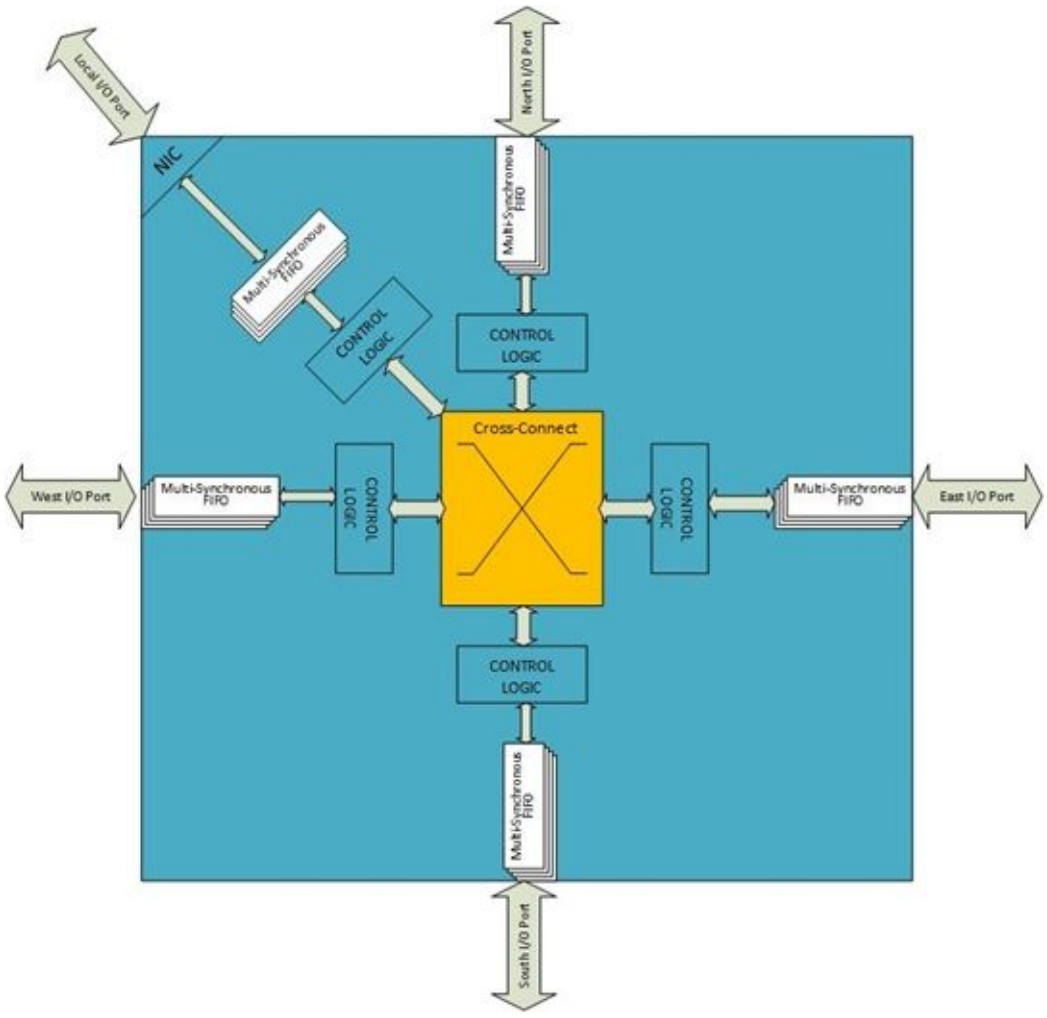


FIGURE 1.4: Top/Block level diagram of proposed multi-synchronous bi-directional NoC

The bi-directional port is controlled by a dynamic channel control module which provides dynamic reconfigurable channel to the router itself and associated sub-module. A synchronous cluster is composed by local subsystem and the associated router. The intra-cluster wire is the longest wire in this method, e.g. we can consider wire length from 1 in-out port on the one side to 1 in-out port on the other side, and this wire length may not be stretched beyond cluster size. This multi-synchronous bi-directional NoC architecture is well-matched with the GALS style, where synchronous islands communicate to each other asynchronously. Here every router is synchronized with the same network clock frequency, but due to different path delay, phase skew can occur amongst two adjacent routers. Hence, each subsystem can be considered as an individual synchronous island or can be independent of network clock frequency. Hence two types of interface will be considered, i.e. router-PE interface and router-router interface. The router-PE interface will be completely asynchronous and router-router interface will be mesochronous (same frequency but different phase). In order to address such two types of interface, a novel multi-synchronous FIFO will be required to resolve mesochronous and asynchronous problem.

Here we proposed microarchitectural modification of available bi-directional NoC's router that improve delay, area and power consumption. We contributed to the architecture and implemented a bi-directional NoC's router for the multi-synchronous GALS systems using parameterized RTL code. The contributions of this thesis are summarized as follows.

1. A novel multi-synchronous FIFO buffer architecture that supports valid/ready flow control mechanism at all the different input/output ports of the routers.

2. A new channel directional control protocol, called Dynamic Channel Control, is proposed for the bi-directional channels. This is based on handshaking protocol.

3. Implementation of the proposed NoC's router and detailed analysis in term of area, delay, and power dissipation.

1.4 Document Organization

The remaining chapters are divided into three chapter groups, i.e. the contributions to the thesis are represented by Chap. 2 and Chap. 3, the implementation chapters describing the implementation of this thesis (Chap. 4 – Chap. 8), and the concluding chapter represented by Chap. 9. A brief descriptions of each chapter is provided in the following paragraphs..

- *Chap. 2* describes contribution to the architecture and investigates implementation aspects for high performance traditional BiNoC routers. After contributing to the traditional architecture, the proposed architecture is called as Multi-synchronous BiNoC (MBiNoC) architecture. It mainly consists of three parts: multi-synchronous FIFO module at the interface of the router, dynamic channel control module and router internal architecture.

- *Chap. 3* discusses elementary FIFO designs. We investigate different asynchronous FIFO types, discuss approaches for providing, scalability and robustness for multiple clock domains, and evaluate power, area and delay using commercial FPGA Virtex 6 family device XC6VLX760.

- *Chap. 4* discusses elementary arbiter designs. We investigate different arbiter types, discuss approaches for providing fairness, scalability and support for multiple priority levels, and evaluate power, area and delay using commercial FPGA Virtex 6 family device XC6VLX760.

- *Chap. 5* discusses a routing computation module that allows routers to be embedded in arbitrary network topologies. This chapter also describes how routing modules are responsible for keeping the network connected and resolving contention for the same resource while allowing multiple packets to flow in the network concurrently.

- *Chap. 6* discusses the relative work in the field of Network Interface (NI) design and implementation. This chapter also deals with complete RTL model of NI and its sub-module.

- *Chap. 7* similarly investigates allocators. In this chapter we discussed different types of available allocators and their implementations. We develop a scheme to alleviate inherent fairness issues in wavefront allocation. this chapter also discusses, how the basic allocator designs can be used to implement VC allocation. We give an overview of practical implementation variants; describe VC allocation and Switch allocation, and present detailed evaluation results for power, area and delay.

- *Chap. 8* Discusses new contributions of this thesis. The directions for future works are also briefly described in this chapter.

Chapter 2

Contributions To The Architecture

2.1 Summary

To enhance the performance of on-chip communications of Globally Asynchronous Locally Synchronous Systems (GALS), a dynamic reconfigurable multi-synchronous router architecture is proposed to increase network on chip (NoC) efficiency by changing the path of the communication link in runtime traffic situation. In order to address GALS issues and bandwidth requirements, the proposed multi-synchronous bi-directional NoC's router is developed and it guarantees higher packet consumption rate and better bandwidth utilization with lower packet delivery latency. All the input/output ports of the proposed router behave as a bidirectional ports and communicate through a novel multi-synchronous first-in first-out (FIFO) buffer. The bi-directional port is controlled by a dynamic channel control module which provides a dynamic reconfigurable channel to the router itself and associated sub-modules. The flow direction at each channel is controlled by a dynamic channel control algorithm. Implemented with a pair of algorithmic finite state machines, this dynamic channel control algorithm is shown to provide high performance, be free of deadlocks, and starvation-free.

2.2 Introduction

Network-on-Chip (NoC) is a general-purpose on-chip interconnection network that offers great promises to mitigate the ever increasing communication complexity of modern many-core system-on-chip (SoC) designs. Specifically, a city-block style tiled NoC architecture proposed in [68-70] has gained high popularity due to its simplicity and flexibility.
Network-on-Chip(NoC) has become a promising approach to solve the communication bottleneck in the modern many-core system-on-chip. With the potential deployment of many-core systems on new applications such as big data, artificial intelligence and deep machine learning, the NoC router requires to transfer a larger amount of communication data among processors. For example,the Google Brain project [71-72] uses 1000 machines to train a deep neural network. Each machine contains 16 cores on it and a subset of neural network will be mapped on each of them [71]. The requirement of the data bandwidth is high and uneven due to the interleaving of the feed-forward and back propagation training phases. To address the intensive bandwidth requirement of these applications, a higher throughput NoC router architecture is essential and crucial for the next generation of many-core systems.
Networks-on-Chips (NoCs) [73] represent a widely accepted solution to connect multiple cores, due to its scalability, flexibility and high bandwidth properties. Several regular patterns for NoC architectures have been proposed and implemented in [33]. These NoC architectures have gained popularity due to their scalability, simplicity and flexibility. In such NoC architectures, main switching component (router) behaves like a synchronous island. In such NoC architecture, neighboring routers are connected via a pair of unidirectional communication channels. Each channel is hard-wired to handle either outgoing or incoming traffic. From

simulation results, it is observed that quite often one channel may be overflowed with heavy traffic in one direction, while the channel in the opposite direction remains idle. This leads to performance loss and inefficient resource utilization. A typical 2-D mesh 4-stage pipelined virtual-channel based NoC router is shown in figure 2.1.

In a typical virtual-channel flow-control based router, the flits are routed via a four-stage

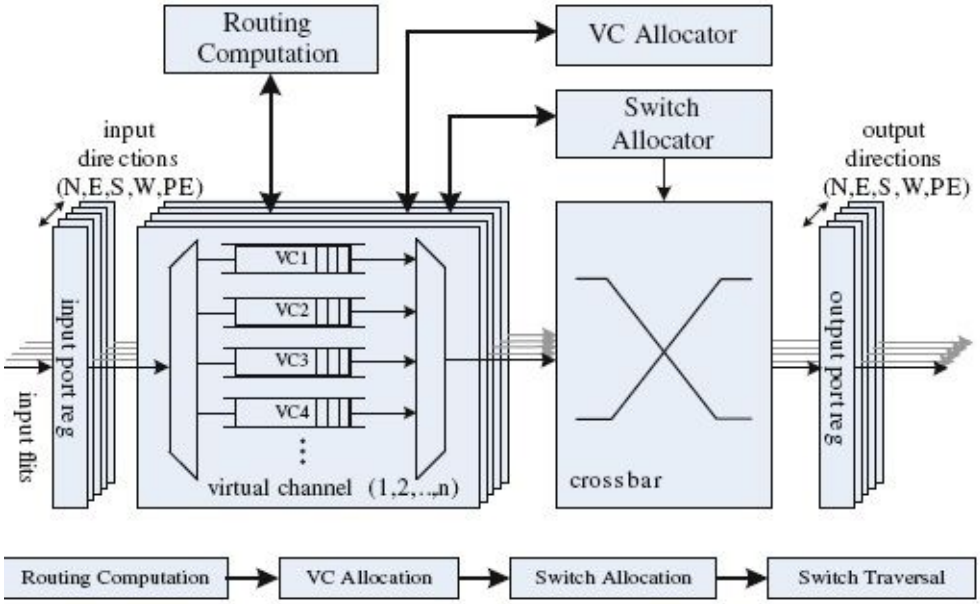


FIGURE 2.1: A typical 2-D mesh 4-stage pipelined virtual-channel based NoC router [68]

pipeline: *routing computation (RC)*, *virtual-channel allocation (VA)*, *switch allocator (SA)*, and *switch traversal (ST)*.

One incoming flit that arrives at a router is first written to an appropriate input virtual-channel queue and waits to be processed. When a head flit reaches the top of its virtual-channel buffer queue and enters the RC stage, it is decoded by the RC module and generates an associated direction request. The direction request of this flit is then sent to the VA module to attain virtual-channel at the downstream router. There might be some contentions among packets that request for the same virtual-channel at the downstream router. The loser packets will be stalled at the VA stage and the following flit in the previous stage will also be blocked due to this contention failure. Note that the processes of RC and VA actually take place only on the head flit. The subsequent body flits and tail flit of a packet simply accede to the routing decision acquired by the head flit and require no further processing at the RC and VA stages. Once a decision on the output virtual-channel selection is made at the VA stage, the SA module will assign physical channels to intra-router flits. Flits granted with a physical channel will traverse through the crossbar switch to the input buffer of the down-stream router during the ST stage, and the process repeats until the packet arrives at its destination.

In above discussed NoC infrastructure, adjacent routers are connected with two unidirectional communication links and these communication links are hardwired to handle communication data into only one direction at a time. It's frequently seen in the simulation result that one channel can be collapsed by a huge number of packets in one route direction while the reverse route direction remains unused. This would tend to reduce throughput and therefore result in an unproductive use of communication channel.

This uneven NoC traffic pattern is very similar to the uneven traffic flow pattern during rush hours on a city highway in a metropolis. A common solution to alleviate such a problem is to

implement reversible lanes (counter-flow lanes). A reversible lane is a highway driving lane with dynamically reversible driving direction assignment. Using electronic signs, the driving direction on a counter-flow lane can be reversed to provide more capacity to the direction with heavier traffic volume.
Dynamic reconfigurable router architectures have been suggested to enhance NoC efficiency by changing the path of the links in runtime traffic situation, since normal traffic pattern is commonly unevenly distributed in the core of the network. To accommodate for such kind of traffic pattern, a bi-directional NoC ((BiNoC) [63] architecture was presented and further optimized as FSNoC [64] and BiLink [65]. However, the greater part of the accentuation on the current reconfigurable NoC architecture has been concentrating on upgrading the design of the switch itself. The optimization of the interconnection between two adjacent routers is hardly satisfied. On the other side, network coding was introduced in communication domain to maximize channel bandwidth to accomplish noteworthy improvement in the system throughput. Over a single physical channel, an extra coding unit was inserted between each pair of routers to enable the data transmission from both ends at a time. BiLink [65] uses an additional link stage amongst routers and transmits two flits in one link for every clock cycle using stage pipelining if both routers require utilizing the current link. To increase additional effective bandwidth, each link path can be arranged in every clock cycle to supply different traffic loads from both sides.
Above discussed bidirectional NoCs are not suitable for the globally asynchronous, locally synchronous (GALS) architecture and it is believed that GALS approach is best suited for the MPSoC and packet based network-on-chip. Initially Dally and Towles [2] introduced the concept of packet switching within the on-chip networks communication structure and same concept followed by the Benini and Micheli in [3]. In the research articles [74-75], different types of NoC and GALS based SoC are explained. In this thesis, we contribute to the architecture and implement a bidirectional NoC router for the multi-synchronous GALS infrastructure.

Key technical contributions of this thesis include the following:

1. A novel multi-synchronous bi-directional router architecture featuring dynamically self reconfigured bidirectional channels. It promises to enhance performance through better resource utilization.

2. A novel multi-synchronous FIFO buffer architecture that supports valid/ready flow control mechanism at all the different input/output ports of the routers.

3. A new channel directional control protocol, called Dynamic Channel Control, that intelligently and automatically determines the channel transmission direction using local information. This is based on handshaking protocol.

4. Implementation of the proposed NoC's router and detailed analysis in terms of area, delay, and power dissipation of each individual module.

2.3 Architectural Overview of Proposed NoC's Router

To enhance the performance of on-chip communication within the GALS infrastructure, we propose a Multi-Synchronous Bi-directional NoC architecture as shown in figure 2.2. This architecture is the extended version of an available BiNoC [63] architecture that is suitable for the GALS infrastructure. In this paper, we contribute a novel multi-synchronous FIFO buffer at all the interface (input/output ports) of the router and a dynamic channel controller that is based on the handshaking protocol. This dynamic channel controller controls the data

flow direction at each link and is implemented using a pair of ASM (Algorithmic State machine) charts namely Talker and Listener. It provides higher performance, deadlock-free, and starvation-free communications. Based on the motivational examples in BiNoC [63] and FSNoC [64], our proposed MBiNoC improves performance by providing more flexibility in using bandwidth. Here bi-directional channels are used to dynamically increase the bandwidth at runtime and new input buffer architecture is introduced to support intra-router data path within the GALS infrastructure. Normally in conventional NoC architecture, two adjacent routers use two unidirectional links in opposite directions for communication of data, but in our multi-synchronous bi-directional NoC, the data link between adjacent routers is able to transmit data in any direction dynamically.

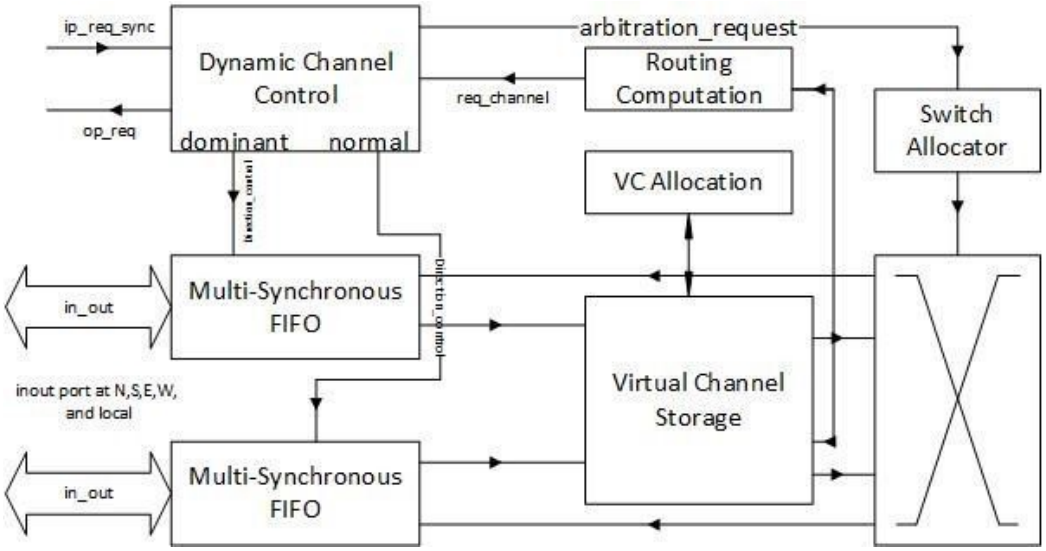


FIGURE 2.2: Detailed Architecture of proposed NoC Router

After contributing to the traditional architecture, the proposed architecture is called as Multi-synchronous BiNoC (MBiNoC) architecture. It mainly consists of three parts: multi-synchronous FIFO module at the interface of the router, dynamic channel control module and router internal architecture. Like a traditional architecture, MBiNoC router with virtual channel flow control has five pipeline stages: Route Computation, Virtual Channel Allocation, Switch Allocation, Switch Traversal, and Link Traversal.

2.3.1 Reconfigurable Inputs/outputs

Figure 2.3 shows the details of the proposed communication scheme between two neighboring routers. In order to support different clock frequencies for GALS infrastructure, we replace the synchronous FIFO of all the input ports by a novel multi-synchronous FIFO.

Details about the multi-synchronous FIFO and its implementation will be discussed in the chapter 3.
The detailed schematic of the bidirectional ports implementation of our proposed router is shown in figure 2.3. The *direction_control* signal is generated from the dynamic channel control and assigned properly to avoid conflict. Each port within the proposed architecture can be used as either an input port or an output port.

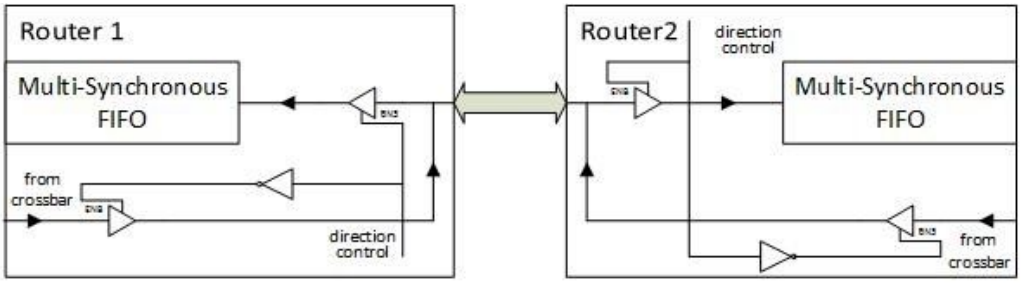


FIGURE 2.3: Bi-directional ports implementation

2.3.2 Dynamic Channel Control

Dynamic channel control is shown in figure 2.4 and is based on handshaking protocol. Here two data channels are available called data channel #1 and data channel #2. This controller performs mainly two tasks, first it decides the link direction dynamically and second it generates the arbitration request signal to the switch allocator.
Figure 2.4 shows the operation of dynamic channel control module. Here we consider that Talker ASM (algorithmic state machine) and Listener ASM work with different frequencies. So, signal *op_req_T* is generated from router #1 and reaches router #2 as *ip_req_syncL*. Signals *clk_T* and *clk_L* are the talker clock and listener clock respectively, and communication between two neighboring routers is performed using a two flip-flop synchronizer.
The detailed operation of dynamic channel control will be discussed in next section.

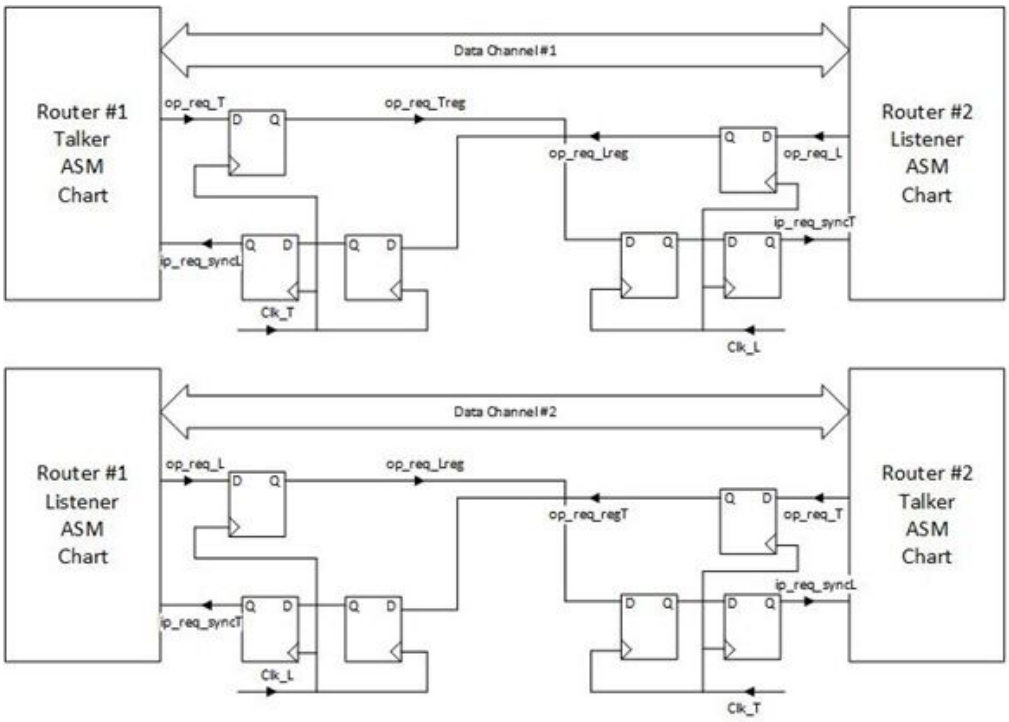


FIGURE 2.4: Dynamic channel control module

2.3.3 Virtual Channel (VC) and VC flow-Control

The Wormhole flow-control based conventional bidirectional router requires two input buffers to receive two packets from different directions at the same time. In the proposed router architecture, we provide the link utilization flexibility by sharing the access authority of two input multi-synchronous FIFOs for the two in-out ports at the same direction, where two input FIFOs can be multiplexed on two physical channels in each direction as shown in figure 2.3. Actually, virtual channels behave like an architecture which has numerous wormhole links existing in parallel. Therefore, this approach can possibly improve performance by decreasing the Head-of-Line Blocking effect of the channels. Since in the proposed router, two physical channels shared the virtual channels in each direction at a time, then total number of virtual channels is equivalent to the conventional bidirectional virtual channel router.
In Virtual Channel (VC) flow control [24], when the head flit of a packet arrives at a router, it must acquire one of the VCs associated with the physical channel that connects to its destination output before it can proceed. To achieve this, the head flit sends a request to the VC allocator once it reaches the front of its input VC. The VC allocator generates a matching between any such requests from the input VCs on the one hand and those output VCs that are not currently in use by another packet on the other hand.
In the general case, a router with P ports and V VCs per port therefore requires a VC allocator that can match PxV agents (all input VCs at all input ports) to PxV resources (all output VCs at all output ports) as shown in figure 2.5. The VC allocator architecture in our proposed router is the same as the available VC flow control based router except there is no overhead associated with this stage (VA stage), since in our router design, there are two bi-directional links in each direction among VC buffer and here we used multi-synchronous FIFOs for each virtual channel.

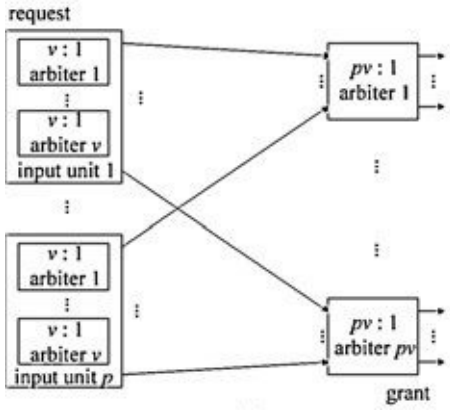


FIGURE 2.5: VC allocator in proposed router.

2.3.4 Switch Allocators

Once a packet has completed Virtual Channel (VC) allocation, its flits can be forwarded to the selected destination port subject to buffer space availability. In a conventional router architecture, for each flit to be transferred, a crossbar connection between the corresponding input and output ports must be established for one cycle. The switch allocator (shown in figure 2.6) is responsible for scheduling such crossbar connections; in particular, it generates matching between requests from active VCs at each of the router's P input ports on the one hand and crossbar connections to its P output ports on the other hand. In the proposed router architecture, since the channels are bidirectional links bandwidth is doubled from P to 2P in

each output direction. The grant signals generated by the switch allocator are used to set up the registers that control crossbar connectivity. In addition, the switch allocator notifies the winning VC at each input port, causing the latter to prepare its front most flit for crossbar traversal.

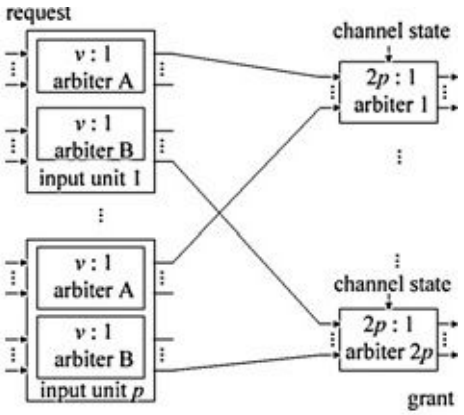


FIGURE 2.6: Switch allocator in proposed router.

2.4 Dynamic Channel Control Protocol

Our proposed dynamic channel control protocol is similar to channel direction control (CDC) protocol used in BiNOC [63], FSNoC [64], and double data rate protocol used in BiLink [65] except that all previously implemented protocols support only synchronous design whereas our proposed protocols support GALS infrastructure. Figure 2.4 shows the inter-router arrangement of dynamic channel control where a pair of ASMs namely Talker ASM and Listener ASM is supporting two unrelated clock domains and communicated on the principle of handshaking protocol.

2.4.1 Inter-router Transmission

The basic operation sequence of four-phase handshaking protocol is illustrated in figure 2.7. Initially The Talker ASM activates the *op_req_T* signal and this signal is passed through a register for unwanted glitches and then passed through two flip-flop synchronizer. In the *L_request1* state, Listener ASM senses activation of *op_req_T*, and *ip_req_sycT* signals, then it sends *ack* as*op_req_L* signal towards Talker. In the *T_request1* Talker detects activation of *op_req_L* as *ip_req_synL*, it sets the value *op_req_T* signal as zero.

Now listener deactivates the *op_req_L* signal after it notices deactivation of the *op_req_T* signal. Finally Talker comes back to the *idle* state once it detects deactivation of the *op_req_L* signal. Initially in *idle* state, the *op_req_T* signal is not activated and it moves to the *T_request1* state at the edge of *clk_T*, in this state, the *op_req_T* signal is set to be active. The finite state machine (FSM) then stays in the *T_request1* state until activation of Listener signal, *in_req_syncL*. Then it transfers to the *T_request0* state and *op_req_T* signal becomes zero. The FSM returns to the idle state after it senses deactivation of the *in_req_syncL* signal. The Talker FSM is similar to the Listener FSM except that it has one more idle state, thus can only respond to the Talker FSM. The *op_req_T* and *in_req_syncL* signals must be *glitch-free* because both signals are synchronized by a different clock domain.

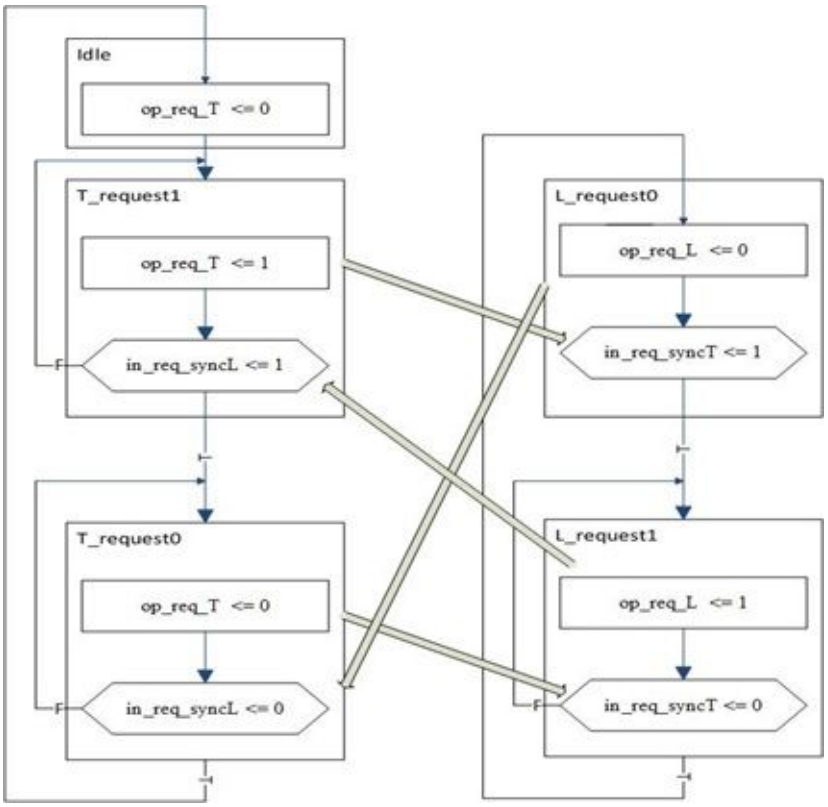


FIGURE 2.7: ASM chart for the Inter-router transmission.

2.4.2 Intra-router Transmission

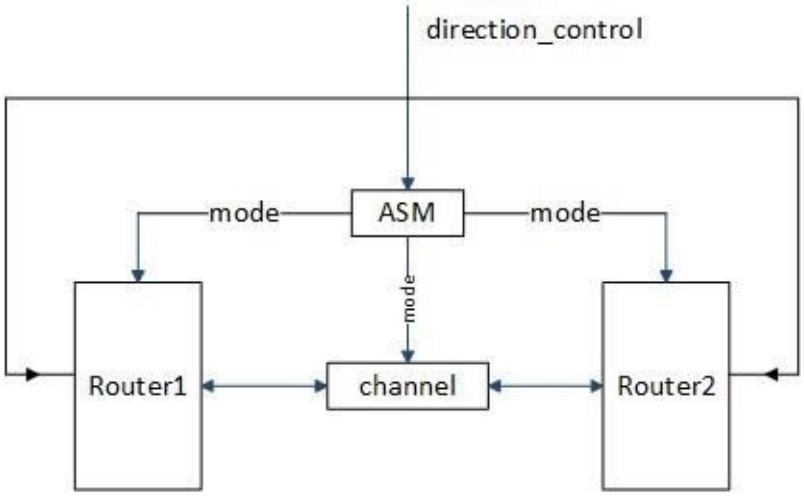


FIGURE 2.8: Mode Controller of ASM.

In our proposed router architecture, each bi-directional channel request to be achieved wisely to confirm that each transmission is valid unalike transmission channel used in conventional routers. To make sure that on each bi-directional link, only one direction is effective at one time, an inter-router communication channel control scheme is implemented for each channel in an algorithmic state machine as shown in figure 2.9. Within the Dynamic channel control module, algorithmic state machine dynamically reconfigures the route of each link. As shown in figure 2.9, our ASM design is quite simple but very efficient and it contains

mainly three states: *initial*, *free*, and *delay*. Here communication between two routers as from router1 to router2 (R1 → R2) and from router2 to router1 (R2 → R1) is shown in figure 2.9.
Here we developed a mode controller, which can set the mode of operation of ASM as priority mode or normal mode as shown in figure 2.8.
To impose symmetry of data transmission, the ASMs on the other end between the same couple of routers will be selected by opposite mode. The processes of neighboring ASMs are asynchronous to each other since they operate on different unrelated clock domains.

2.4.2.1 Priority Mode ASM

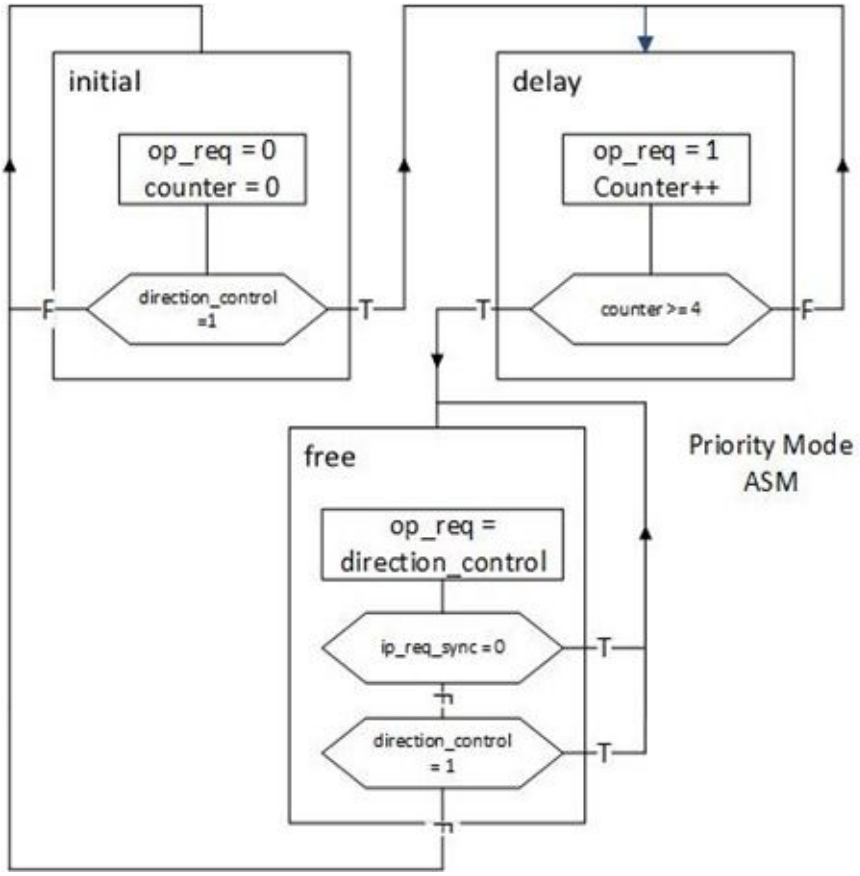


FIGURE 2.9: Dynamic channel control ASM in priority mode.

Dynamic channel control ASM in priority mode is originated at the *free* state as shown in figure 2.9. In this state if control signal *direction_control = 1* or input signal *in_req_sync = 0*, it will continue in this state. In other words, the link path will remain outward path as long as there are data flits within the current router to be sent via this link. The channel path should still remain unaffected even when there is no data to transmit, and if there is no request to send data from the neighboring router. When there are data to be sent from the neighboring router and if there are no data to transfer from the current router, on this condition the priority ASM will move from *free* state to *initial* state. The output signal was *op_reg = direction_control* in *free* state, subsequently, the output may get to be 0, permitting the neighboring router to ask for the channel. After entering in initial *state*, priority ASM will remain in this state as long as there are no data to transmit outward path, i.e. *direction_control = 0*. Meanwhile, in initial state, *op_req = 0* and also ASM initializes the counter as *counter = 0*. Whenever

the value of signal *direction_control* changes from 0 to 1 the priority ASM will move into *delay* state waiting to regain the channel control to transmit data. Within the *delay* state, the counter will increment its value during each clock cycle and *op_req = 1*. Whenever the value to *counter = 4*, the priority ASM returns to the *free* state and starts data transmission. Within *delay* state the output signal is *op_reg = 1* and has to reach the neighboring router in such a manner that the normal ASM will capture the link by initializing the *initial* state. This is the main reason for the *delay* state.

2.4.2.2 Normal Mode ASM

Dynamic channel control ASM in normal mode is originated at the initial state with *op_req = 0* as shown in figure 2.10. If a local route compute module requests to use the link *(direction_control = 1)* and the priority ASM of neighboring routers yields the channel *(ip_req_sync = 0)*, normal ASM will move from the *initial* state to the *delay* state. For eight clock cycles, the normal ASM continues in the *delay* state and it will return to an *initial* state whenever priority ASM requests the channel as *in_req_sync = 0*. Whenever *in_req_sync = 0* and after eight consecutive clock cycles, the normal ASM will move into the *free* state. If priority ASM does not have any data to communicate, i.e. *in_req_sync = 0*, the normal ASM could continue in the *free* state. Normal ASM stops immediately data transmission after *in_req_sync = 1* and returns back to the *initial* state.

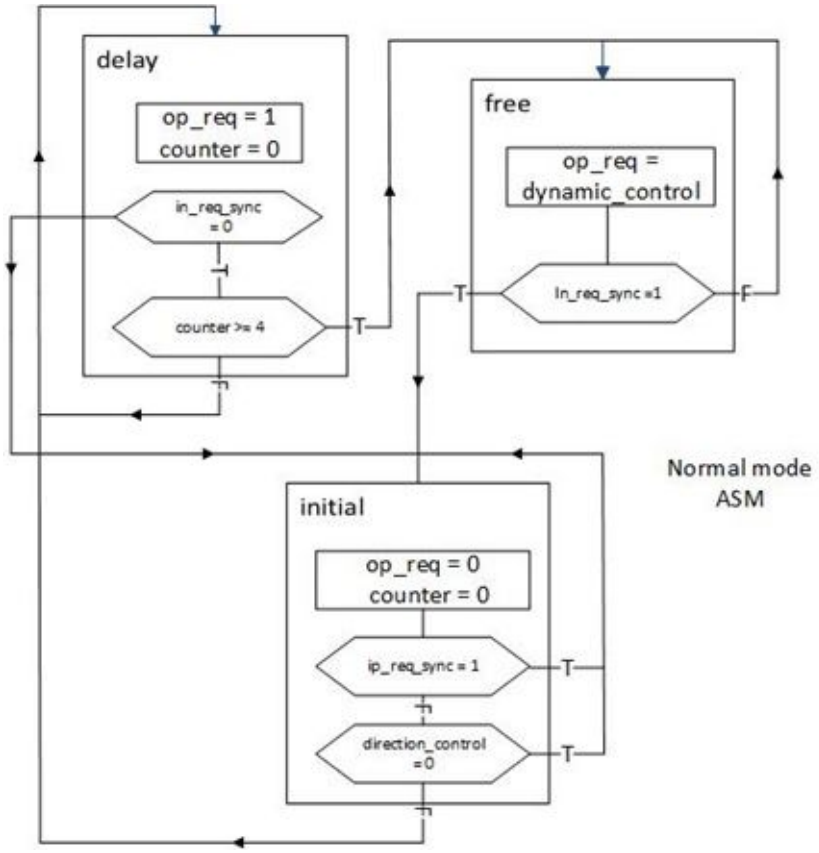


FIGURE 2.10: Dynamic channel control ASM in normal mode.

2.5 Conclusion

A Multi-clock Bi-directional channel Network-on-Chip (MBiNoC) architecture was proposed in this chapter to enhance the performance of on-chip communication for the GALS infrastructure. In a MBiNoC, each communication channel allows itself to be dynamically reconfigured to transmit flits in either direction. This added flexibility promises better bandwidth utilization, lower packet delivery latency, and higher packet consumption rate. Novel on-chip router architecture is developed to support dynamic self-reconfiguration of the bidirectional traffic flow. The flow direction at each channel was controlled by a new dynamic channel control protocol. Implemented with a pair of algorithmic state machines, this dynamic channel control protocol is shown to be high performance, free of deadlock, and free of starvation. A novel multi-synchronous first-in first-out (FIFO) buffer is implemented at the input/output ports to resolve GALS issue.
This chapter has described in detail the bidirectional channel control scheme and its design mechanism. An inter-router transmission scheme was provided first to achieve bidirectional data transmission. To avoid deadlock and starvation, a priority-based design of ASM was introduced.
This chapter mainly focused on contributions to the existing architecture and explained dynamic channel control protocol in detail.

Chapter 3

Multi-Synchronous FIFO

3.1 Summary

Microarchitectural structures of FIFO in Multi-Synchronous Bi-Directional NoC's routers have a noteworthy impact on the overall performance of an on-chip network for Globally Asynchronous Globally Synchronous (GALS) infrastructures. This buffering can be at the input or the output interface of a router to accommodate incoming flits and outgoing flits which cannot be directly forwarded due to traffic situations. In GALS infrastructure multi-synchronous FIFO is used at the interface of the routers, which supports dynamic, extendable and power efficient multi-clock architectures. This projected architecture of buffer allows the allocation of data amongst entirely separated clock domain modules with minimum cycles of latency between sender and receiver. The ready/valid handshake permits the sender and the receiver to prevent their operation for an arbitrary quantity of time. An abstract FIFO may be tailored to the ready/valid protocol both inside the upstream and the downstream connections.In this thesis we developed a novel multi-synchronous buffer architecture that supports valid/ready flow control mechanism at all the different interfaces of the routers. This proposed multi-synchronous buffer architecture is implemented using parametric Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology and its performance is evaluated in terms of power, area and delay.

3.2 Introduction

It is becoming more difficult and expensive to distribute a global clock without skew within a System-on-Chips (SoCs) and Chip-Multiprocessors (CMPs) due to shrinking technologies and design sizes. GALS systems provide a better alternative for the CMPs and SoCs [3]. NoC architectures are becoming the effective material for both general purpose chip-multiprocessors (CMPs) and application specific SoCs. The router microarchitecture plays a crucial function in accomplishing performance goals in the design of NoCs, i.e. low latency and high throughput. High throughput routers permit a NoC to fulfill the communication needs of multi-core SoC (MPSoC) applications.
An Multi-Synchronous FIFO Design refers to a FIFO Design where in the data values are written to the FIFO memory from one clock domain and the data values are read from a different clock domain, and the two clock domains are not related (in terms of frequency and phase) to each other. Multi-Synchronous FIFO's are widely used to safely pass the data from one clock domain to another clock domain
Buffering can be used at the input interface or the output interface of a router to accommodate incoming flits and outgoing flits, which cannot be directly forwarded due to traffic situations. In GALS infrastructure multi-synchronous FIFO is used at the interface of the routers to support multi-clock architectures. The presented architecture facilitates passing data between

different cores which are in a totally isolated clock island. It is particularly useful in applications where size of FIFO is important rather than latency which is critical such as in many NoC applications [76-78]. As it has been already discussed, Multi-Synchronous FIFO is used at the interface of the proposed router architecture, which supports dynamic, extendable and power efficient multi-clock architectures. It is very useful to transfer data between different clock domain modules. This projected architecture of buffer allows the allocation of data amongst entirely separated clock domain modules with minimum cycles of latency between sender and receiver.

An abstract form of Valid/Ready protocol can be built around a multi-synchronous buffer queue as shown in figure 3.1. This multi-synchronous buffer model has two interfaces called Push and Pop, which indicate to its connecting module when the buffer is empty or full. The AND gate outside the buffer queue controls the push and pop operation, i.e. push cannot be performed when signal full is asserted and pop cannot be performed when signal empty is asserted. The write (push) and read (pop) operations are performed on the write clock and read clock domain respectively. In both cases (write and read), it can be observed that a data transfer is occurring when the corresponding handshaking signal valid or ready is asserted.

Many bi-synchronous and asynchronous buffers are implemented by different authors in

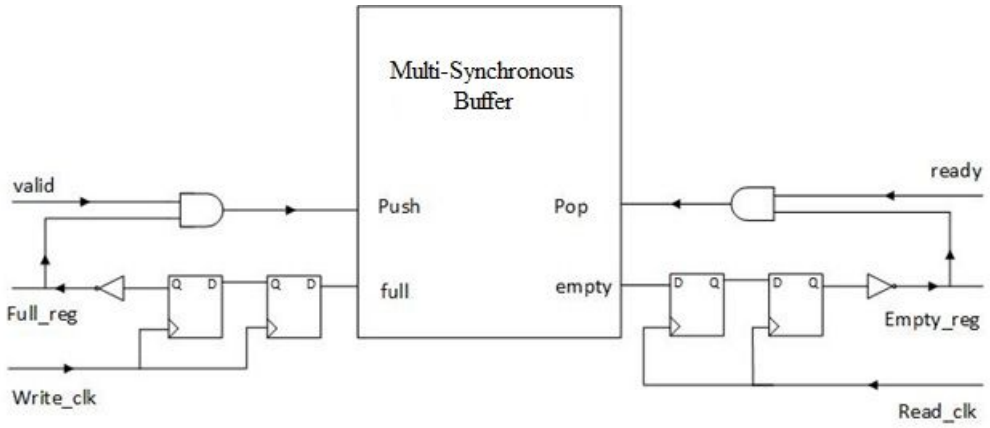


FIGURE 3.1: Valid/Ready protocol can be built around a multi-synchronous FIFO

literature for specific applications. Dally, Poulten, and Balch present top-level view of bi-synchronous FIFO architecture [79], but detail microarchitecture is not available in the literature. Ebergen [80] and Molnar et al. [81] often discuss fully asynchronous FIFO into the literature, but these designs do not utilize the clocks, that's why it is difficult to apply synchronization between different clock domains. Table 3.1 provide information about different bi-synchronous FIFO design. The FIFO designed by Greenstreet, which supports mesosynchronous system only (individual islands clock derived from global clock, which have same frequency but different phase like) [82]. Chakraborty et al. presented a buffer design; it first calculates time to develop a frequency difference estimate, before transferring the data [83]. A linear FIFO architecture is presented by Seizovic for data synchronization; it has some limitation in terms of initial latency [84]. Apperson et al. represented a scalable and robust bi-synchronous FIFO architecture but its memory size is fixed, that's why it does not provide high throughput and is not cost effective [85]. Similarly Panades and Greinear proposed a buffer architecture that is well-suited for GALS system but is not appropriate for mesosynchronous systems [86]. Chelcea and Nowick proposed an alternative FIFO architecture for the application of GALS infrastructure [87]. This design is based on a Register File and each register has its own full and empty flags. This style is suitable for small buffer only.

The contributions of this paper chapter summarized as follows:

TABLE 3.1: Comparison of various bisynchronous FIFOs

Bisynchronous FIFO Designs	*Buffer Types*	*Synchronization Types*
Greenstreet	Fixed size	Mesochronous
Chakraborty	Fixed size	—
Siezovic	Linear fixed size	Bisynchronous
Apperson	Fixed size SRAM	Bisynchronous & Pausable
Panades	Fixed memory array	Bisynchronous
Chelcea	Fixed Register file	Bisynchronous
Cummings	Fixed size SRAM	Bisynchronous
Implemented work	Register file	Multi-Synchronous

- A novel multi-synchronous buffer architecture that supports valid/ready flow control mechanism at all the different input/output interfaces of the Multi-synchronous bi-directional NoC's routers.
- We have implemented this router for different depth configuration i.e. 64, 128, 256, and 512.
- Detailed analysis in terms of area, delay, and power dissipation.

3.3 Synchronous FIFOs

A Synchronous FIFO describes the FIFO design where the data and information is stored in the memory and transition a data in a appropriate fashion using clock pulse. Both read and write operation handle by control circuit. In computer programming, FIFO (first-in, first-out) is an approach to handling program work requests from queues or stacks so that the oldest request is handled first. In hardware it is either an array of flops or Read/Write memory that store data given from one clock domain and on request supplies with the same data to other clock domain following the first in first out logic. Basically, FIFOs are divided in two categories, Synchronous and Asynchronous. Synchronous FIFOs have quickly become the FIFOs of choice for new designs. This movement to synchronous FIFOs from their asynchronous predecessors is due mainly to speed and ease of operation. However, there are also many other advantages which these devices bring such as synchronous flags, programmable almost empty and almost full flags, depth expansion, and retransmit. Synchronous FIFOs are easier to use at high speeds since they can be operated by free running clocks. Asynchronous FIFOs required read and write pulses to be generated as data is moved through the part, and generating these pulses is difficult to do at high speed. Synchronous FIFOs can be used just as their asynchronous counterparts by tying the read and write strobes to the *read clock* and *write clock* lines respectively. This makes migration to synchronous FIFOs very easy, even for designers who are mostly familiar with asynchronous FIFOs [88].
The basic building blocks of a synchronous FIFO are the memory array, flag logic, and expansion logic. The memory array is built from dual ported memory cells. These cells allow simultaneous access between both ports of the memory, the write port and the read port. This simultaneous access gives the FIFO its inherent synchronization property. There are no restrictions regarding timing or phase between accesses of the two ports. This means simply, that while one port is writing to the memory at one rate, the other port can be reading at another rate totally independent of one another.

3.3.1 Linear FIFO

This section discusses the fundamental principle and practices of basic synchronous FIFO structure called linear FIFO. The simplest form of FIFO consists of flip-flops which are connected like a serial input serial output shift register as shown in figure 3.2. Data is serially enters at the one end and propagates through every flip-flop until it reaches at the end of the register. Since all the movements of the data are controlled by a single clock it is called synchronous buffer.

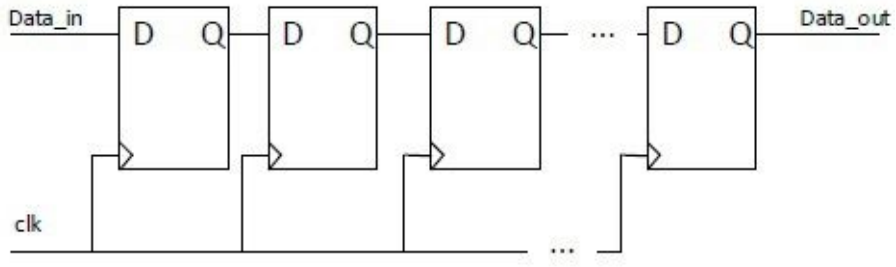


FIGURE 3.2: Linear FIFO block diagram

3.3.2 Elastic Bufffer FIFO

The ready/valid handshake allows the sender and the receiver to stop their operation for an arbitrary amount of time. Therefore, some form of buffering should be implemented in both sides to keep the available data that cannot be consumed during a stall in either side of the link.
The elastic buffer is the most primitive form of a register (or buffer) that implements the ready/valid handshake protocol. Elastic buffers can be attached to the sender and the receiver as shown in Figure 3.3. The EB at the sender implements a dual interface; it accepts (enqueues) new data from its internal logic and transfers (dequeues) the available data to the link, when the valid and ready signal are both equal to 1. The same holds for the EB at the receiver that enqueues new valid data when it is ready and drains the stored words to its internal logic [89]. The EB at the sender implements a dual interface; it accepts new data from

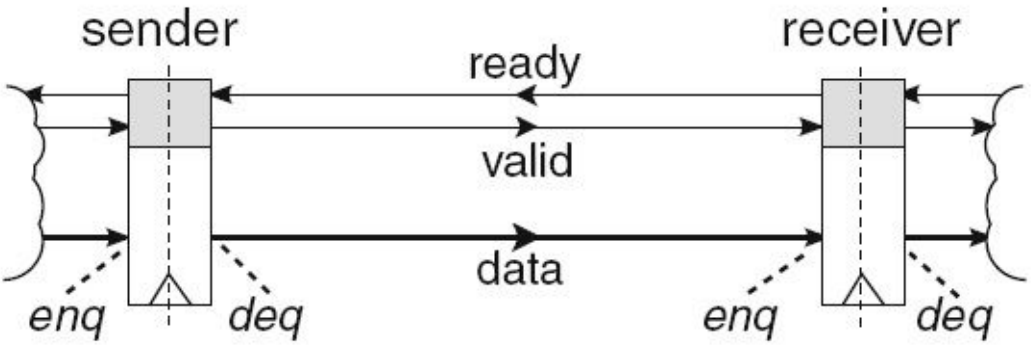


FIGURE 3.3: An elastic buffer attached at the sender and the receiver's interfaces [89]

its internal logic and transfers the available data to the link as shown in figure 3.4, when the valid and ready signals are both equal to logic high, so an EB can be built around a memory queue. An abstract memory queue provides a push and a pop interface and informs its connecting modules when it is full or empty [89]. The abstract memory model does not provide

any guarantees on how a push to a full queue or a pop from an empty queue is handled. The AND gates outside the buffer provide such protection. A push (write) is done when valid data are present at the input of the buffer and the buffer is not full.
At the read side, a pop (read) occurs when the upstream channel is ready to receive new data and the buffer is not empty, i.e., it has valid data to send. In both sides of the EB we can observe that a transfer to/from the buffer occurs, when the corresponding ready/valid signals are both asserted (as implemented by the AND gates in front of the push and pop interfaces).

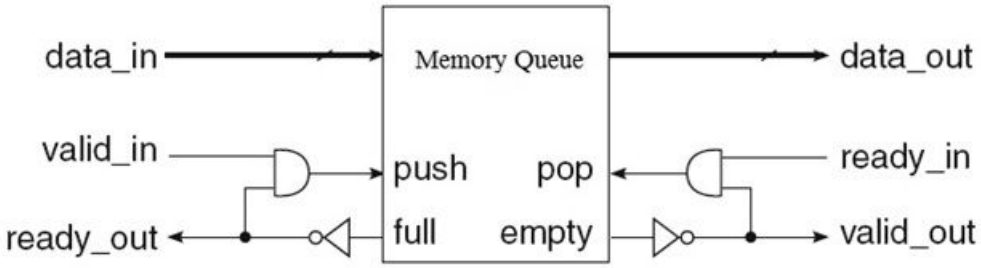


FIGURE 3.4: An elastic buffer built around an abstract memory queue model

3.3.3 Circular FIFO

A circular buffer or FIFO is a memory allocation scheme where memory is reused (reclaimed) when an index, incremented modulo the buffer size, writes over a previously used location.
A circular buffer makes a bounded queue when separate indices are used for inserting and removing data. The queue can be safely shared between threads (or processors) without further synchronization so long as one processor enqueues data and the other dequeues it. (Also, modifications to the read/write pointers must be atomic, and this is a non-blocking queue–an error is returned when trying to write to a full queue or read from an empty queue)[90].
Note that a circular buffer with n elements is usually used to implement a queue with n-1 elements–there is always one empty element in the buffer. Otherwise, it becomes difficult to distinguish between a full and empty queue–the read and write pointers would be identical in both cases.

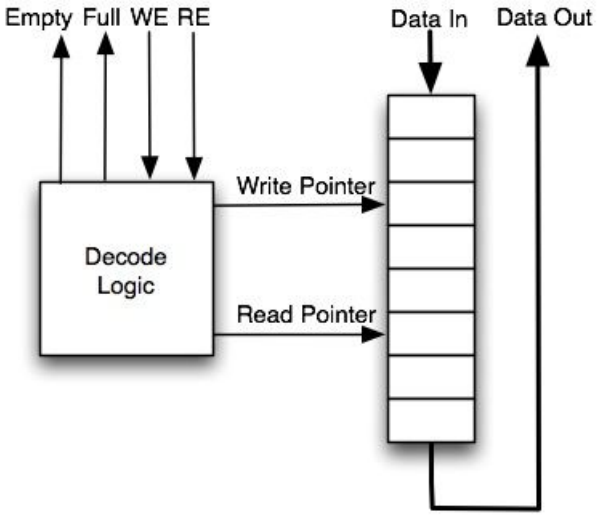


FIGURE 3.5: Block Diagram of Circular Buffer

As shown in figure 3.5, circular FIFO uses an array of arbitrary addressable memory elements supporting high throughput and low latency, and its scalability is radically improved due to the fact that data and clock signals are not affected by buffer size.It generates two control flags called full and empty flag for the valid data read and write into the buffer memory. Using read and write pointer alone to define the full and empty condition, always comparison of pointer must be taking place. For the empty condition the value of read pointer must be equal to the value of the write pointer. For the full condition, firstly increases the size of address pointers by one bit then equivalence tests of lower bits and Ex-or of MSB of address pointers. Following inequalities must be satisfied for the correct operation of the buffer.

$$rptr \leq wptr \leq rptr + N \tag{3.1}$$

Where *rptr* is read pointer, *wptr* is write pointer and *N* is the number of words.

3.4 Metastability and Synchronization

Metastability events are common in digital circuits, and synchronizers are a necessity to protect us from their fatal effects.Originally, synchronizers were required when reading an asynchronous input (that is, an input not synchronized with the clock so that it might change exactly when sampled). Now, with multiple clock domains on the same chip, synchronizers are required when on-chip data crosses the clock domain boundaries [91].
Understanding metastability and the correct design of synchronizers to prevent it is sometimes an art. Stories of malfunction and bad synchronizers are legion. Synchronizers cannot always be synthesized, they are hard to verify, and often what has been good in the past may be bad in the future. Consider the crosscut through a vicious miniature-golf trap in Figure 3.6. Hit the ball too lightly, and it remains where ball 1 is. Hit it too hard, and it reaches position 2. Can you make it stop and stay at the middle position? It is metastable, because even if your ball has landed and stopped there, the slightest disturbance (such as the wind) will make it fall to either side. And we cannot really tell to which side it will eventually fall.

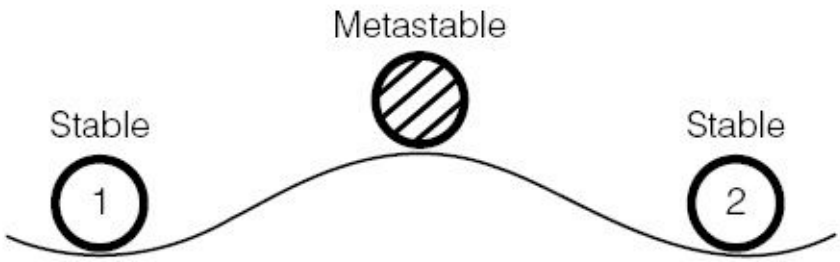


FIGURE 3.6: Mechanical metastability: the ball in the center position is metastable because the slightest disturbance will make it fall to either side.[17]

In flip-flops, metastability means indecision as to whether the output should be 0 or 1. Whenever flip-flop not receiving a stable input value near the positive or negative edge of the system clock, metastability is generated within the system. After resolution time T_{rol} (time required to reach a stable time) [90], metastability is resolved itself within the flip-flop and characterized by the *probability distribution function* (*PDF*) as

$$p(T_r) = e^{\frac{-T_r}{\tau}} \tag{3.2}$$

Where τ is the *time decay constant* and depend upon the electrical property of flip-flop.

The average time interval between two synchronization failures is known as *mean time*

TABLE 3.2: Sample of MTBF(T_r) computation

T_r *in ns*	*MTBF*
00.00	$3.99 \times 10^{-05}\ sec\ (0.039\ msec)$
20.50	$5.91 \times 10^{-03}\ sec\ (5.93\ msec)$
50.00	$8.82 \times 10^{-01}\ sec\ (0.84\ sec)$
70.50	$1.33 \times 10^{+02}\ sec\ (133\ sec)$
10.00	$1.95 \times 10^{+04}\ sec\ (5.41\ hours)$
12.50	$2.85 \times 10^{+06}\ sec\ (3.35\ days)$
15.00	$4.25 \times 10^{+08}\ sec\ (1.33\ years)$
20.00	$9.42 \times 10^{+012}\ sec\ (2.99 \times 10^{5}\ years)$
25.00	$2.07 \times 10^{+017}\ sec\ (6.58 \times 10^{9}\ years)$
30.00	$4.57 \times 10^{+021}\ sec\ (1.45 \times 10^{14}\ years)$
35.00	$1.01 \times 10^{+026}\ sec\ (3.19 \times 10^{18}\ years)$

between synchronization failures (MTBR) and it is expressed as a function of T_{rol}.

$$MTBR(T_r) = \frac{e^{\frac{-T_r}{\tau}}}{\omega \times f_{clock} \times f_{data}} \tag{3.3}$$

where ω is the *susceptible time window*, f_{clock} is the *system clock frequency*, and f_{data} is the *rate of change of input data*.

Synchronization is a method through which we can remove or reduce the probability of metastability. From table 3.2 it can be seen that increasing the value of T_{rol}, it increases the value of MTBR. Here consider f_{clock} is 50 MHz, data rate is $0.1 f_{clock}$, ω is $0.1ns$ and τ is $0.5ns$.

There are three types of synchronizer available called single-FF, Double-FFs and Triple-FFs synchronizer as shown in figure 3.7. They can remove metastability within a digital systems. In the diagram, synchronizer flip-flops provide the sufficient resolution time to move metatstable state to one of the stable state of input signal.

3.5 Multi-Synchronous FIFO Architecture

Let us have a small recap of bi-synchronous FIFO working and then we will go to proposed multi-synchronous FIFO design.

The general block diagram of bi-synchronous FIFO is shown in Figure 3.8. Functionality wise We can distinguish four functional blocks in this diagram. They are: dual port RAM, read pointer logic, write pointer logic and synchronizer. Dual port RAM has two ports-one is for reading and the other one is for writing operation. These two accesses of the FIFO are independent of each other and are completely controlled by read pointer logic and write pointer logic. Number of memory locations of the FIFO varies from 8 locations to some kilobytes. The data width of each location is also varying from one to 256 bits depending on the applications and technology. Modern day FIFOs provide options to program both of the above parameters as per requirements.

Data is written sequentially into the FIFO and read sequentially such that the first data written is the first data read out and so on with the remaining sequential data. Thus architecture of FIFO is completely characterized by these two independent operations. Dual port RAM and read-write logic circuits with synchronizers accomplish this task. Read port has its associated

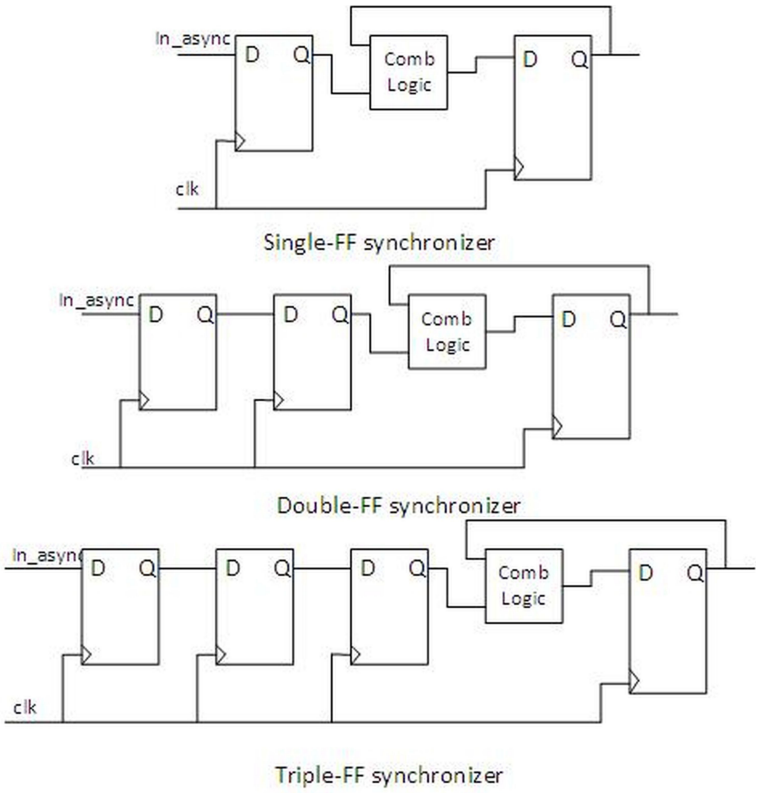


FIGURE 3.7: All type of available synchronizer [90]

memory addressing logic called as 'read pointer' logic and write port has 'write pointer' logic. When FIFO is reset both read and write pointers point to first memory location of the FIFO. As and when data is written to FIFO write pointer gets incremented and points to next memory location. Similarly when read operation takes place read pointer gets incremented for every read. Both pointer works in circular fashion i.e. after reaching the last position it will jump to first location of the FIFO.
'Full flag' and 'empty flag' are used to detect the status of the FIFO. These two flags are generated depending on the comparison result of FIFO pointers. Full flag is asserted when FIFO is completely full. Empty flag is asserted when FIFO is empty. Assertion of full flag indicates that no data can be written further unless at least one data is read out of the FIFO. Assertion of empty flag indicates the condition that no more data can be read from the FIFO unless until at least one data is written to the FIFO.
Even after the assertion of full flag, if data is written to FIFO 'overflow' condition occurs. Similarly after the assertion of empty flag if read operation is performed then 'underflow' occurs. Either overflow or underflow condition causes the data corruption or data loss. Safe and reliable FIFO designs always avoid both extreme conditions.

3.5.1 Proposed Architecture

Figure 3.9 describes the proposed architecture of Multi-Synchronous FIFO buffer architecture model, which supports data transfer between two different arbitrary clock domains as well as *ready/valid* protocol. The sender clock (*wclk*) and receiver clock (*rclk*) are not related to each other in terms of their phase and frequency. This block diagram consists of mainly five sub modules: Register File, Ready/Full generation block, Valid/Empty generation block, and two flip-flop synchronizer. On the left hand side of figure 3.9 write logic is presented and similarly in the right hand side of figure 3.9 read logic is presented.

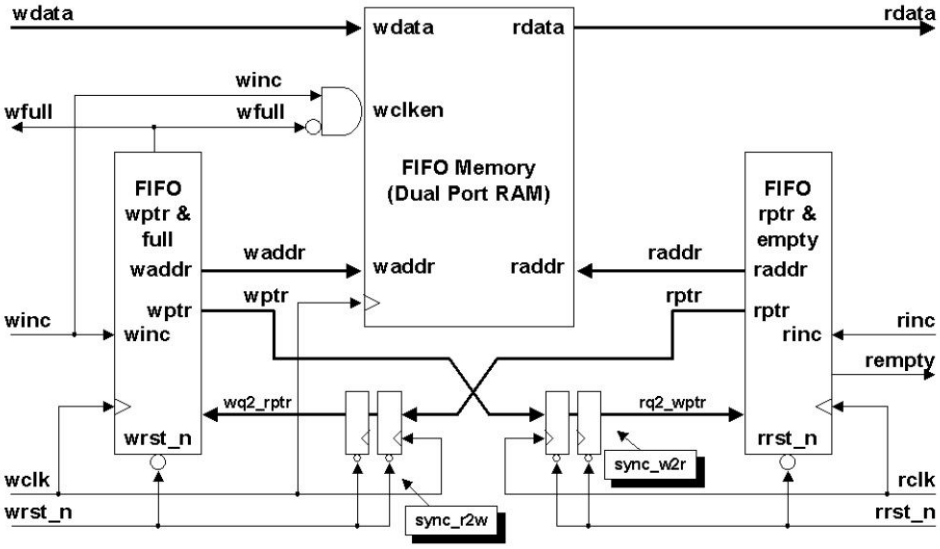


FIGURE 3.8: General approach of bi-synchronous FIFO design

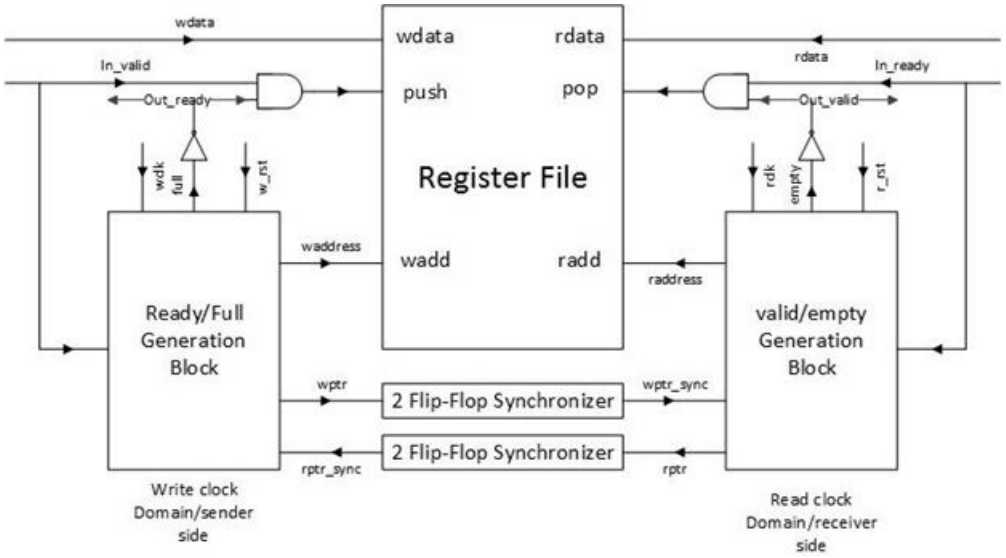


FIGURE 3.9: Proposed architecture of Multi-Synchronous FIFO buffer

3.5.1.1 Address Pointers and Gray Coding

The suggested architecture uses two pointers called *read* and *write* address pointers to track utilization of the buffer memory. In order to understand FIFO design, one needs to understand how the FIFO pointers work. The write pointer always points to the next word to be written; therefore, on reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.
Similarly, the read pointer always points to the current FIFO word to be read. Again on reset, both pointers are reset to zero, the FIFO is empty and the read pointer is pointing to invalid data (because the FIFO is empty and the empty flag is asserted). As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the

read pointer that is still addressing the contents of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic. The fact that the read pointer is always pointing to the next FIFO word to be read means that the receiver logic does not have to use two clock periods to read the data word. If the receiver first had to increment the read pointer before reading a FIFO data word, the receiver would clock once to output the data word from the FIFO, and clock a second time to capture the data word into the receiver. That would be needlessly inefficient.
The FIFO is empty when the read and write pointers are both equal. This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, having read the last word from the FIFO.
A FIFO is full when the pointers are again equal, that is, when the write pointer has wrapped around and caught up to the read pointer. This is a problem. The FIFO is either empty or full when the pointers are equal, but which?
One design technique used to distinguish between full and empty is to add an extra bit to each pointer. When the write pointer increments past the final FIFO address, the write pointer will increment the unused MSB while setting the rest of the bits back to zero as shown in Figure 3.10 (the FIFO has wrapped and toggled the pointer MSB). The same is done with the read pointer. If the MSBs of the two pointers are different, it means that the write pointer has wrapped one more time that the read pointer. If the MSBs of the two pointers are the same, it means that both pointers have wrapped the same number of times.

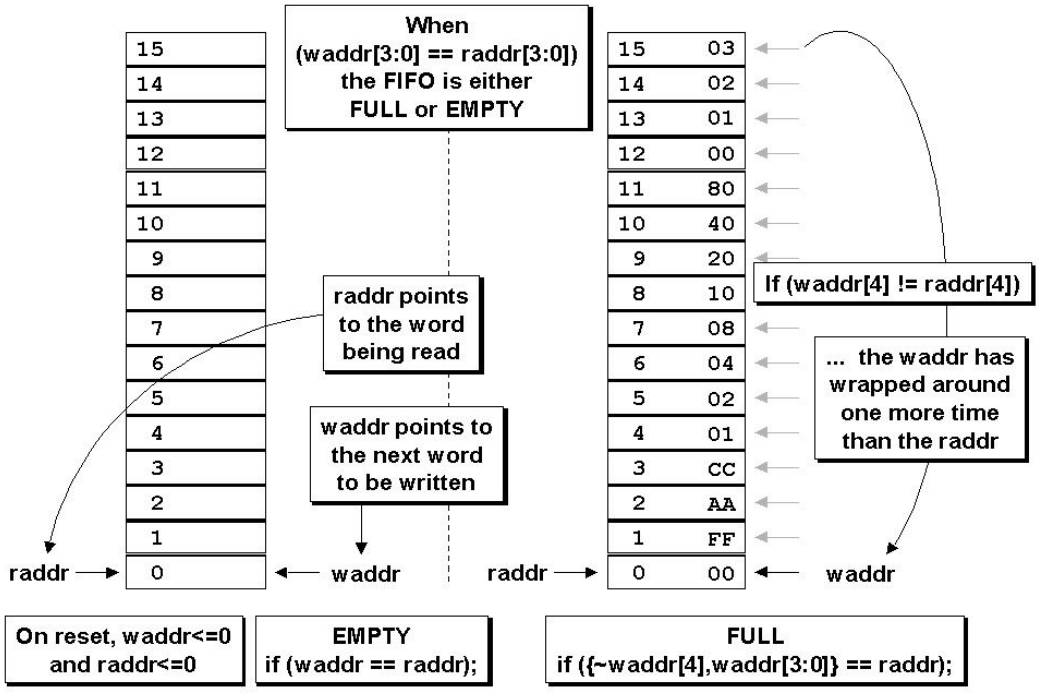


FIGURE 3.10: Proposed architecture's full and empty conditions

Using n-bit pointers where (n-1) is the number of address bits required to access the entire FIFO memory buffer, the FIFO is empty when both pointers, including the MSBs are equal. And the FIFO is full when both pointers, except the MSBs are equal.
Due to the metastability issues, the pointers are transferred to a Gray code format before the clock domain crossing as shown in figure 3.11.In the case of Binary pointers, trying to synchronize binary count value from one clock domain to another clock domain is challenging. Consider an example, when pointer value changes from 0111 to 1000, then all bits changed and increase the probability of metastability. In figure 3.11, Gray code counter assumes that

the outputs of the register bits are the Gray code value itself (ptr, either wptr or rptr). The Gray code outputs are then passed to a *Gray-to-binary converter* (bin), which is passed to a conditional binary-value incrementer to generate the *next-binary-count-value* (bnext), which is passed to a binary-to-Gray converter that generates the next-Gray-count-value (gnext), which is passed to the register inputs.
The implementation of *Binary-to-Gray* conversion and *Gray-to-Binary* conversion requires special circuit that is based on xoring operations. In the case of *Binary-to-Gray*, an *n-bit* binary vector (B_{n-1}, $B_{n-2}\ldots$ B_2, B_1, B_0) can be used to convert to *n-bit* gray coded vector (G_{n-1}, $G_{n-2}\ldots$ G_2, G_1, G_0) as shown in given equation 3.4, where $\oplus$ indicates the XOR function.

$$g_{n-1} = b_{n-1},\ g_{n-2} = b_{n-1} \oplus b_{n-2},\ g_{n-3} = b_{n-2} \oplus b_{n-3}$$

$$\ldots ; g_1 = b_2 \oplus b_1,\ g_0 = b_1 \oplus b_0 \tag{3.4}$$

Similarly in the case of *Binary-to-Gray*, an *n-bit* gray coded vector (G_{n-1}, $G_{n-2}\ldots$ G_2, G_1, G_0) can be used to convert to n-bit binary coded vector (B_{n-1}, $B_{n-2}\ldots$ B_2, B_1, B_0) as shown in given equation 3.5, where $\oplus$ indicates the same xor function.

$$b_{n-1} = g_{n-1},\ b_{n-2} = b_{n-1} \oplus g_{n-2},\ b_{n-3} = b_{n-2} \oplus g_{n-3},$$

$$\ldots ; b_1 = b_2 \oplus g_1,\ b_0 = b_1 \oplus g_0 \tag{3.5}$$

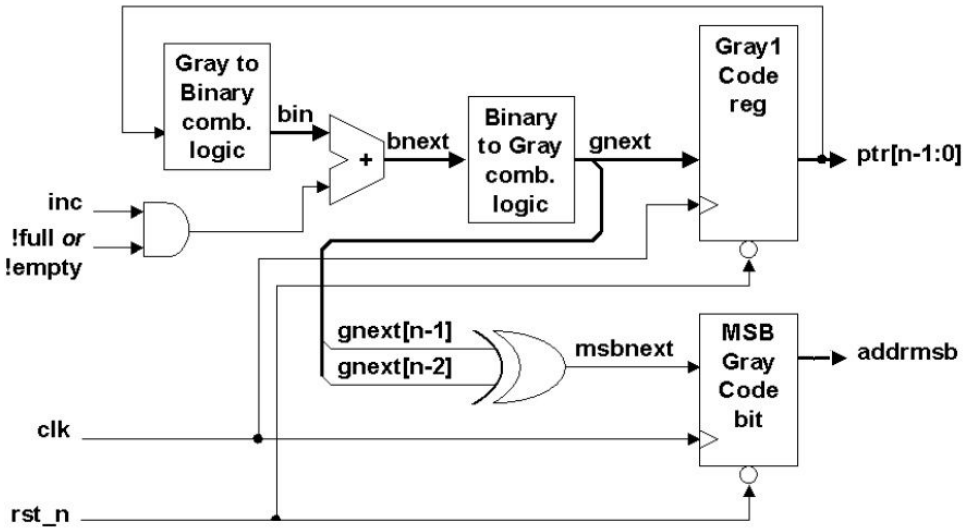


FIGURE 3.11: Dual n-bit Gray code counter block diagram

To better understand the problem of converting an n-bit Gray code to an (n-1)-bit Gray code, consider the example of creating a dual 4-bit and 3-bit Gray code counter as shown in Figure 3.12.
The most common Gray code, as shown in Figure 3.12, is a reflected code where the bits in any column except the MSB are symmetrical about the sequence mid-point[6]. This means that the second half of the 4-bit Gray code is a mirror image of the first half with the MSB inverted.
To convert a 4-bit to a 3-bit Gray code, we do not want the LSBs of the second half of the 4-bit sequence to be a mirror image of the LSBs of the first half, instead we want the LSBs of the second half to repeat the 4-bit LSBsequence of the first half.

Upon closer examination, it is obvious that inverting the second MSB of the second half of the 4-bit Gray code will produce the desired 3-bit Gray code sequence in the three LSBs of the 4-bit sequence. The only other problem is that the 3-bit Gray code with extra MSB is no longer a true Gray code because when the sequence changes from 7 (Gray 0100) to 8 (Gray 1000) and again from 15 (Gray 1100) to 0 (Gray 0000), two bits are changing instead of just one bit. A true Gray code only changes one bit between counts
Therefor we change the *gray-to-binary* counter sequences for comparison of their values.

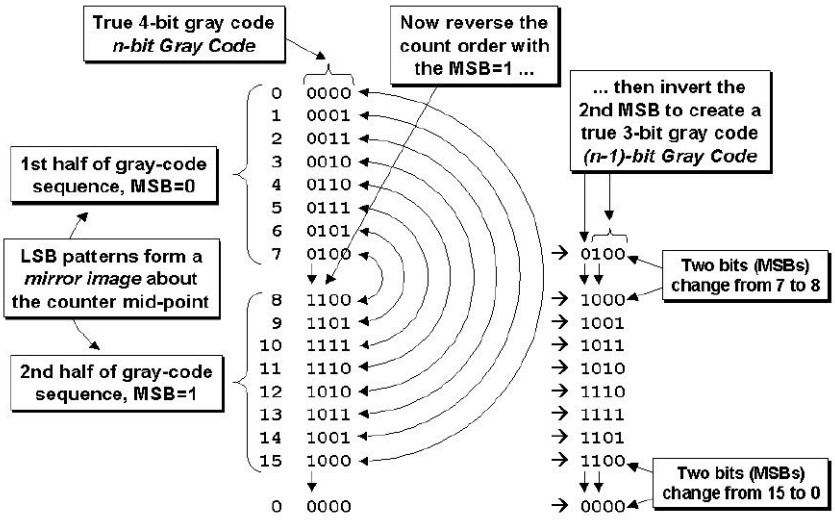


FIGURE 3.12: n-bit Gray code converted to an (n-1)-bit Gray code

3.5.1.2 Full generation block

Since the full flag is generated in the write-clock domain by running a comparison between the write and read pointers, one safe technique for doing FIFO design requires that the read pointer be synchronized into the write clock domain before doing pointer comparison.
Pointers that are one bit larger than needed to address the FIFO memory buffer are still used for the comparison, but simply using binary code counters with an extra bit to do the comparison is valid to determine the full condition.
Figure 3.13 shows the details about the left hand side of the main architecture of the multi-synchronous FIFO. On the write side, the FIFO indicates whether it is full or not. The sender should only send data when the FIFO is not full and asserting *in_valid* signal. From figure 3.9, when synchronized read pointer is equal to the write pointer except MSBs, the FIFO is full. The condition for the full flag generation in the write clock domain is shown in algorithm 1.

if *(wprt[n] == ¬ rptr_sync[n]) and (wptr[n-1]== rptr_sync[n-1])* **then**
    full = 1 and out_ready = 0;
else
    full = 0 and out_ready = 1;
end

Algorithm 1: Condition for full flag generation

3.5.1.3 Valid/Empty generation block

The empty comparison is simple to do. Pointers that are one bit larger than needed to address the FIFO memory buffer are used. If the extra bits of both pointers (the MSBs of the pointers) are equal, the pointers have wrapped the same number of times and if the rest of the read

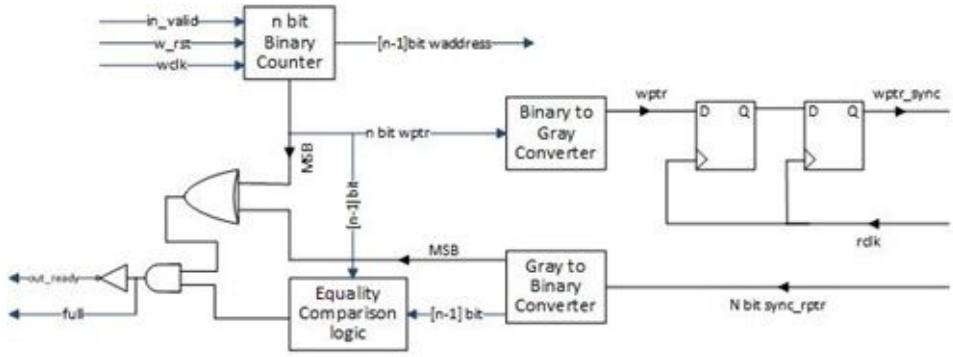


FIGURE 3.13: Detailed Architecture of Full/Ready generation module

pointer equals the synchronized write pointer, the FIFO is empty.
Similarly In this proposed multi-synchronous buffer architecture the empty flag will be produced in the right side or read clock domain. Whenever Register File is unoccupied, immediately the empty flag is generated. Figure 3.14 shows the details about the right hand side of the main architecture of the multi-synchronous FIFO. On the read side, the FIFO indicates whether it is empty or not, on the basis of in_ready signal, the receiver can consume all the data available within the FIFO memory. The condition for the valid/empty flag generation in the read clock domain is shown in algorithm 2.

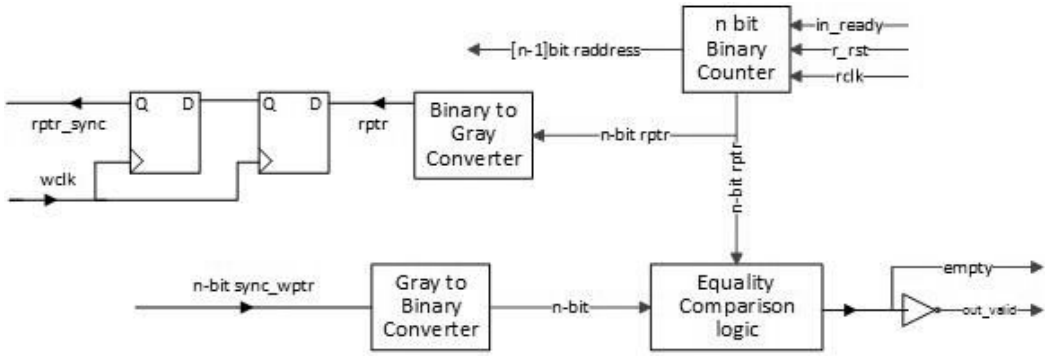


FIGURE 3.14: Internal Architecture of Valid/Empty generation module

if *(rprt == wptr_sync)* **then**
  empty = 1 and out_valid = 0;
else
  empty = 0 and out_valid = 1;
end

Algorithm 2: Condition for empty flag generation

3.6 Power, Area, and Delay Calculations

We have developed a parameterized RTL model using Verilog HDL and synthesized it in a commercial FPGA design flow. Synthesis is performed using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. Our proposed Buffer for Multi-synchronous bi-directional NoC's router is implemented for 4 different depth configuration i.e. 64, 128, 256, and 512. Here we present synthesis reports of each individual depth configuration which include macro statistics, power report and delay information. Power

analysis is performed by the Xpower analyzer and graphs are plotted by the Xpower estimator.
The throughput of the Multi-Synchronous FIFO is analyzed in function of the FIFO depth. As the synchronizers add latency, the flow control of the FIFO is penalized and its performances are influenced. In case of deep FIFO, those latencies do not decrease the FIFO throughput since the buffered data compensate the latency of the flow control.
Tables 3.3-3.6 show synthesis reports in term of power, area, and delay of implemented Multi-Synchronous FIFO RTL module for different depths i.e. 64, 128, 256, and 512.
Figures 3.15-3.18 show power analysis report of implemented Multi-Synchronous FIFO RTL module for different depths i.e. 64, 128, 256, and 512.

TABLE 3.3: Synthesis report of Multi-Synchronous FIFO (64 depth)

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
# RAMs : 1	2.298ns	3.692 W
64x8-bit dual-port distributed RAM : 1	(0.561ns logic, 1.737ns route)	
# Adders/Subtractors : 2	(24.4% logic, 75.6% route)	
7-bit adder : 2		
# Accumulators : 2		
7-bit up accumulator : 2		
# Registers : 44		
Flip-Flops : 44		
# Comparators : 2		
7-bit comparator equal : 2		
# Xors : 2		
7-bit xor2 : 2		

TABLE 3.4: Synthesis report of Multi-Synchronous FIFO (128 depth)

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
# RAMs : 1	2.211ns	3.703 W
128x8-bit dual-port distributed RAM : 1	(0.561ns logic, 1.650ns route)	
# Adders/Subtractors : 2	(25.4% logic, 74.6% route)	
8-bit adder : 2		
# Accumulators : 2		
8-bit up accumulator : 2		
# Registers : 50		
Flip-Flops : 50		
# Comparators : 2		
8-bit comparator equal : 2		
# Xors : 2		
8-bit xor2 : 2		

TABLE 3.5: Synthesis report of Multi-Synchronous FIFO (256 depth)

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
# RAMs : 1	2.40ns	3.732 W
256x8-bit dual-port distributed RAM : 1	(0.561ns logic, 1.839ns route)	
# Adders/Subtractors : 2	(23.4% logic, 76.6% route)	

Continued on next page

Table 3.5 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
9-bit adder : 2		
# Accumulators : 2		
9-bit up accumulator : 2		
# Registers : 56		
Flip-Flops : 56		
# Comparators : 2		
9-bit comparator equal : 2		
# Xors : 2		
9-bit xor2 : 2		

TABLE 3.6: Synthesis report of Multi-Synchronous FIFO (512 depth)

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
# RAMs : 1	2.549ns	3.761 W
512x8-bit dual-port distributed RAM : 1	(1.644ns logic, 0.905ns route)	
# Adders/Subtractors : 2	(64.5% logic, 35.5% route)	
10-bit adder : 2		
# Accumulators : 2		
10-bit up accumulator : 2		
# Registers : 62		
Flip-Flops : 62		
# Comparators : 2		
10-bit comparator equal : 2		
# Xors : 2		
10-bit xor2 : 2		

Synthesis reports include device macro statistics, timing parameters and total power dissipation. It can be seen that, all three parameters have a linear relation with the depth size of implemented multi-synchronous FIFO. Area (device macro statistics) is directly proportional to the depth of FIFO. Similarly, delay (timing parameters) and power dissipation are also directly proportional to the depth of implemented FIFO. Here it can be seen that if we doubled the depth size there are slightly increment in power, area and delay of the device.

To determine device power supply requirements and estimate thermal dissipation throughout the design process data exchange mechanisms are available between the different power estimation tools, Xilinx Power Estimator (XPE) and XPower Analyzer (XPA), which is in the ISE Design Suite (IDS).

In our design development process we first performed power estimation in XPE to size the voltage supply sources, evaluated thermal power dissipation paths, and allocated the total power budget to the different blocks in the FPGA system. Later in the development cycle we performed post implementation power analysis in XPower Analyzer to validate against our power and thermal goals.
Here power calculation is performed by Xpower analyzer and graph is plotted using Xpower estimator.

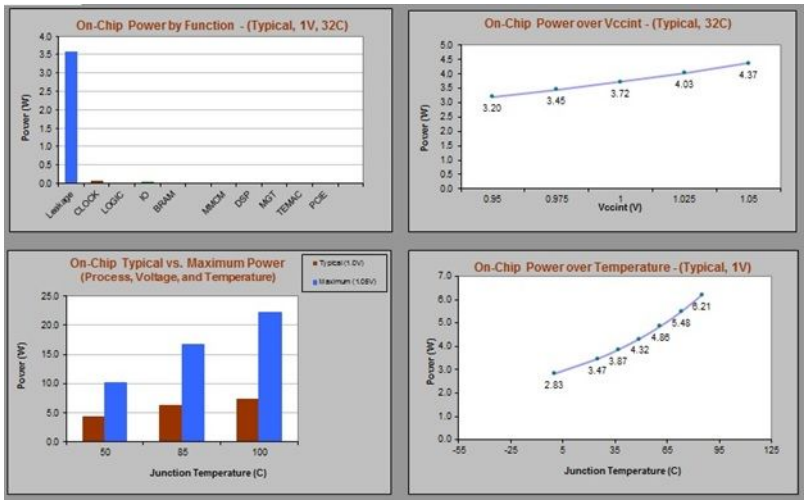


FIGURE 3.15: Power analysis report of Multi-Synchronous FIFO (depth 64)

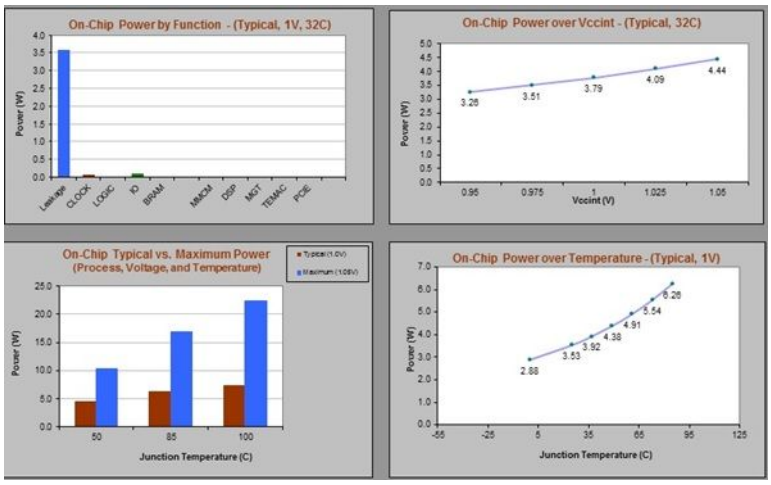


FIGURE 3.16: Power analysis report of Multi-Synchronous FIFO (depth 128)

3.7 Conclusion

This proposed multi-synchronous buffer architecture is implemented using parametric Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology.
The projected Multi-Synchronous FIFO design is well-matched for the many applications especially at the interface of two different clock domains. It can be utilized as a drop-in module at the router interface of the multi-synchronous network-on- chip. This design provides high sturdiness, fixed size register files, good energy proficiency, high frequency clock support and good scalability.

Multi-Synchronous FIFO design requires careful attention to details from pointer generation techniques to full and empty generation. Ignorance of important details will generally result in a design that is easily verified but is also wrong. Finding FIFO design errors typically requires simulation of a gate-level FIFO design with backannotation of actual delays and a whole lot of luck!

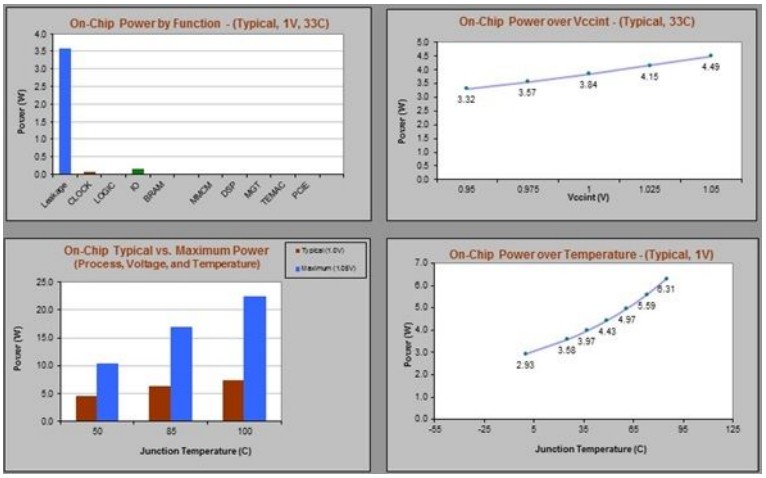


FIGURE 3.17: Power analysis report of Multi-Synchronous FIFO (depth 256)

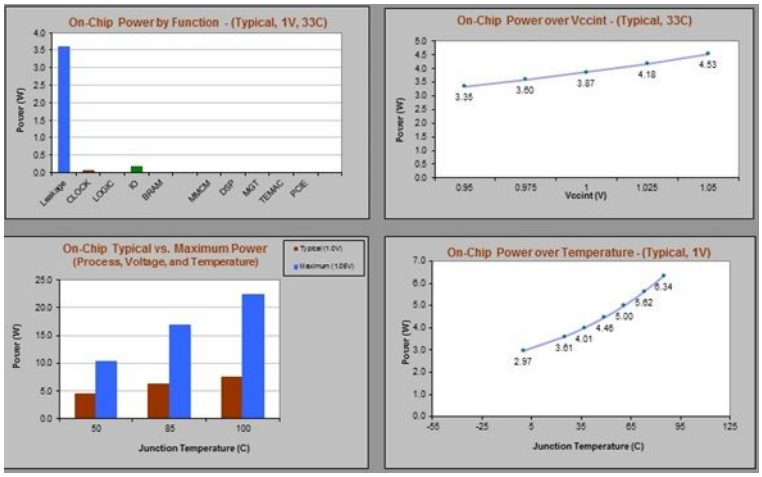


FIGURE 3.18: Power analysis report of Multi-Synchronous FIFO (depth 512)

Synchronization of FIFO pointers into the opposite clock domain is safely accomplished using Gray code pointers.
Generating the FIFO-full status is perhaps the hardest part of a FIFO design. Dual n-bit Gray code counters are valuable to synchronize and n-bit pointer into the opposite clock domain and to use an (n-1)-bit pointer to do “full” comparison.
Generating the FIFO-empty status is easily accomplished by comparing-equal the n-bit read pointer to the synchronized n-bit write pointer.
Careful partitioning of the FIFO modules along clock boundaries with all outputs registered can facilitate synthesis and static timing analysis within the two asynchronous clock domains.

Chapter 4

Arbiter Designs

4.1 Summary

Large systems-on-chip (SoCs) and chip multiprocessors (CMPs), incorporating tens to hundreds of cores, create a significant integration challenge. Interconnecting a huge amount of architectural modules in an efficient manner, calls for scalable solutions that would offer both high throughput and low-latency communication. The switches are the basic building blocks of such interconnection networks and their design critically affects the performance of the whole system. So far, innovation in switch design relied mostly in architecture-level solutions that took for granted the characteristics of the main building blocks of the switch, such as the buffers, the routing logic, the arbiters, the crossbar's multiplexers, and without any further modifications, tried to reorganize them in a more efficient way.
The need for efficient implementation of simple crossbar schedulers has increased in the recent years due to the advent of on-chip interconnection networks that require low latency message delivery. The core function of any crossbar scheduler is arbitration that resolves conflicting requests for the same output. Since, the delay of the arbiters directly determine the operation speed of the scheduler, the design of faster arbiters is of paramount importance. The core of each NoCs router involves arbiter and multiplier pairs that need to be carefully co-optimized in order to achieve an overall efficient implementation. Low transmission latency design is one of the most important parameters of NoC design.
In this chapter, we use parametric Verilog HDL to implement designs and compares performance in terms of power, area, and delay of different types of arbiters using for NoCs routers.The RTL implementation is performed used parametric Verilog HDL and analysis in terms of power, area and delay is performed using Xilinx ISE 14.7 and Xpower Analyzer (XPA) with Xpower Estimator (XPE).

4.2 Introduction

Arbitration is needed in any case that multiple contenders request access to a shared resource. This scenario appears in many forms in almost every computer system. The most common cases, where conflicting requests are resolved by arbiters, are the widely used bus-based systems where multiple masters and slave modules compete for gaining exclusive access to the bus, and the memory systems, where the small number of supported memory ports do not suffice to serve the read or write requests. Additionally, arbitration is the core function of network switching fabrics where packet flows arriving from different inputs need to be directed to the appropriate output.
The Network-on-Chip architecture is a data packet based communication system on a single chip [92]. Due to packet switching, NoC provides wider bandwidth and higher performance as compared with traditional bus structure[93]. Conventional interconnections like shared bus architectures provide lack of flexibility and extensibility in dealing with lager number of IP cores in a SoC.

Network-on-Chip are widely used in many complex System-on-Chips specifically in embedded system domain. Normally centralized switching techniques is preferred when number of cores are less in a SoC [3]. In the case of 2D Mesh, switches at each node contains routers and some of I/O ports where packet need to travel using different router and hops to reaching its final target on the network [93-95]. The most generic architecture of routers is shown in figure 4.1, for either using a centralized switching or a distributed switching. The NoC router is responsible for forwarding the incoming packets from the input buffers to output ports with proper arbitration and switch allocation techniques.

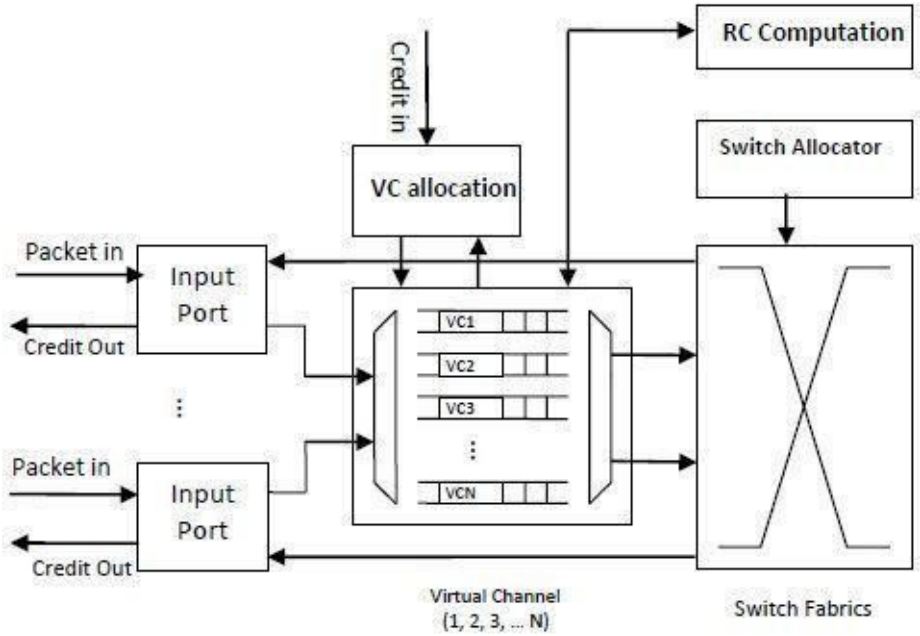


FIGURE 4.1: Generic Architecture of the NoC's Router

As shown in figure 4.1, Incoming packets are stored in input buffers and possibly in output buffers (not show in Figure) after crossing the crossbar. Which inputs are allowed to send their data over the crossbar are determined by the crossbar scheduler (or switch allocator) that resolves all conflicting requests for the same outputs. In many cases, in order to allow the sharing of the network's channels, to differentiate between separate traffic classes, i.e., request/reply packets, and to offer deadlock-free adaptive routing, virtual channels (VCs - or virtual lanes) are used [24]. The VC allocator similar to the crossbar scheduler is responsible for distributing the outputs' VCs to the requesting inputs. The complexity of the VC allocator increases with the number of available VCs per output link and the versatility of the routing logic [96]. The time needed to complete, either switch or VC allocation is critical to the performance of the switch and it determines the critical path of the design [95], [96-97].

Whenever incoming packets first arrive and are stored in the input FIFO after crossing the switch fabrics, the Switch allocator decides which inputs are allocated to send their data towards the switch fabrics by using chain of arbiters as shown in figure 4.2. This chain of arbiters also resolves all conflicting requests for the same output. Here virtual-channels (VCs) are used, since they offer deadlock free adaptive routing and are very useful in case of conflicting requests. Similar to the switch allocator, virtual channel allocator is responsible for distributive VCs, i.e. the output virtual lanes to the requesting inputs. Virtual-channel flow control, which associates several virtual channels (channel state and flit buffers) with a

single physical channel, and overcomes the blocking problems or confliction [98]. When a packet blocks virtual channel allow other packets to use the channel bandwidth that would otherwise be left idle . Virtual channel is a less expensive approach that gives almost the same performance as dividing the channels across multiple single-port buffers, and all even virtual channels are stored in one buffer and all odd channels in another one. This approach provides multiple buffers ports per physical channel and also switch input speed up is enabled without the expense of a per-virtual-channel buffer by providing multiple ports on a per-physical-channel buffers.

An efficient, fast, and scalable arbiter is one of most dominant factor for the high perfor-

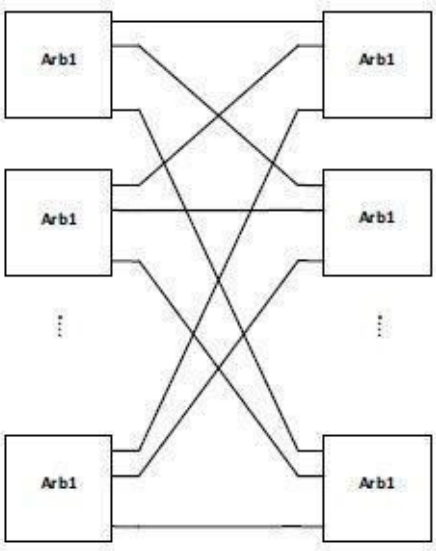


FIGURE 4.2: Chain of Arbiters

mance Network-on-Chips routers. Due to above explanation, it is very useful to analyze the different arbiters on the basis of area, speed, and power for the design of NoCs routers.

The core function of the scheduling operation is performed by the arbiter which grants only one of the incoming requests, serving first the request with the highest priority. To allow a fair allocation of resources and to achieve high performance switch operation, we should be able to change the priority of the arbiters. The way the priority changes is part of the policy employed by the crossbar scheduling algorithm. For example, the round-robin policy which is one of the most widely used priority update schemes, dictates that the request served in the current cycle gets the lowest priority for the next arbitration cycle.

Arbitration is access to a shared resource between multiple agents and it is one of the funda-

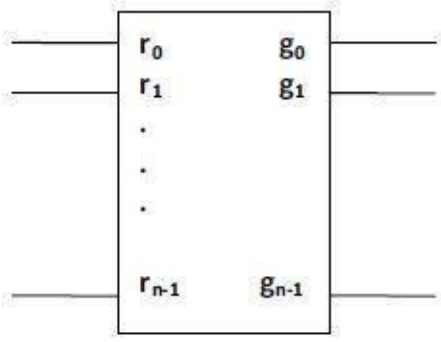


FIGURE 4.3: Block Diagram of Arbiters

mental operations performed by the control paths in a router of Network-on-Chips. Formally an arbiter accepts n request lines, r_0, r_1, r_3,...,r_n arbitrates among the asserted request lines, selecting one, r_i, for service, and asserting the corresponding grant line, g_i as shown in figure 4.3. The arbiter, not only resolves the conflict of the different contender for the same resource, but it also confirms that, resource is allocated to the contenders or not and grants it to the input which have highest priority[99].

4.3 Dynamic Priority Arbiter (DPA)

The organization of a generic Dynamic Priority Arbiter (DPA) is shown in Fig. 4.4 [100]. The DPA consists of two parts; the arbitration logic that decides which request to grant based on the current state of the priorities, and the priority update logic that decides, according to the current grant vector, which inputs to promote. The priority state associated with each input may be one or more bits, depending on the complexity of the priority selection policy. For example, a single priority bit per input suffices for round-robin policy, while for more complex weight-based policies such as first come first served (FCFS), multibit priority quantities are needed.

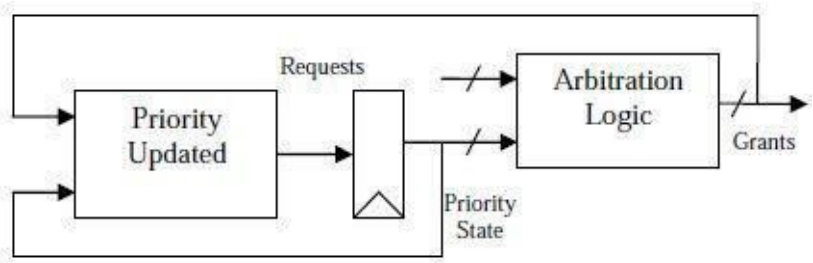


FIGURE 4.4: Block Diagram of Dynamic Priority Arbiter

DPA scans the input requests in a cyclic manner beginning from the position that has the highest priority. For example, if the i^{th} request has the highest priority then the priority is diminishing in a cyclic manner to positions *i + 1, i + 2, . . . ,n - 1,0,1, . . . i - 1,* giving to the request *i - 1* the lowest priority to win a grant.

4.4 Fixed Priority Arbiters

Priority encoder is the simplest form of the switch allocator also known as Fixed Priority Arbiters (FPA). We can express this logic in equation form as:

$$g_i = r_i \wedge c_i \tag{4.1}$$

$$c_{i+1} = \neg r_i \wedge c_i \tag{4.2}$$

Where,

$$g_i = i^{th}\ generated\ grant\ output\ of\ FPA,$$
$$r_i = i^{th}\ requested\ input\ of\ FPA,$$
$$c_i = i^{th}\ carry\ input\ of\ FPA,$$

and

$$r_{i+1} = (i+1)^{th}\ carry\ output\ of\ FPA.$$

This simplest form of arbiters grants access to a shared resource based on a predetermined priority order. If the request inputs are sorted in descending priority order, solving this problem is equivalent to finding the first set bit in a bit vector. If we assign priority in a linear order, we can construct an arbiter as an iterative circuit, as illustrated for the fixed-priority arbiter as a linear array of *bit cells*. Each bit cell *i*, as shown in Figure 4.5, accepts one request input, r_i , and one carry input, c_i , and generates a grant output, g_i , and a carry output, c_{i+1}. The carry input c_i indicates that the resource has not been granted to a higher priority request and, hence, is available for this *bit cell*. If the current request is true and the carry is true, the grant line is asserted and the carry output is deasserted, signaling that the resource has been granted and is no longer available. We expressed this logic in equations 4.1 and 4.2.

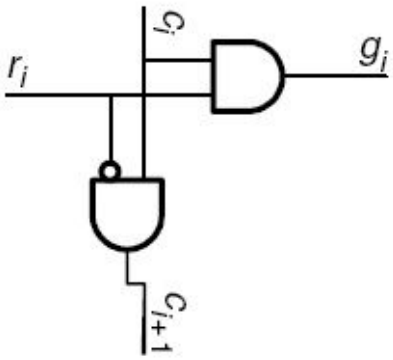


FIGURE 4.5: A bit cell for fixed priority arbiter

Figure 4.6 shows a four-bit fixed priority arbiter constructed in this manner. The arbiter consists of four of the *bit cells* of Figure 4.5. The first and last bit cells, however, have been simplified. The first bit cell takes advantage of the fact that c_0 is always 1. While the last bit cell takes advantage of the fact that there is no need to generate c_4.
Figure 4.6 shows a straightforward implementation using a linear array of basic bit cells, each of which generates a grant g_i if both its request input r_i and the incoming priority signal c_i are asserted. In addition, the incoming priority signal is propagated to the next cell only if r_i is not asserted. This design minimizes hardware complexity; however, its critical path delay scales linearly with the number of inputs.
If a large number of inputs must be supported, we can improve delay by taking advantage of the fact that the logic equations for the g_i and c_{i+1} outputs are structurally similar to those for a binary half adder's sum and carry outputs, respectively. As such, it is possible to transform the design shown in Figure 4.6 into an equivalent prefix network that hierarchically computes propagation conditions for the initial priority signal, causing the delay to scale logarithmically with the number of inputs.
Although useful for illustrating iterative construction, the arbiter of Figure 4.6 is not useful in practice because it is completely unfair. It is not even fair in the weak sense. If request r_0 is continuously asserted, none of the other requests will ever be served.

We can make a fair iterative arbiter by changing the priority from cycle to cycle, as illustrated in Figure 4.7. A one-hot priority signal p is used to select the highest priority request. One bit of p is set. The corresponding bit of r has high priority and priority decreases from that point cyclically around the circular carry chain.2 The logic equations for this arbiter are:

$$g_i = r_i \wedge (c_i \vee p_i) \tag{4.3}$$

$$c_{i+1} = \neg r_i \wedge (c_i \vee p_i) \tag{4.4}$$

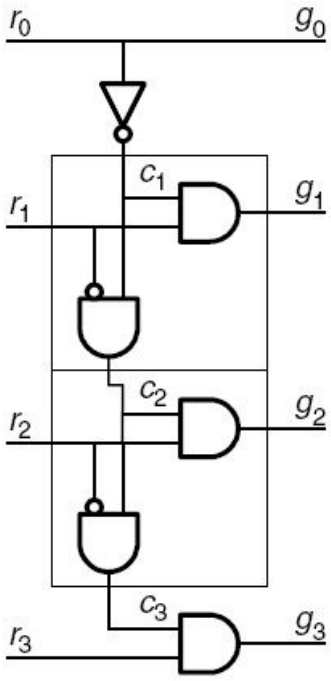


FIGURE 4.6: A four-bit fixed priority arbiter

$$c_0 = c_n \tag{4.5}$$

Both the rotating arbiter and the random arbiter have weak fairness. Eventually, each request

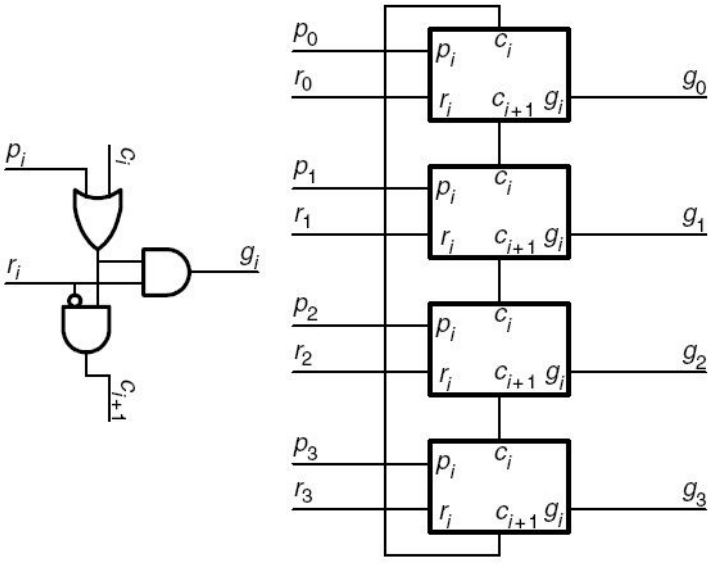


FIGURE 4.7: A four bit slice of a variable priority arbiter [12]

will become high priority and get service. These arbiters, however, do not provide strong fairness. Consider the case where two adjacent inputs r_i and r_{i+1} repeatedly request service from an n-input arbiter, while all other request inputs remain low. Request r_{i+1} wins the arbitration only when p_{i+1} is true while r_i wins the arbitration for the other n-1 possible priority inputs. Thus, r_i will win the arbitration n-1 times as often as r_{i+1}. This unfairness can be overcome by using a Round Robin arbiter.

4.5 Round Robin Arbiter

A round-robin arbiter operates on the principle that a request that was just served should have the lowest priority on the next round of arbitration [101]. This can be accomplished by generating the next priority vector p from the current grant vector g. In *Verilog HDL*, this logic is given by:

assign next_p = | g ? g[n-2]:0, g[n-1] : p;

The strong fairness is provided by the round-robin arbiters. Suppose any i_{th} request is served, its priority becomes the lowest and it will be served again after all other pending requests. There are several types of round robin arbiters (RRAs) proposed by different authors in the literature. Some of them are Baseline arbiters, Time speculative arbiters, Parallel prefix arbiters, Acyclic arbiters, Priority based arbiter, Weighted round robin etc. There are several methods available in the literature for designing fast round robin arbiter e.g. one of the methods is presented in McKeown et al. (1999)[102] and Dimitrakopoulos et al. (2008)[103].
Round robin arbiter operates on the principle that highest priority request becomes the lowest priority request on the next round of arbitration[101]. The Gate level architecture of round robin arbiter is shown in figure 4.8. In this arbiter, the priority select input is controlled by state register, which contains the most recent grant.We can design n-bit round-robin arbiter using bit cell 1 and bit cell 2, which are shown in figure 4.5 and figure 4.8.The bit cell 1 generates the grant signal and the bit cell 2 generates the priority signal. The round robin arbiters provide strong fairness.

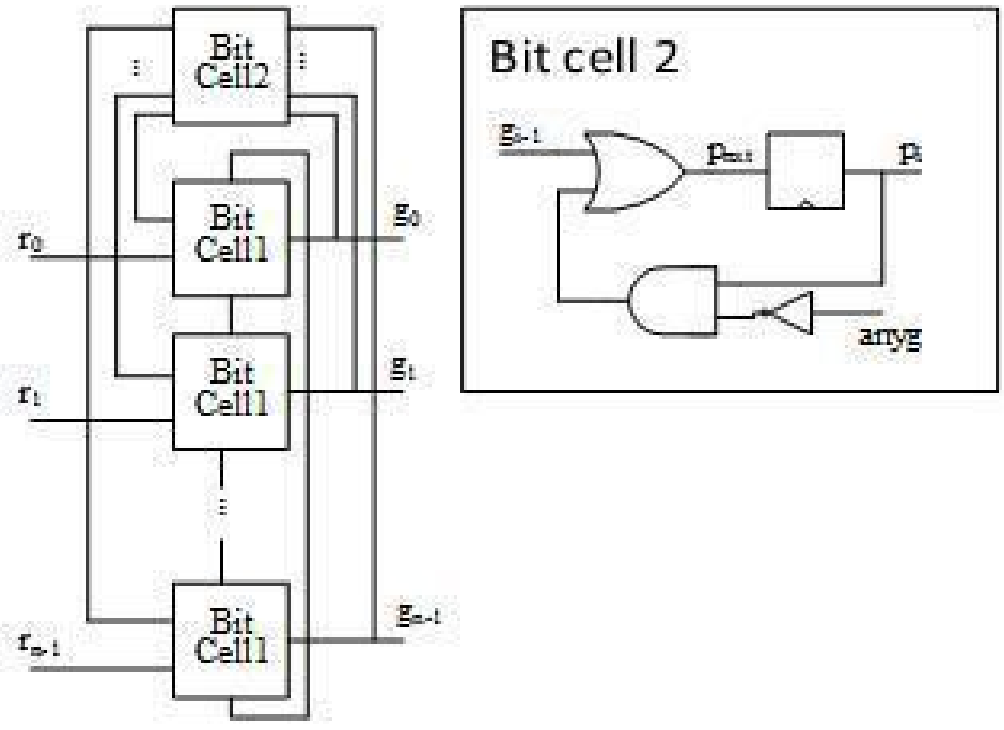


FIGURE 4.8: An n-bit round-robin arbiter

4.6 Matrix Arbiters

For the strong fairness, Matrix arbiters are one of the other alternatives. They work on the principle of the least-recently-served policy. A matrix arbiter wants to implements least-recently-served policy, it must maintain a triangular array of state bits P_{mn} for all m <n. The state bit P_{mn}, where *m* denotes the row position and *n* denotes the column position, indicates that request *m* takes priority over *n*. As we know that the diagonal elements are not needed and P_{mn} is not equal to P_{nm}, only upper triangular portion of the matrix need to be maintained.The matrix arbiter is also known as Hybrid First-Come First-Served and Least-recently used arbiters, proposed in [101]. It provides the benefits of both least-recently-used-priority and First-come First-served arbiters.

Figure 4.9 shows a matrix arbiter which has four inputs. In the upper triangular portion

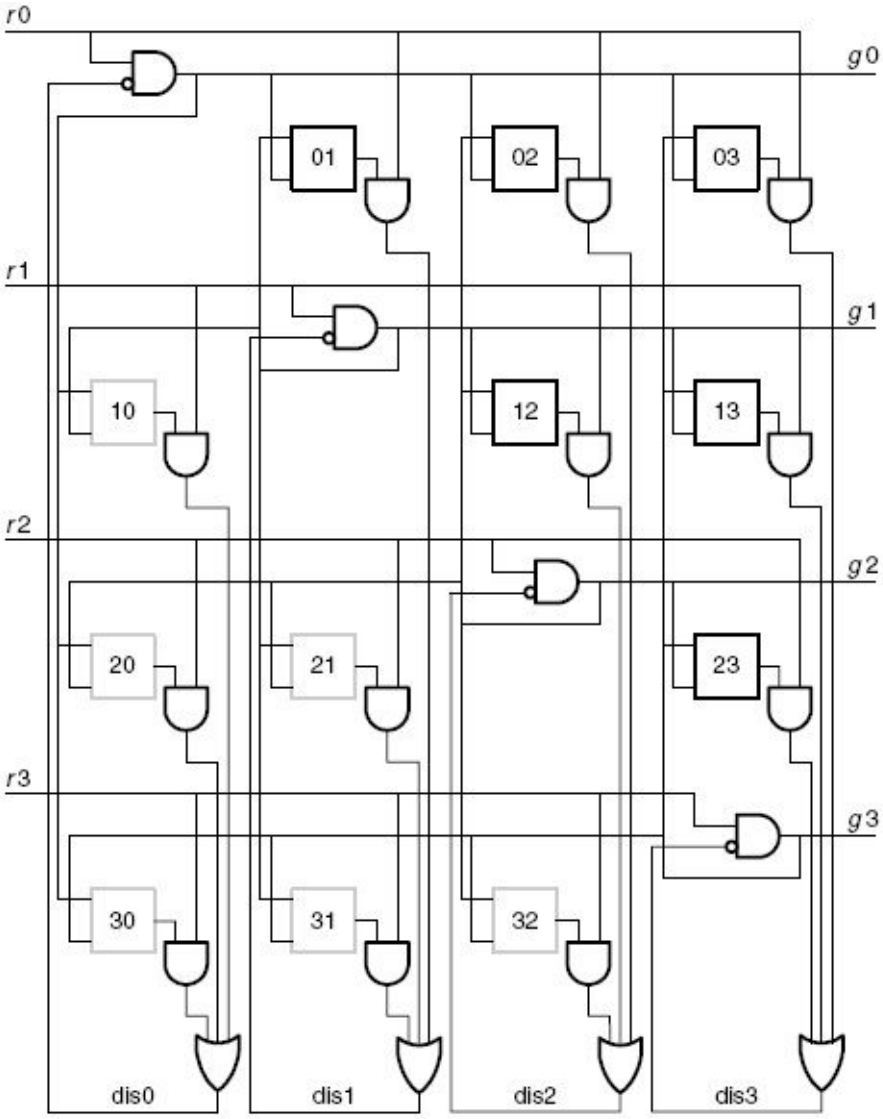


FIGURE 4.9: A 4-bit Matrix arbiter[12]

of matrix, solid boxes denoted the six flip flops which maintained the state and in the lower triangular portion of matrix, each of the shaded boxes represents the inverted output of the diagonally symmetric solid boxes.

A matrix arbiter maintains a triangular array of state bits P_{mn} that implement a least recently served order of priority. If P_{mn} is true, then request *m* will take priority over request *n*. When

a request z wins an arbitration, it clears bits P_z and sets bits P_z to make itself lowest priority. For better understanding, we can consider an example, suppose request r_0 is asserted and bit P_{02} is set then for disable the lower priority request 2, the signal 2 will be asserted as shown in Figure 4.9. Similarly a request travel to the corresponding grant output through a single *AND* gate if it is not disable[89].

4.7 Implementation and Analysis

This section we develop parameterized RTL implementations and synthesize them in a FPGA Virtex 6 family device XC6VLX760 using Xilinx ISE 14.7. We varied number of *requesters* in the design and analyze their performance on the basis of area, power, and delay.
Power analysis is performed by the Xilinx Xpower analyzer and graphs are plotted by the Xilinx Xpower estimator.

4.7.1 FPA Implementation and Analysis

In this subsection we analyze power, area, and delay of fixed priority arbiter for the 4 requesters. Table 1 shows all the statics of devices used , delays in term of nano second and power in term of watts. The implementation is performed using Verilog HDL and synthesis is performed using Xilinx 14.7.The functional simulation and Verification is performed by the Modelsim ISE 6.0d. For the 4 requesters FPA required only 2 1-bit register and the 2 2-bit 2-to-1 multiplexers with maximum delays of 2.782ns and total power dissipation is around 1.164 Watts.
Figure 4.10 shows the variation of power dissipation in all the terms. The power calculation is performed by Xpower analyzer and graph is plotted using Xpower estimator.

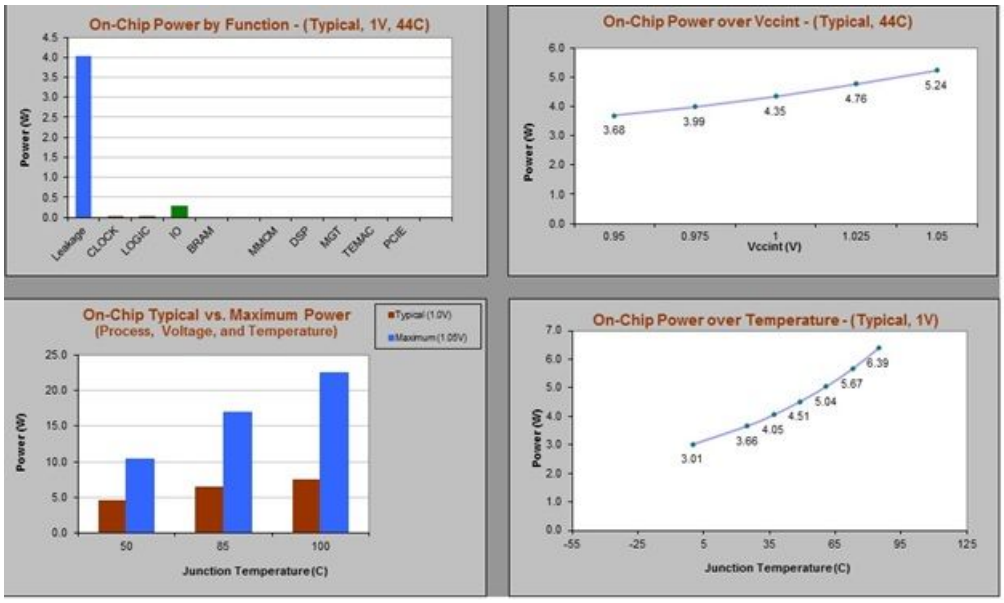


FIGURE 4.10: Power analysis of Fixed Priority Arbiter for *n* requester

TABLE 4.1: Synthesis report of FPA for 4 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :2	2.782ns	1.164 W

Continued on next page

Table 4.1 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
2-bit 2-to-1 multiplexer : 2	(1.127ns logic, 1.655ns route) (22.68% logic, 77.31% route)	

4.7.2 RRA Implementation and Analysis

In this paper round-robin arbiters are implemented using bcd coding and one-hot coding for different number of requester. Here number of requester are 2, 3, and 4. In this chapter it is also implemented using finite state machine and using fixed priority arbiters.

TABLE 4.2: Synthesis report of RRA using BCD encoding for 2 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :1	3.782ns	1.164 W
1-bit 2-to-1 multiplexer : 2	(1.088ns logic, 2.694ns route)	
	(28.6% logic, 71.23% route)	

TABLE 4.3: Synthesis report of RRA using BCD encoding for 3 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :2	4.14ns	1.166 W
2-bit 2-to-1 multiplexer : 4	(0.89ns logic, 3.25ns route)	
2-bit 4-to-1 multiplexer : 1	(21.5% logic, 78.5% route)	

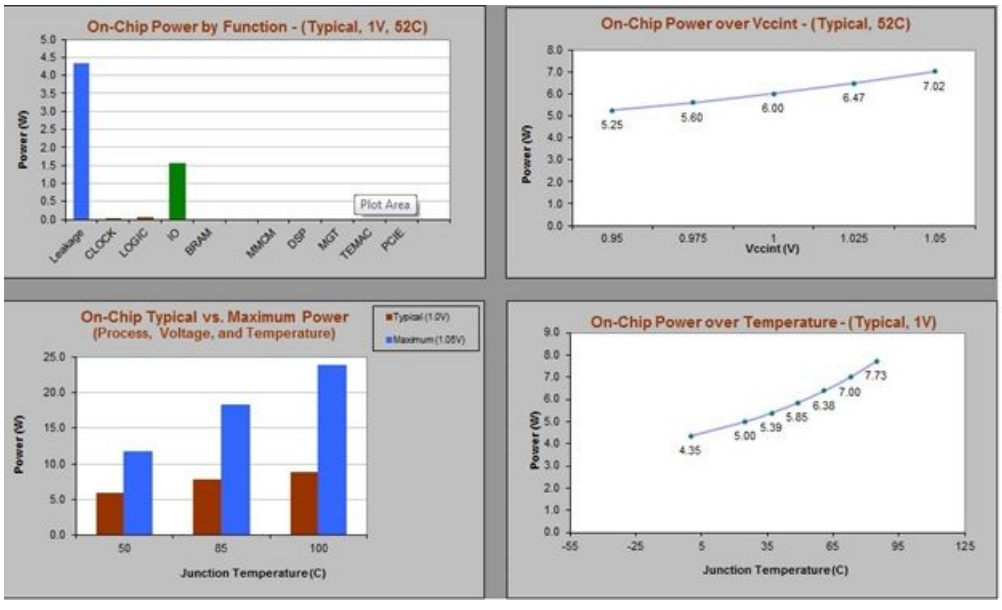


FIGURE 4.11: Power analysis of Round Robin Arbiter for *n* requester

TABLE 4.4: Synthesis report of RRA using BCD encoding for 4 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :2	4.16ns	1.167 W

Continued on next page

Table 4.4 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
2-bit 2-to-1 multiplexer : 9	(0.83ns logic, 3.33ns route)	
2-bit 4-to-1 multiplexer : 1	(19.95% logic, 80.5% route)	

TABLE 4.5: Synthesis report of RRA using ONE-HOT encoding for 2 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :2	2.782ns	1.164 W
2-bit 2-to-1 multiplexer : 2	(1.127ns logic, 1.655ns route)	
	(40.51% logic, 59.48% route)	

TABLE 4.6: Synthesis report of RRA using ONE-HOT encoding for 3 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :4	2.982ns	1.164 W
3-bit 4-to-1 multiplexer : 1	(1.127ns logic, 1.255ns route)	
3-bit 2-to-1 multiplexer : 7	(35.51% logic, 64.48% route)	

TABLE 4.7: Synthesis report of RRA using ONE-HOT encoding for 4 Requester

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :4	6.527ns	1.167 W
4-bit 2-to-1 multiplexer : 13	(1.135ns logic, 5.392ns route)	
4-bit 4-to-1 multiplexer : 1	(17.3% logic, 82.7% route)	

TABLE 4.8: Synthesis report of RRA using Finite State Machine

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :3	6.034ns	1.165 W
4-bit 2-to-1 multiplexer :12	(1.497ns logic, 4.537ns route)	
4-bit 4-to-1 multiplexer :1	(24.80% logic, 75.19% route)	

TABLE 4.9: Synthesis report of RRA using FPA

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1-bit registers :4	6.45ns	1.167 W
2-bit comparator greater : 3	(1.076ns logic, 5.374ns route)	
2-bit 2-to-1 multiplexer : 7	(16.68% logic, 83.317% route)	

Tables 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 4.8, and 4.9 shows the analysis report of round robin arbiter. According to different table data round robin is preferable than the fixed priority arbiter and its also provides the strong fairness. Like FPA, RRA is also implemented using

Verilog HDL and synthesis is performed using Xilinx 14.7. The verification and simulation is performed with Modelsim 6.0d.
Figure 4.11 shows the variation of power dissipation for RRA in all the terms. The power calculation is performed by Xpower analyzer and graph is plotted using Xpower estimator.

4.7.3 Matrix Arbiter Implementation and Analysis

In matrix arbitration when all input packets have the same priority request for same output port then matrix arbiter generates the matrix depending upon input and output port. In that matrix arbiter set the corresponding bit which is requested for same output port. Now matrix arbiter checks the priority if input *a* has highest priority and input e has lowest priority then matrix arbiter gives priority to input *a* and input *e* will get lowest priority. Matrix Arbiter generates *a* control signal so particular select line is selected and source packet is transmitted to destination.

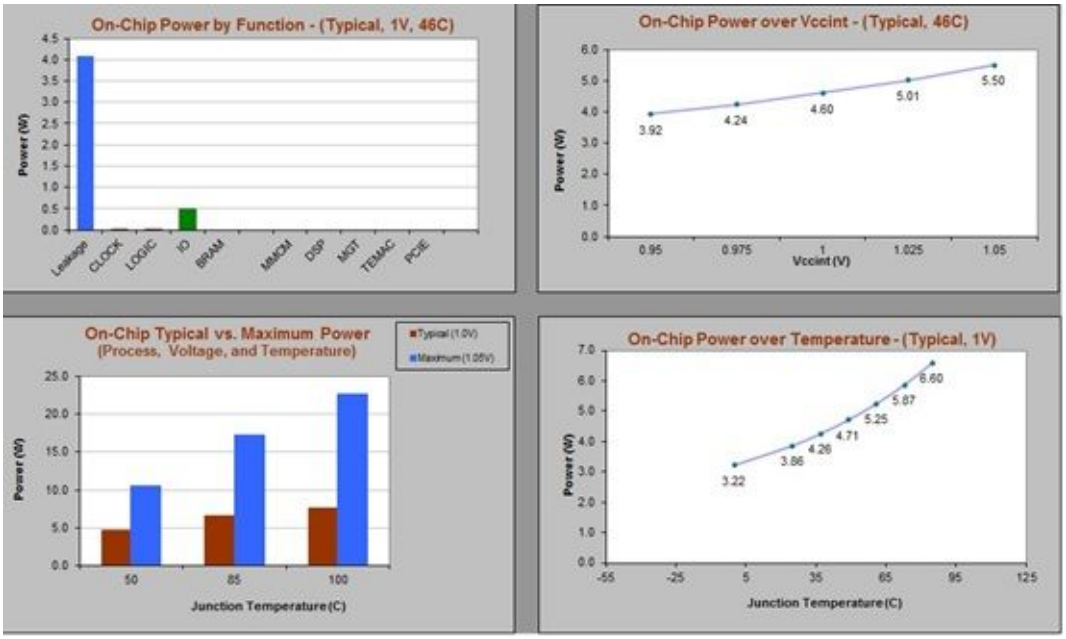


FIGURE 4.12: Power analysis of Matrix Arbiter for *n* requester

Matrix arbiter in one of most useful arbiter for the network on chips because it emulates the LULS (Last Used Last Served) notion of fairness. The reason it is able to do that and RRA is not is because we are now keeping additional state information and hence know how much in the past a particular input was served. Also another interesting thing to note is that we only need to maintain the triangular portion of the Matrix, which effectively means that we need to store only half the bits. This is a good improvement when we consider large switches with a large number of inputs.

TABLE 4.10: Synthesis report of Matrix Arbiter

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
4-bit 2-to-1 multiplexer : 12	6.15ns	1.167 W
2-bit comparator greater : 3	(1.076ns logic, 5.249ns route)	

Continued on next page

Table 4.10 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
4-bit 7-to-1 multiplexer : 1 FSM : 1	(20.68% logic, 79.317% route)	

Table 4.10 shows the analysis of Matrix arbiter in terms of power area and delays. The area and delays calculation is performed using Xilinx 14.7 ISE and the power calculation is performed using Xpower Analyzer and Xpower Estimator as shown in figure 4.12.

4.8 Conclusion

In this paper, we presented a detailed overview of arbitration and description of FPA, RRA and Matrix Arbiters implementation and analysis on the basis of power, area and delay.
A comparison of above discussed arbiter design shows that Fixed priority arbiter and matrix arbiter are slower and less efficient than the round robin arbiter. In this chapter, we have presented a brief overview of arbitration and provided detailed descriptions of representative arbiter implementations, which will used as building blocks for more complex structures in subsequent chapters. A comparison of arbiter implementations in terms of delay, area and power efficiency shows that contrary to popular wisdom matrix arbiters are both less efficient and slower than round-robin arbiters at sizes commonly encountered in NoC routers. Throughout the remainder of this dissertation, we will therefore consider the Round Robin arbiter exclusively.
As we know that VC allocation and crossbar scheduling determine the dynamic speed of the complete router, then we say that router designer may be truly profit by the use of the above discussed arbiters and their analysis.

Chapter 5

Routing Computation Module

5.1 Summary

Network-on-Chip (NoCs) provide a robust and scalable communication method for multi-processors system-on-chip (MPSoCs) architectures with increasing number of cores. Their performance is highly dependent on the throughput and latency properties of the micro-architecture of routers and its routing algorithms. Routing algorithm is one of the most important design choices for NoC implementation, as it controls the path decision that a flit has to follow while traveling along the network. In the case of look-ahead routing computation, each router pre-computes the preferred output ports based on its local blockage and transfers the preferred output ports to the adjacent routers.
Just like a computer network, a NoC network consists of devices that use the network, routers that direct the traffic between devices and wires that connect devices to routers and routers to other routers. In the network design of the NoC the most essential things are a network topology and a routing algorithm. Routers route the packets based on the algorithm that they use. There are many kinds of different algorithms for different systems to choose. Every system has its own requirements for the routing algorithm.
In this chapter we present RTL Implementation and analysis of Synchronous Look Ahead Routing Computation using XY, Adaptive XY, and Dimension-Order algorithms dedicated to 2D-Mesh Topology for a Distributed Scalable Predictable Interconnect Network (DSPIN). The RTL implementation is performed using Verilog HDL and analyzed in terms of power, area and delay using Xilinx ISE 14.7 and Xpower Analyzer (XPA) with Xpower Estimator (XPE). Here FPGA Virtex 6 family device XC6VLX760 is considered as target technology.

5.2 Introduction

Networks-on-Chip (NoC) is an emerging on-chip interconnection-centric platform that influences modern high speed communication infrastructure to improve the ever increasing on-chip communication challenges of recent many-core System-on-Chip (SoC) designs [103]. Continuing shrinkage of feature dimensions of nano-scale semiconductor devices has raised serious concerns of the reliability, signal integrity, and quality of services (QoS) of traditional bus-based on-chip interconnect infrastructure [104]. NoC represents a major standard move to address these concerns by incorporating state-of-the-art high-speed data network components (such as routers and switches) and packet-based routing protocols in novel on-chip network infrastructure [105]. A NoCs aim is to provide a reliable on-chip communication platform to facilitate scalable giga-scale SoC design [7]. Conventional interconnections based on shared bus lack flexibility and extensibility in dealing with large number of IP cores in a SoC. NoC is an on-chip communication network to addresses the challenges of designing SoC interconnections [13].
A NoC router is responsible for forwarding the incoming flits (large network packets are broken into small pieces called flits) from the input buffers, with its proper arbitration and

routing algorithms towards outputs ports [67]. The router makes a decision based on routing algorithm and topology of network and the decision may be centralized or distributed [68].
For designing of an efficient NoC micro-architecture, selection of an appropriate routing algorithm should be one of the important criteria [33]. Nowadays, popular routing algorithms are XY, Adaptive routing and balance dimension order routing due to the simplicity of implementation in hardware as well as its natural deadlock freedom. Balance dimension order routing can only provide high performance under random traffic due to the uniformly distributed traffic at all links. Actually network traffic is bursty and dynamic in nature [7]. Therefore, dimension order routing gives poor performance for unbalanced distributed of traffic on all links. Therefore, adaptive routing is a more preferred routing algorithm for the recent NoC based systems.
Routers of the NoC are responsible for keeping the network connected and resolving contention for the same resource while allowing multiple packets to flow in the network concurrently. Each router should respect the properties of the routing algorithm and forward the incoming packets to the appropriate output following the path decided by the routing algorithm [106]. Depending on the routing algorithm, each router should translate the destination address to a local output request allowing each packet to move closer to its destination. This translation is an obligation of the routing computation logic. The routing computation logic can be implemented using a simple lookup table or with simple turn-prohibiting logic [107]. Routing computation reads the destination address of the head flit of a packet and translates it to a local request following the rules of the network-wise routing algorithm and the address of the current router. In this way, request generation, arbitration and multiplexing should wait first routing to complete before being executed. This serial dependency can be removed if the head flit carries the output port request for the current router in parallel to the destination address. Allowing such behavior requires the head flit to compute the output port request before arriving at the current router using lookahead routing computation. Lookahead routing computation (LRC) was first employed in the SGI Spider switch [108], and extended to adaptive routing algorithms in Vaidya et al. (1999) [109].

5.2.1 Routing Algorithm Classifications

Routing algorithms can be classified in various ways, as shown in Figure 5.1. In unicast routing, the packets have a single destination, while in the case of multicast routing, the packets have multiple destinations. For on-chip communication, unicast routing strategies seem to be a practical approach due to the presence of point-to-point communication links among various components inside a chip. Based on the routing decision, unicast routing can be further classified into four classes: centralized routing, source routing, distributed routing and multiphase routing.
In centralized routing, a centralized controller controls the data flow in a system. In case of source routing, the routing decisions are taken at the point of data generation, while in distributed routing, the routing decisions are determined as the packets/flits flow through the network. The hybrid of the two schemes, source and destination routing, is called multiphase routing.
Routing algorithms can also be defined based on their implementation: lookup table and Finite State Machine (FSM). Lookup table routing algorithms are more popular in implementation. They are implemented in software, where a lookup table is stored in every node. We can change the routing algorithm by replacing the entries of the lookup table. FSM based routing algorithms may be implemented either in software or in hardware.
These routing algorithms may further be classified based on their adaptability. Deterministic

routing always follows a deterministic path on the network. Examples of such routing algorithms are XY routing, North first, South first, East first, and West first. Adaptive routing algorithms need more information about the network to avoid congested paths in the network. These routing algorithms are obviously more complex to implement, thus, are more expensive in area, cost and power consumption. Therefore, we must consider a right QoS (Quality-of-Service) metric before employing these algorithms.
Routing algorithms can be fault-tolerant algorithms such as backtracking. In case of progressive algorithms, a channel is reserved before a flit is forwarded. Some routing algorithms send packets/flits only in the direction that is nearer to the destination. These routing algorithms are referred as profitable algorithms. A misrouting algorithm may forward a packet/flit away from the destination as well. Based on the number of available routing paths, routing algorithms can be finally classified as complete and partial routing algorithms.
Various routing algorithms have been proposed for NoCs. Most researchers suggested static routing algorithms and performed communication analysis based on the static behavior of NOC processes, thus, determining the static routing for NOC. Siebenborn et al. and Hu et al. used a CDG (Communication Dependency Graph) to analyze inter-process communications [110] [111].
Most NOC implementations used either XY routing or street sign routing algorithms. In [112], a comparison of deterministic (dimension-order) and adaptive routing algorithms for mesh, torus, and cube networks was presented. Mello et al. researched the performance of minimal routing protocol in NOC [113]. They concluded that the minimal routing provided better results than adaptive routing for on-chip-communications, as the adaptive routing concentrates on the traffic in the center of the NOC.

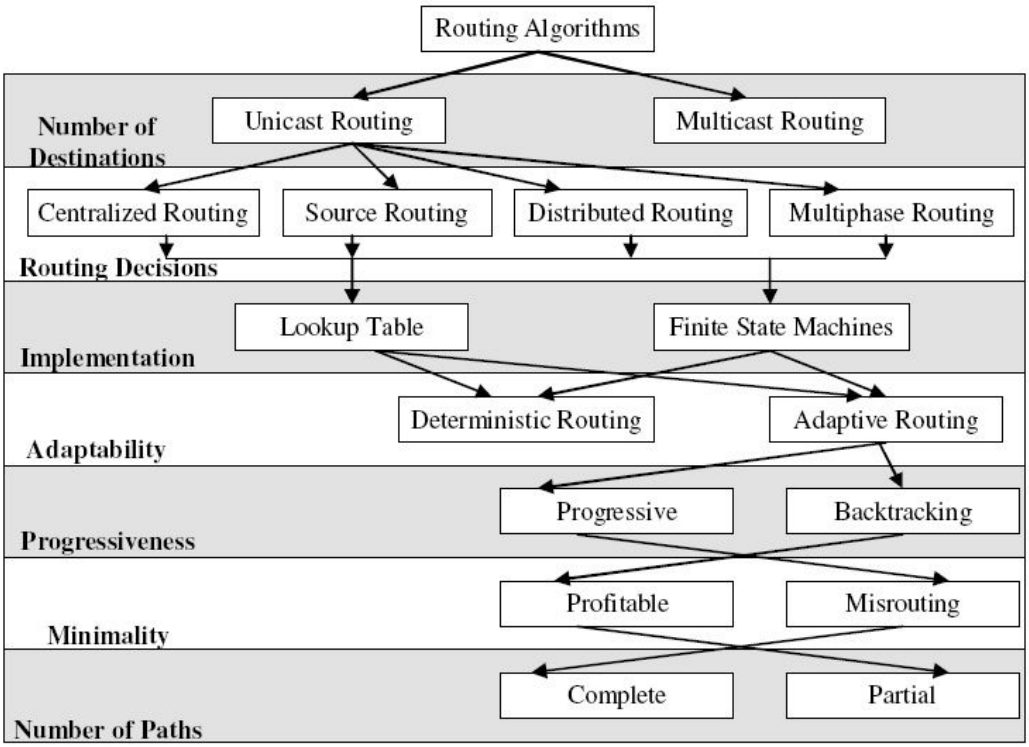


FIGURE 5.1: Classification of routing algorithms [24]

5.2.2 Previous Related Work

Routing algorithms and their implementation for the NoC have been extensively studied in previous related research literature. Jingcao Hu et. al. proposed a special routing algorithm namely DyAD, which is combination of deterministic XY and Adaptive routing [111]. In this routing algorithm, whenever routers are under light traffic, the XY routing is selected, otherwise if congestion exceeds a predefined threshold, routers switch to adaptive routing. A fully adaptive lookahead routing for two clock-cycle NoC router architecture was proposed by Kim et. al. [115]. In this design, the preferred output direction is pre-computed one clock cycle in advance using a congestion aware flow control. Nadera Najib et. al. proposed a partially adaptive look ahead routing algorithms for a low latency, virtual channel wormhole NoC router. In this each router pre computes the preferred output ports based on its local traffic congestion and transfers output ports to the neighboring routers [115]. R. Manevich et. al. proposed a centralized adaptive routing which adaptively select XY or YX deterministic routing for each source-destination pair based on monitoring global NoC traffic [116]. Binzhang Fu et.al. proposed the abacus-turn-model (AbTM) for designing time or space efficient reconfigurable wormhole routing algorithm. AbTM exploits dynamics communication pattern in application to reduce the routing latency and chip area [117]. Su Hu et. al. proposed a probabilistic odd-even (POE) routing algorithm that addresses both deadlock and network performance issues jointly achieves the minimum packet delivery delay [118]. Probabilistic odd-even (POE) determines the deadlock probability for each turn based on the current network status. Sheng Ma et. al. proposed whole packet forwarding (WPF) algorithm for fully adaptive routing that allows non-atomic virtual-channels reallocation on the virtual-channels which has empty place for receiving a whole new packet packet [119].

5.3 Comparison of three popular routing algorithms

5.3.1 Deterministic Routing

The Dimension Order Routing is one kind of deterministic routing algorithms, such as XY routing algorithm [120] for 2D-mesh. In this algorithm, the data packet is routed firstly along the X-dimension and then along the Y-dimension until the packet reaches its final destination. The path between the source and destination used by this algorithm is one of the shortest ones but always the same. XY routing has low latency at low network traffic due to its static, but the performance decreases quickly when the network becomes congested for lacking path diversity. XY routing algorithm performs better than other routing algorithms under uniform traffic pattern, but under nonuniform traffic pattern, XY routing blindly maintains the unevenness of the nonuniform traffic, just as it maintains the evenness for the uniform traffic. The load in the center of a network is much higher rather than total average and this leads to hot spot in the center of network [121]. Another disadvantage of XY algorithm is that it can not handle faulty nodes and regions. If a faulty node or regions (area in the networks having a size larger than a tile) is placed on the path between the source and destination, the packet will remain blocked in one of the switches [122]. However, XY routing has its own advantages,the routing algorithm is simple and can be implemented easily,and its router architecture is very simple and has lower hardware overhead than adaptive router. Then, XY routing easily supports in-order end-to-end packet delivery. Moreover XY routing can easily avoid deadlock and livelock. Therefore, it is not surprising that designers would like to use deterministic routing algorithms in the NoCs which suffer from limited silicon resources.

5.3.2 Partially Adaptive Routing

Examples of the partially-adaptive routing algorithms are the turn model types[123]. In the algorithms based on the turn model, three partially adaptive routing algorithms, namely west-first, north-last, and negative-first, were presented for two-dimensional meshes. The basic idea of turn model is to prohibit the minimum number of turns that break all of the cycles so that deadlock can be avoided. Unfortunately, the degree of adaptiveness provided by the turn model is highly uneven, at least half of the source destination pairs are restricted to having only one minimal path, while full adaptiveness is provided for the rest of the pairs. Such uneven adaptiveness not only causes unfairness but also curtails the ability of the model in alleviating traffic congestion problems. Performance of the network communication may be affected as a result. An improved routing algorithm called Odd-Even(OE) based on turn model is proposed in[124], OE routing algorithm is a sort of distributed partially adaptive routing algorithms, which provides more even routing adaptiveness. Explaining some definitions are necessary in order to represent this algorithm.In this model, a column is called even if its x dimension element is even numerical column. Also, a column is called odd if its x dimension element is an odd number. A turn involves a 90-degree change of traveling direction. There are two main theorems in odd-even algorithm:

1. No packet is permitted to do EN turn in each node which is located on an even column. Also, No packet is permitted to do NW turn in each node that is located on an odd column.

2. No packet is permitted to do ES turn in each node that is in an even column. Also, no packet is permitted to do SW turn in each node which is in an odd column.

In comparison with other turn-model based routing mentioned above, the degree of routing adaptability provided by the OE routing is more even for different source destination pairs. By experiments, it is shows that, under nonuniform traffic pattern, by comparing with XY routing, OE routing has a lower average latency and has a higher throughout, also, when the networks have relatively high injection rates, OE routing can balance load better due to its path diversity. However, as well as other adaptive routing, packets may be delivered by different paths, this may cause packets to arrive out-of-order at receiver. Thus, some control logic should be added to guarantee that packets can be received in order at destination.

5.3.3 Fully Adaptive Routing

A example of fully adaptive routing algorithm was proposed by Duato in[106]. In this algorithm, each physical channel of the network is shared between k virtual channels ($k > 1$). Virtual channels in turn are divided into two classes, namely deterministic class and adaptive class. Consequently, the network is divided into deterministic and adaptive virtual networks. Packets in the adaptive virtual network are routed without any restriction i.e., packets can choose every virtual channel which makes them closer to destination nodes. In the deterministic virtual network, on the other hand, packets are routed according to a deterministic routing algorithm. In the case of a deadlock occurrence between packets traversing in the adaptive virtual network, the deadlock handling mechanism selects one of the engaged packets and releases to break the group dependency. The selected packet then will be routed in the deterministic virtual network until the deadlock is completely resolved. Fully adaptive routing algorithms have lower average packet latency and higher throughout. It has the same problems as partially routing and needs more energy dissipation as compared to the deterministic routing algorithms.

5.4 Route Computation Module

5.4.1 Route Computation *(RC)*

Route Computation is an integral part of any Network-on-Chip router architecture. It implements the routing algorithm that is a network wide operation, and manages the paths that the packets should follow when traveling in the network. Consequently, each router should respect the properties of the routing algorithm and forward the incoming packets to the appropriate output following the path decided by the routing algorithm. In the case of source routing, the packet knows the exact path to its destination beforehand. On the opposite case, when distributed routing is employed the packet carries at its head flit only the address of the destination node. In the case of distributed routing the head flit carries only the address of the destination node. Depending on the routing algorithm, each router should translate the destination address to a local output request allowing each packet to move closer to its destination. This translation is an obligation of the routing computation logic. The routing computation logic can be implemented using a simple lookup table or with simple turn-prohibiting logic.
Routing computation is driven by the destination address of each packet and returns the id

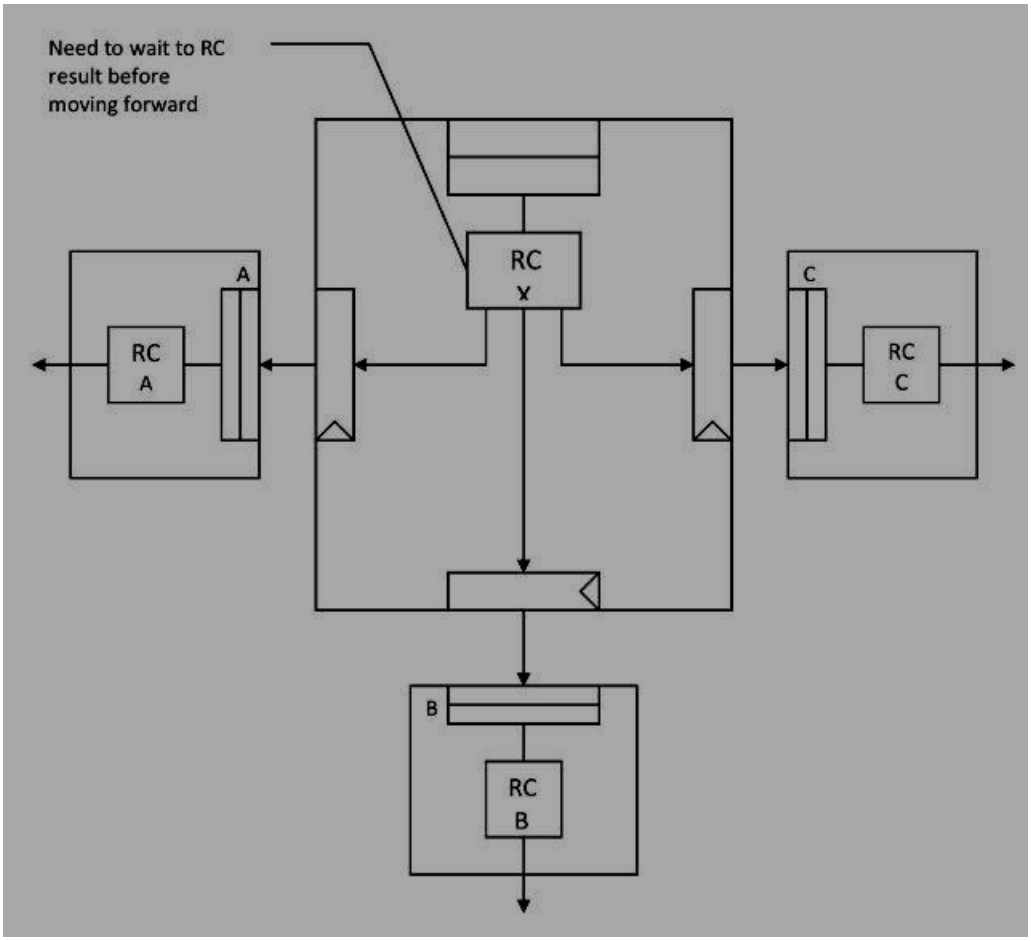


FIGURE 5.2: Baseline RC placement

of the output port that the packet should use for leaving the current router.
In the case of adaptive routing where each packet is allowed to follow more than one paths to reach its final destination the routing computation logic delivers a set of eligible output ports instead of a single output. Selecting the output port to which the packet can leave the router needs an extra selection step that may take other network-level metrics into account such as

the available credits of the eligible outputs or additional congestion notification signals that will be provided outside the routers (Ascia et al. 2008) [125].
The implementation of Route Computation *(RC)* can take many forms. In the traditional case RC, as shown in figure 5.2, each input port first executes RC and then continues with the remaining operations of the router. In this case, the *RC* unit of input *X* selects to which outputs, *A*, *B*, or *C*, the arriving packets should move. Then, once the packet has arrived to the input of the next router it would repeat RC computation for moving closer to its destination.

5.4.2 Look-ahead Routing Computation

Routing computation reads the destination address of the head flit of a packet and translates it to a local output port request following the rules of the network-wide routing algorithm and the *ID* of the current router. In this way, request generation, arbitration and multiplexing should wait first for *RC* to complete before being executed. This serial dependency can be removed if the head flit carries the output port request for the current router in parallel to the destination address. Allowing such behavior requires the head flit to compute the output port request before arriving at the current router using look ahead routing computation. As discussed previously, Lookahead routing computation *(LRC)* was first employed in the SGI Spider switch and extended to adaptive routing algorithms in Vaidya et al. (1999) [109].
In the case of *LRC*, instead of implementing *RC* after a head flit has arrived at the input

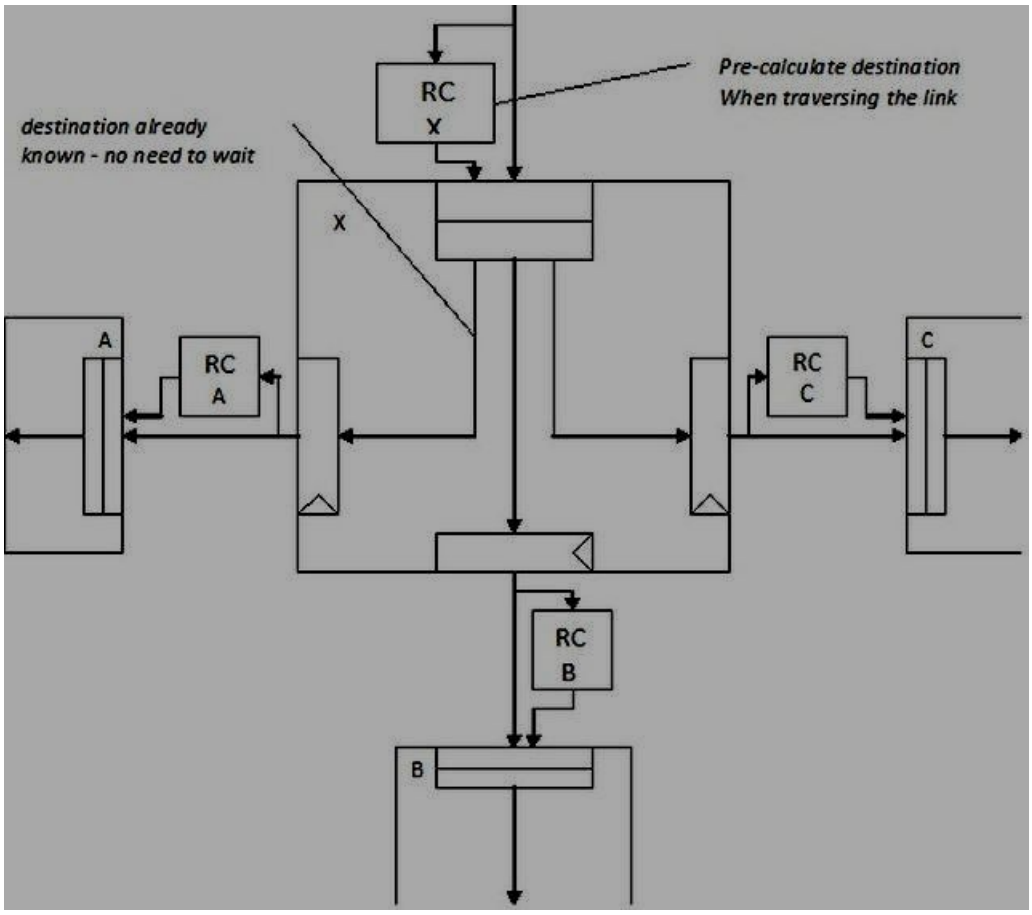


FIGURE 5.3: Look-ahead RC in parallel to Link traversal

buffer, it changes the order of execution and implements *RC* in parallel to link traversal. Therefore, at input *X* the head flit of the packet has already computed RC_x at the link, and

present to the router the pre-computed output port requests. The same happens also for input *A*, *B*, and *C*. Look-ahead routing computation can move one step forward and the let RC_A, RC_B and RC_C modules exist at input X instead of the links. When a packet arrives at input *X*, it has already computed beforehand the output port that should select for arbitration and multiplexing. The output port request vector of a head flit at input X would point to one of the links connecting to the next inputs *A*, *B* or *C*.
Therefore, depending on which output the packet of input *X* is heading to, it should select the result of the appropriate routing computation logic RC_A, RC_B or RC_C The organization of the *LRC* unit at input *X* is illustrated in figure 5.4. All RC units receive the destination address of the packet and based one the output port request of the packet arriving at input *X* selects the output port request for the next router. For example, if the incoming packet moves to input *A* of the next router, the output port requests of RC_C should be selected and attached to the header.

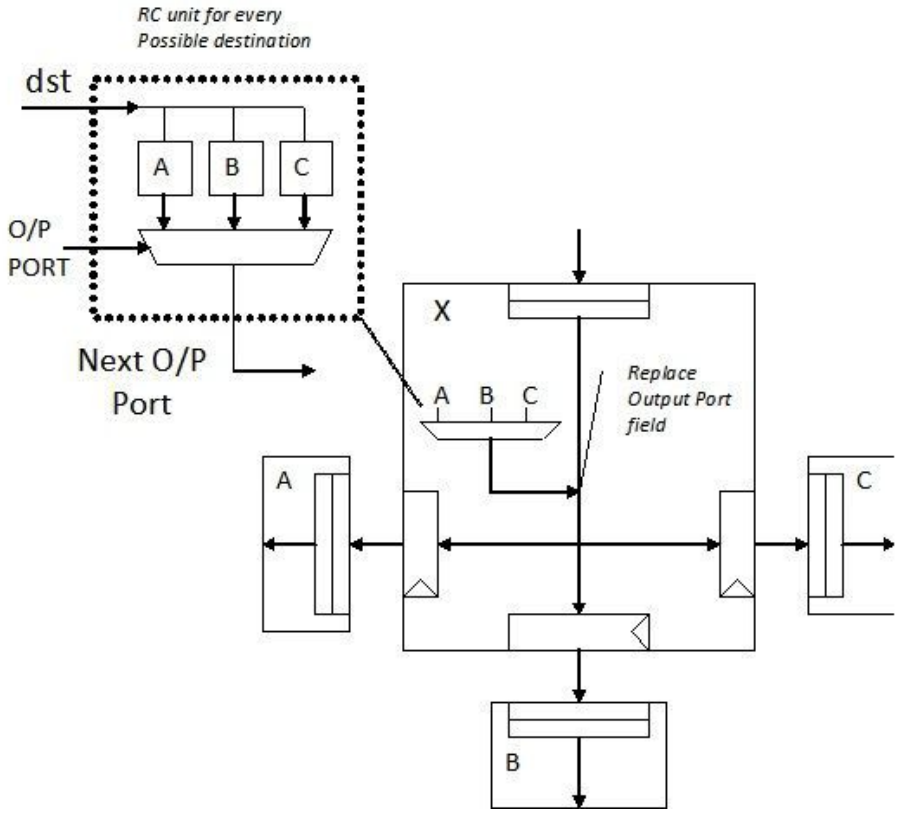


FIGURE 5.4: Look-ahead RC at each input port that runs in parallel

5.5 Routing Algorithms Analysis and Implementation

In this section we describe the three different routing algorithms namely Classic XY, Adaptive XY, and Balanced dimension-order routing with their implementation and analysis on the basis of Area, delay and power.//

5.5.1 XY Routing

XY routing is the most popular deterministic class of algorithms as well as a design methodology. In the case of 2D-MESH, it is straightforward to compute the distance between source and destination node co-ordinate (X-coordinate, Y-coordinate) as the sum of the offsets in all the dimensions. XY routing works on the principle of progressing routing algorithms, consists of reducing an offset to zero before considering offset in the next dimension.
The classic xy routing algorithm is shown in Algorithm 3. The packet is first forwarded to the x dimension until reaching to the same destination column then forwarded to the y direction to reach the destination. As described in Algorithm 1, this routing routes the packets by crossing dimension in increasing or decreasing order, reducing to zero the offset in one dimension before routing in the another dimension.

Input:
Current node co-ordinates (X_c,Y_c),
Destination node co-ordinates (X_d,Y_d);
Output:
Next Port Number
Begin
Direction of Next Port :
EAST, WEST, NORTH, SOUTH, and LOCAL;
X_{diff} := X_d - X_c;
Y_{diff} := Y_d - Y_c;
if (X_{diff} = 0) and (X_{diff} = 0) **then**
Next Port Number = LOCAL;
else
if ($X_{diff} > 0$) **then**
Next Port Number = EAST;
else if ($X_{diff} < 0$) **then**
Next Port Number = WEST;
else
if ($Y_{diff} > 0$) **then**
Next Port Number = SOUTH;
else if ($Y_{diff} < 0$) **then**
Next Port Number = NORTH;
end if
end if
end if

Algorithm 3: XY Routing Algorithm for 2D-MESH Topology

XY routing has several advantages such as simplicity of implementation in hardware as well as preventing out of order packet delivery in destination core. However, in dealing with real-life application, it has low performance since it ignores of network status.

Synchronous Look-ahead routing computation module based on the XY algorithm is implemented in Verilog HDL. The Verilog code is synthesized using Xilinx ISE 14.7 and verified or evaluated using Modelsim Questa 10.2b. Table 5.1 shows the Synthesis report which consists of device count, delays in ns and power dissipation in watts.

Figure 5.5 shows power analysis through four types of graphs. Power analysis of the module is performed by Xpower analyzer and graphs are plotted by Xpower estimator.

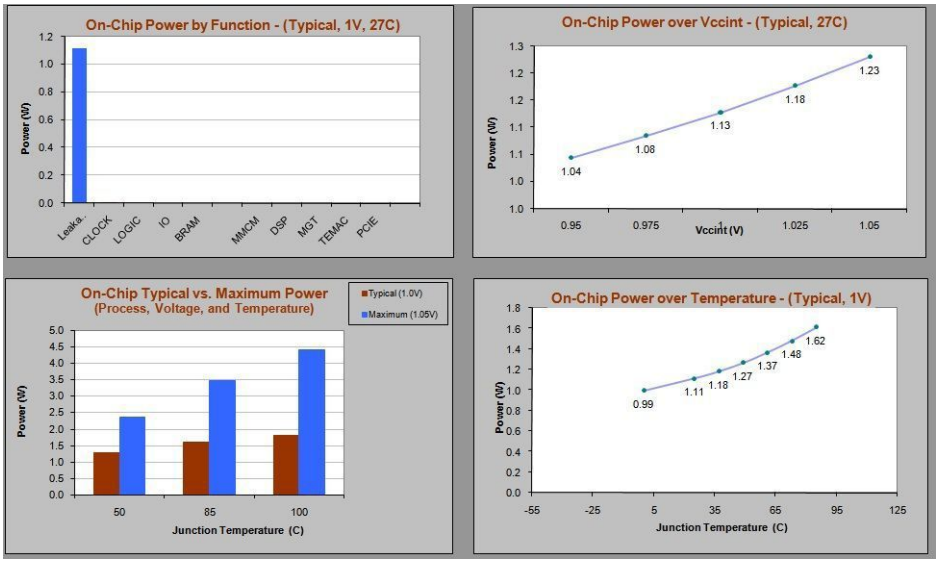


FIGURE 5.5: Power analysis of XY using XPower Analyzer

TABLE 5.1: Synchronous Look-ahead NoC routing computation module using XY

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
8x4-bit single-port RAM :1	3.142 ns	1.146 W
3-bit subtractor : 2	(0.142ns logic, 2.009ns route)	
2-bit register : 2	(6.6% logic, 93.4% route)	
3-bit register : 1		
32-bit comparator greater : 4		
3-bit 2-to-1 multiplexer : 3		

5.5.2 Adaptive XY Routing

It is an adaptive version of classic xy algorithm. First one dimension in x and another in y with less number of nodes is selected then the packet is forwarded to the dimension with less congestion.

An Adaptive XY routing algorithm is the extended version of the classic XY routing algorithm which can improve performance and give support to congestion control and its routing decision to reflect traffic changes. This algorithm gets routing information from adjacent node or from all nodes. Adaptive XY algorithm works on the principle that, when the network is not or slightly congested, it behaves as deterministic and on the other hand it works on adaptive mode when network becomes blocked. While a traditional XY routing causes network loads more in the middle of the network than to lateral areas, the adaptive XY algorithm divides the traffic more equally over the whole network. The algorithm is shown in Algorithm 4.

From an implementation point of view Adaptive XY requires more resources than XY algorithm and of course its takes more power.

As discussed above Synchronous Look-ahead routing computation module based on Adap-

Input:
Current node co-ordinates (X_c,Y_c); *Destination node co-ordinates* (X_d,Y_d);
Congestion_Control: West_Vs_South, West_Vs_North, East_Vs_North, East_Vs_South;
Output:
Next Port Number
Begin
Direction of Next Port : EAST, WEST, NORTH, SOUTH, and LOCAL;
X_{diff} := X_d - X_c; Y_{diff} := Y_d - Y_c;
if (X_{diff} = 0) and (X_{diff} = 0) **then**
Next Port Number = LOCAL;
else
if ($X_{diff} > 0$) **then**
if ($Y_{diff} > 0$) **then**
if (Congestion_Control[East_Vs_South]) **then**
Next Port Number = EAST;
else
Next Port Number = SOUTH;
end if
else if ($Y_{diff} < 0$) **then**
if (Congestion_Control[East_Vs_North]) **then**
Next Port Number = EAST;
else
Next Port Number = NORTH;
end if
else
Next Port Number = EAST;
end if
else if ($X_{diff} < 0$) **then**
if ($Y_{diff} > 0$) **then**
if (Congestion_Control[West_Vs_South]) **then**
Next Port Number = WEST;
else
Next Port Number = SOUTH;
end if
else if ($Y_{diff} < 0$) **then**
if (Congestion_Control[West_Vs_North]) **then**
Next Port Number = WEST;
else
Next Port Number = NORTH;
end if
else
Next Port Number = EAST;
end if
else
if ($Y_{diff} < 0$) **then**
Next Port Number = SOUTH;
else if ($Y_{diff} > 0$) **then**
Next Port Number = NORTH;
else
Next Port Number = LOCAL;
end if
end if
end if

Algorithm 4: Adaptive XY Routing Algorithm for 2D-MESH Topology

tive XY algorithm is also implemented using parametric Verilog HDL.This parametric Verilog RTL code is synthesized using Xilinx ISE 14.7 and synthesis report is shown in Table 5.2. As compared to Table 5.1 (synthesis report of XY algorithm), Adaptive XY requires more produces area.
Adaptive XY produces more delay when compared to classic XY algorithms but Power dissipation is almost the same in both cases. By using Xpower analyzer, power analysis of synchronous look-ahead routing module is performed. Figure 5.6 shows the graphs plotted by Xpower estimator.

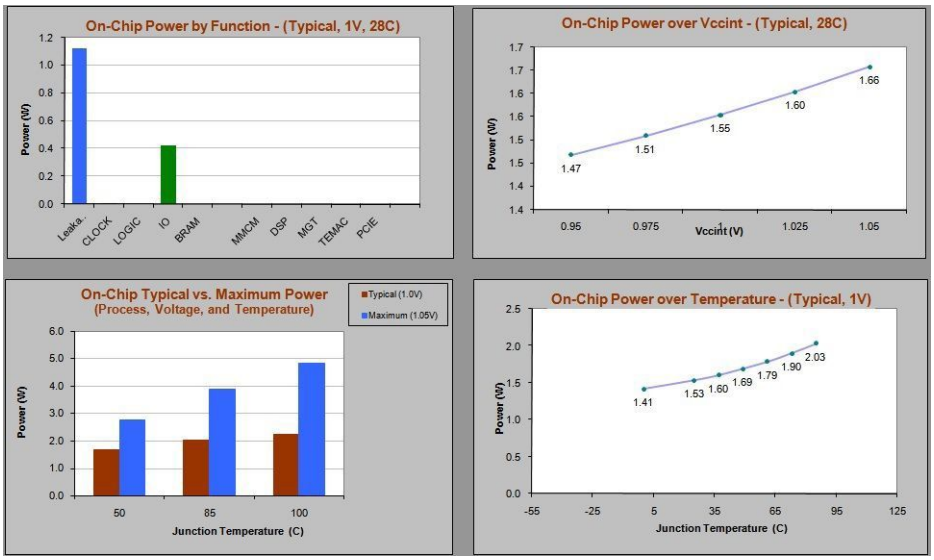


FIGURE 5.6: Power analysis of Adaptive XY using XPower Analyzer

TABLE 5.2: Synchronous Look-ahead NoC routing computation module using Adaptive XY

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
8x4-bit single-port RAM :1	3.85 ns	1.15 W
3-bit subtractor : 2	(1.019ns logic, 2.831ns route)	
2-bit register : 2	(26.46% logic, 73.532% route)	
3-bit register : 1		
32-bit comparator greater : 4		
3-bit 2-to-1 multiplexer : 3		
4-bit 7-to-1 multiplexer : 1		

5.5.3 Balanced Dimension-order Routing

In the case of Balanced dimension-order routing, packet always move to the dimension with higher difference value. It is also considered as a dead-lock free routing algorithm. It provides the balanced minimal path to each destination based on the simple routing regulations.
Balanced Dimension-order routing used by many NoC designers benefits from its simplicity in router design. Dimension-order routing is commonly used deterministic routing algorithm. It provides features of shortest path, deadlock free, simple realization, and applicable to 2D-MESH topology. The algorithms is shown in Algorithm 5.
To evaluate the synthesis report of Synchronous Look-ahead routing module based on the

Input:
Current node co-ordinates (X_c,Y_c),
Destination node co-ordinates (X_d,Y_d);
Output:
Next Port Number
Begin
Direction of Next Port :
EAST, WEST, NORTH, SOUTH, and LOCAL;
X_{diff} := X_d - X_c;
Y_{diff} := Y_d - Y_c;
if ($X_d > X_c$) **then**
$X_{addr_{low}}$ = X_c;
$X_{addr_{high}}$ = X_d;
else
$X_{addr_{low}}$ = X_d;
$X_{addr_{high}}$ = X_c;
end if
X_{diff} := $X_{adder_{high}}$ - $X_{addr_{low}}$;
if ($Y_d > Y_c$) **then**
$Y_{addr_{low}}$ = Y_c;
$Y_{addr_{high}}$ = Y_d;
else
$Y_{addr_{low}}$ = Y_d;
$Y_{addr_{high}}$ = Y_c;
end if
Y_{diff} := $Y_{adder_{high}}$ - $Y_{addr_{low}}$;
if (X_{diff} = 0) and (X_{diff} = 0) **then**
Next Port Number = LOCAL;
else if ($X_{diff} > Y_{diff}$) **then**
if ($X_d > X_c$) **then**
Next Port Number = EAST;
else
Next Port Number = WEST;
end if
else if ($X_{diff} <= Y_{diff}$) **then**
if ($Y_d \leq Y_c$) **then**
Next Port Number = SOUTH;
else
Next Port Number = NORTH;
end if
end if

Algorithm 5: Balanced Dimension-order Routing Algorithm for 2D-MESH Topology

balanced dimension-order routing model, we develop a RTL-level module using Verilog-HDL in Xilinx ISE 14.7. As per synthesis report statistics shown in table 5.3, Balanced dimension-order routing algorithm based on synchronous look-ahead routing computation module requires more device macros with respect to Adaptive XY and XY algorithms.
Balanced dimension-order algorithm results in more delay when compared to classic XY but less than Adaptive XY. But power dissipation is almost the same in all the three cases. Power analysis is again performed by the Xpower Analyzer and graphs are plotted by Xpower estimator of the synchronous look-ahead routing module as shown in figure 5.7.

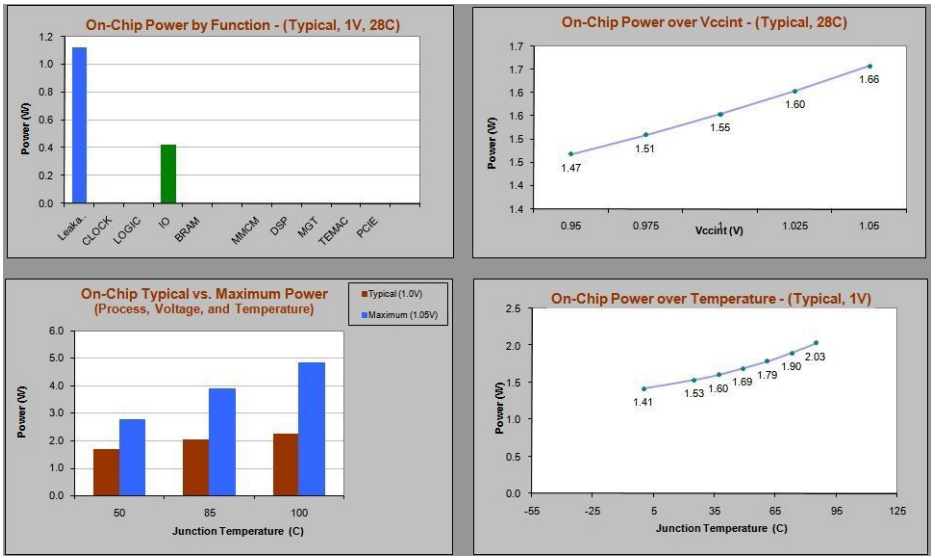


FIGURE 5.7: Power analysis of Balanced dimension-order using XPower Analyzer

TABLE 5.3: Synchronous Look-ahead NoC routing computation module using Balanced dimension-order

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
8x4-bit single-port RAM :1	3.399 ns	1.149 W
2-bit subtractor : 2	(0.951ns logic, 2.448ns route)	
2-bit register : 2	(27.97% logic, 72.03% route)	
3-bit register : 1		
2-bit comparator greater : 3		
2-bit 2-to-1 multiplexer : 4		
3-bit 2-to-1 multiplexer : 3		

5.6 Conclusion

In this section we summarize routing algorithms used for NoC by simply comparison and analysis,and give the merits and demerits of some routing algorithms.
In this chapter, we presented RTL implementation comparatively analysis of classic XY, Adaptive XY, and Balanced dimension-order algorithms based Synchronous Look-ahead routing module using Verilog-HDL for the low latency NoC router.
Synthesis report shows implementation of Adaptive XY routing requires more device macros than Classic XY and Balanced direction-order routing algorithms but performance-wise, specially in the case of traffic congestion, Adaptive routing is better than others two algorithms.

From a delay point of view again performance of Adaptive XY routing is worse than classic XY and Balanced dimension-order but in case of power dissipation, performance of all the discussed algorithms is almost the same.
The addition of a routing computation module transforms a switch to a network router that can participate in an arbitrary network topology allowing incoming packets to find their path towards their destination.

Chapter 6

Network Interface (NI)

6.1 Summary

The multi-core revolution pushes to the limit the need for a fresh interconnect and memory design to deliver architectures that can match current and future application requirements. Moreover, the current trend towards multi-node multi-core architectures to further improve the system performance imposes the use of hierarchical and eventually heterogeneous interconnection subsystems.
Increasing design complexities and time-to-market pressures have made modular SoC development inevitable. Obviously, the most fundamental requirement for a modular SoC development is the ability to interconnect the cores with the minimum effort. Point-to-point communication standards such as OCP (Open Core Protocol) [128], VCI (Virtual Component Interface) [129] and AMBA AXI [130] were introduced to address this issue in bus-based SoC development. The communication interfaces have been successfully adopted to offer communication between an IP core and the bus, as well as between two IP cores. The existing communication protocols adopted in bus-based development can be used in the NoC realm to connect the IP cores to a NoC. However, the hop-by-hop and (possibly) packet-based nature of interconnection in NoC requires a specifically customised interface between the core and the interconnection network. In the NoC paradigm such an interface is referred to as network interface (NI).
In this scenario the Network Interface (NI) controller represents a critical component to allow an easy efficient and effective link between the cores and the memory blocks to the interconnect. Moreover, the buffers and the pipeline stages of the NI must be carefully evaluated not to waste valuable on-chip resources.
The rapid introduction of NoC for the multi-core GALS architecture requires proper Network Interface (NI) design to interface two different IP blocks via network routers within the same chip. This chapter proposes a DMA based NI for connecting the NoC router to a processor element (PE). This proposed NI has three memory mapped registers, Read packet register, Write packet register, and Status packet register. This chapter focuses on proposed NI design and analyses its performance in terms of power, area, and delay using RTL NI model. The RTL coding is performed using Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology.

6.2 Introduction

Network-on-Chip (NoC) has been proposed to support integration of multiple IP cores on a single chip in the new SoC design paradigm [2] . The NoC architectures have been recommended as a capable solution for highly scalable, reliable, and modular on-chip communication infrastructure platform [127]. The rapid growth of the multi-core GALS architecture requires proper NI design to interface two different interconnects in the same chip, rather

than a single IP block to the interconnect as shown in figure 6.1. For reducing the design time, the reuse of IP blocks can be achieved by using a flexible network interface and plug and play manner[75].

The NI functions as a glue between computation and communication by implementing the

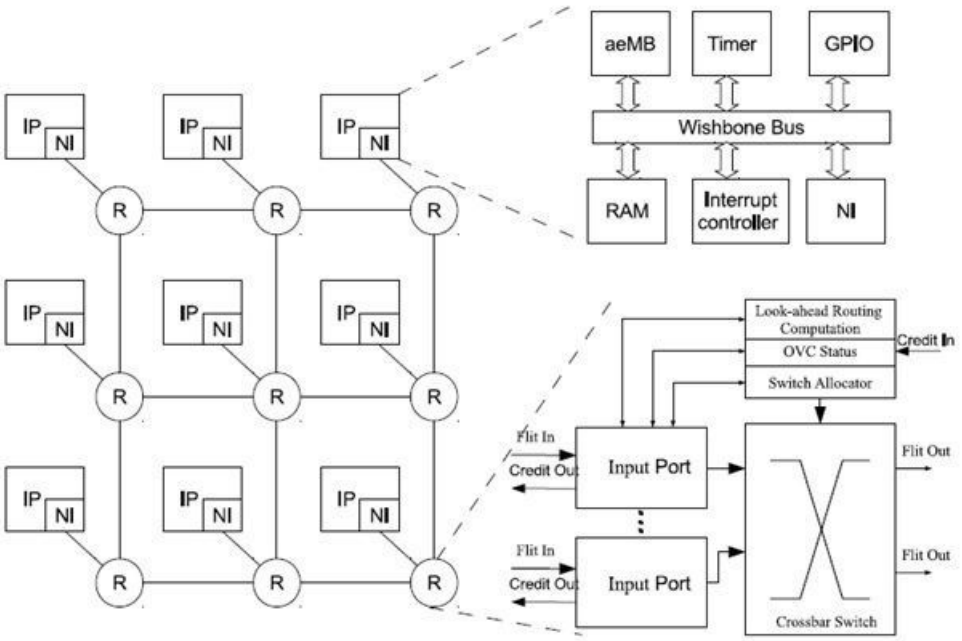


FIGURE 6.1: NoC's architecture for 2D-MESH topology

interfaces to both the IP core and interconnection network. The IP core interface implements a standard point-to-point protocol allowing core reuse across several platforms. The most widely used core interface protocols include OCP [128], VCI [129], AMBA AXI [130]. These interfaces assume the attributes of a socket, which captures all signaling between the core and the system. This can offer a transaction-based model [38] of communication which is backward compatible with bus-based SoCs. Message passing and shared memory abstraction are two transaction-based programming models employed in the NoC domain. Shared memory is easier to implement, while message passing is more scalable.

The NI core interface can be viewed as an implementation of the session layer in OSI reference model. Traditionally, the session layer represents the user's interface to the network. It determines when the transaction session is opened, how long it will be used and when to close it. Moreover, it controls the transmission of data during the session, supports security and name lookup, enabling computers to locate each other. Flow control strategies and QoS negotiations can be also implemented at this layer.

The communication services made available at the session layer must be implemented by the transport layer, in order to make the communication behavior fit to the interconnect. The transport layer provides reliable, sequenced, and QoS-oriented data transfer. This layer provides the basic end-to-end connection.

The transport layer provides transparent transfer of data between end nodes using the services of the network layer. These services, together with those offered by the link and physical layer are implemented in the network interface part of the NI. Data packetization and routing related functions are considered as essential tasks performed by the network layer; offering a reliable links is considered as a service of the data-link layer.

Network interface design has been extensively researched for parallel computers [131-133], and computer networks [134-135]. The designs of the interface for such networks are to

optimize for performance (high throughput, low latency), and often consist of a dedicated processor, and large amount of buffering, hence prohibitively costly for the NoC paradigm. On-chip network interfaces must provide a low area overhead, because the size of the IP modules attached to NoC is relatively small.
This chapter will focus on a 2D-MESH topology NoC architecture and will propose a DMA based Network-Interface (NI) for connecting the NoC router to a processor. This Network Interface contains three type of memory mapped registers, which are Read Packet Register, Write Packet Register, and Status Register. The first one, Read packet register, contains the pointer and maximum size of the pointer which has been dedicated by IP block to store the received packet. The second register contains the pointer and maximum size of the pointer of the packet which must be sent. The destination address must be updated by IP block in the first word of the packet and the last register provide information about the current status of the router [79].

6.2.1 Related Work

Network Interface design is important in the communication-centric system of Multi-processor SoCs. In [136], a TV companion chip was redesigned with a NoC as interconnect, and 78% of increase in chip area was proved to come from the NIs. In a similar case, in the xpipes based system in [137], more than half of the NoC area is due to NIs. Despite this NI design doesn't receive a lot of attention in the literature, since the support for processing cores with advanced communication capabilities (e.g., multiple outstanding transactions, out-of-order completion, quality of service guarantees) requires complex architectures with a restricted margin for optimization. There are several literature available for standardization of Network Interface fabrics, their throughput and speed. Most of the Interface are based on Advance Microcontroller Bus architecture (AMBA), AXI, GALS and open core protocol. S.P.Singh et al. implemented a generalized Network Interface structure for fast plug-and-play with minimum overhead [138], whereas Daneshtalab et al. implemented efficient Network Interface architecture that is based on transaction based protocol and suitable for Network-on-Chips [139]. W. Chouchene et al. [140] implements a low power Network Interface for Network-on-Chip, based on the OCP interface with multi-clock. Seung Eun Lee et. al. [141] present a generic architecture for network interface (NI) and associated wrappers for a networked processor array (NoC based multiprocessor SoC) in order to allow systematic design flow for accelerating the design cycle. Paul Wielage and Kees Goossens [142] proposed to investigate hybrid architecture with first-level communication over a shared-medium, and the higher levels using a packet-switched network. The usefulness of resource sharing has been proved in [143], where sharing an NI between several nodes significantly reduced the area overhead of the NoC with low performance overhead in terms of execution cycles. The architecture consists in an arbiter placed between the processing elements and the Network Interface. In [144], an existing NoC has been extended to support WISHBONE interconnection architecture, so it can be used in many SoC designs based on the WISHBONE interface. Radulescu et al [38] design a modular NI that interfaces a NoC with several protocols to allow reusability and flexibility of the attached IPs. In [145] a clock gating technique is implemented to reduce the NI power consumption. The NI is designed as a bridge between an OCP interface and the NoC switching fabric. Bolotin et al [21] implement an irregular mesh NoC where NI offers a bus-like protocol for the connected IP with conventional read and write semantics, and a QoS with four service priority levels. However, they use a functional NI description and do not analyze the area and power overhead introduced by the NI.

6.3 NI Fundamentals

This chapter will focus on a two stages, wormhole virtual channel Bi-NoC router. In NoC IP cores communicate by sending network packets. An NoC packet consists of several fixed size flow control digit (flits). There are three type of flits: header flit, buddy flit and tail flit. The minimum packet size is 2 flits,(one header and one tail).

6.3.1 NoC Flit Format

Figure 6.2 illustrates the format of NoC flit. A flit is made two parts of header and payload. The header part is assigned automatically by NoC and indicate the flit type and the assigned virtual channel number to the packet. The payload is 32 bit wide and will be assigned by software. The payload in buddy and tail flits contains the data which is going to be transferred to the destination core. The payload of header flit contains information for forwarding the packet between cores which has been described in Table 6.1.

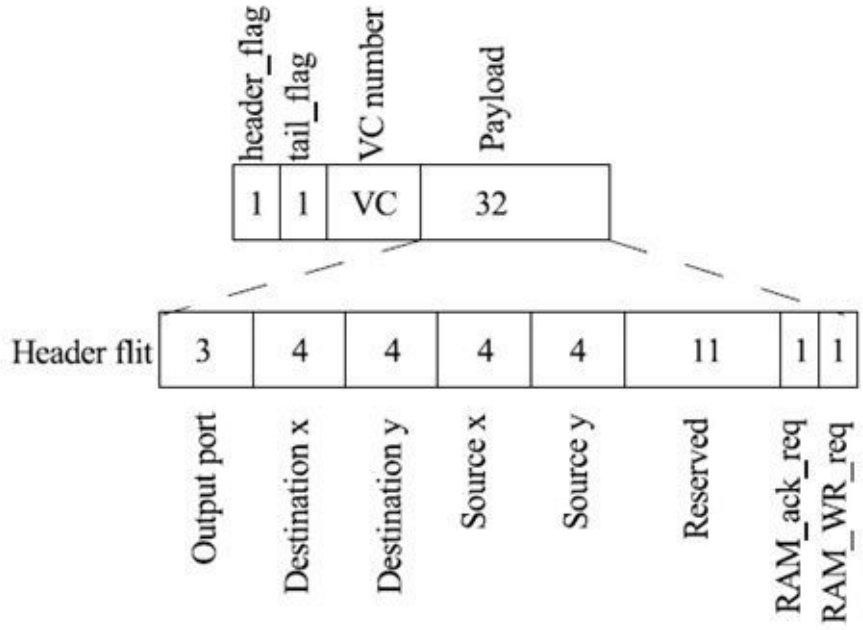


FIGURE 6.2: The NoC packet format

TABLE 6.1: The header flit fields description

Filed name	*Description*
Output port	The result of look ahead routing algorithm.It will be updated by NI at source and NoC routers along the path.
Destination x & y	The destination IP x & y address
Source x & y	The source IP address, it will be used by the shared memory IP core to send the requested data or ack to the source core.

Continued on next page

Table 6.1 – *Continued from previous page*

Filed name	*Description*
RAM_ack_req	This field is for communicating with shared RAM. By setting this field the processor asks for receiving an acknowledge packet from shared ram core after the write process on ram is finished.
RAM_WR_req	This field is for communicating with shared RAM. If its set the packet contains data which must be written on shared RAM otherwise it contain a read request.

6.3.2 The NI Registers

The NI works as a bridge between the wishbone bus and the NoC router. The NI transfers data between the NoC and the program RAM in a DMA-like way. The NI has only three memory mapped registers: read, write and status register. Figure 6.3 illustrates the NI internal registers. The Read register is used for capturing received packet from NI to the programming memory. The capturing process is started by writing the address pointer of the buffer and its size on the read register.

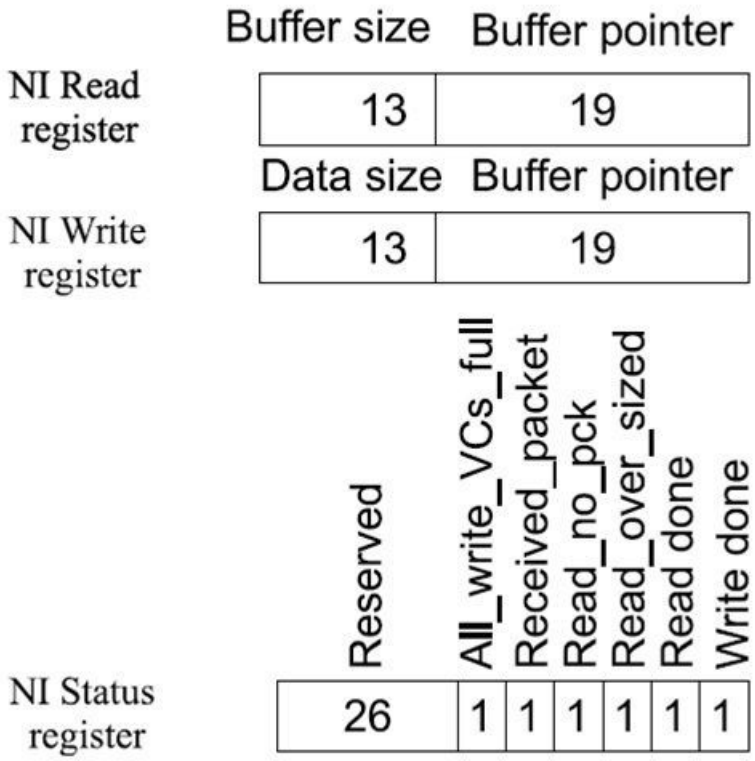


FIGURE 6.3: The NI registers

By writing the buffer pointer address and the size of data to be sent in word on write register the NI starts sending the packet to NoC.
The status registers hold some information about the status of NI. This information are shown in table 6.2.

TABLE 6.2: The NI Status fields descriptions

Filed name	*Description*
Write done	A flag which has been asserted when the packet is sent out from the NI. This flag remains zero during sending process.
Read done	A flag which has been asserted when the process of transferring a packet from NI to the programming RAM is finished. This flag remains zero during transferring of packet.
Read_over_size	An error flag which indicates that the length of received packet was larger than the dedicated buffer size.
read_no_pck	An error flag which indicates an store (read) command has been received by NI while there was no received packet in NI.
All Write VCs full	This flag is asserted when all write VC are full, Hence, until this flag is asserted no packet can be sent out to the NoC.

6.3.3 The NI Architectural View

Figure 6.4 shows the NI architecture that is composed of two loosely coupled modules. The first module handles packets coming from the NoC, and sends them to the node over the bus. The second module receives messages from the bus and injects them into the network.

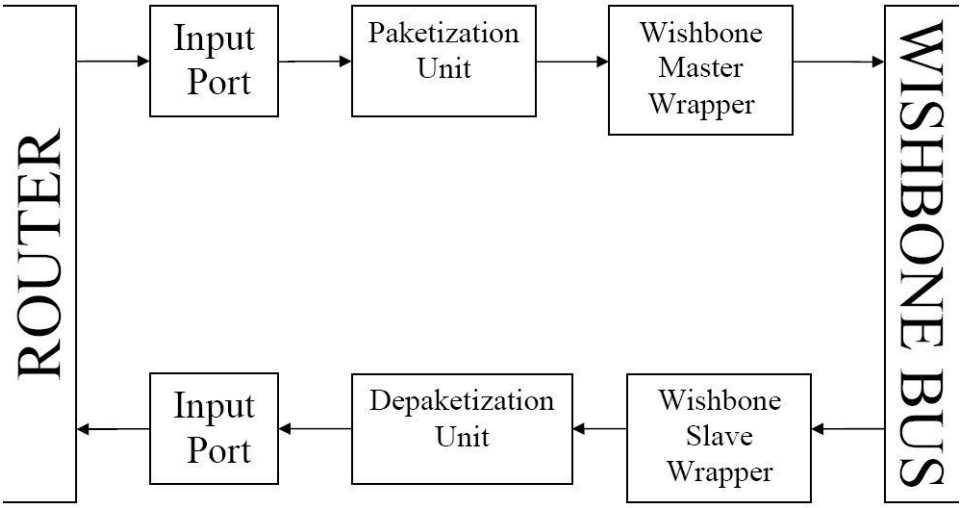


FIGURE 6.4: The proposed NI design : Block Diagram

In particular, input Port receives its inputs from the NoC. When a tail it arrives and an entire packet is received, the message is extracted and moved in the Message Queue. Thus, the message waits until it can be transmitted over the bus through the WISHBONE Master Wrapper. On the other hand, a message is received from the bus thanks to the WISHBONE Slave Wrapper, while it remains packaged and stored in the Packet Buffer. After that, like for NoC, a virtual channel must be assigned at the packet and a link arbiter manages the transmission of various inputs in the buffer on the single link available.

6.4 Complete RTL Model

Figure 6.5 shows the Network Interface RTL model written in Verilog HDL. It has a modular implementation that follows the architecture discussed above. Timing information about the pipeline stages has been obtained from this model. Furthermore, the pipeline changes allowed the extraction of conservative timing latency of three and two stages pipeline architectures as well as of a single cycle NI architecture.

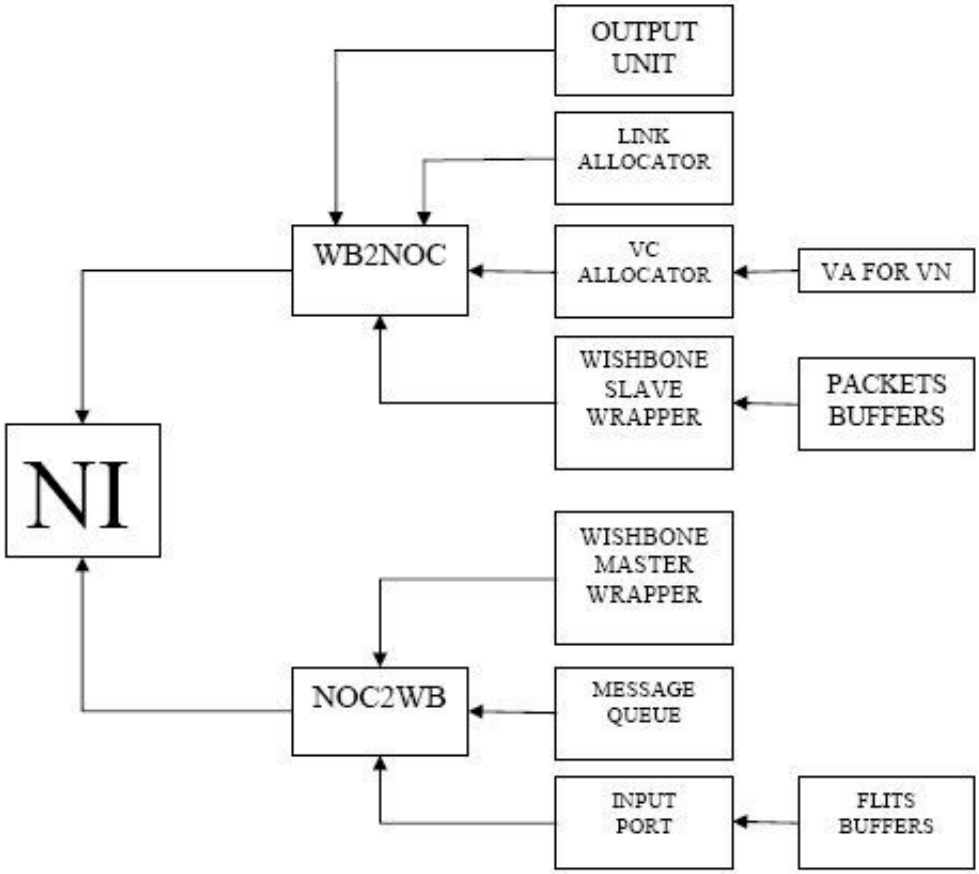


FIGURE 6.5: Verilog HDL modules inclusion

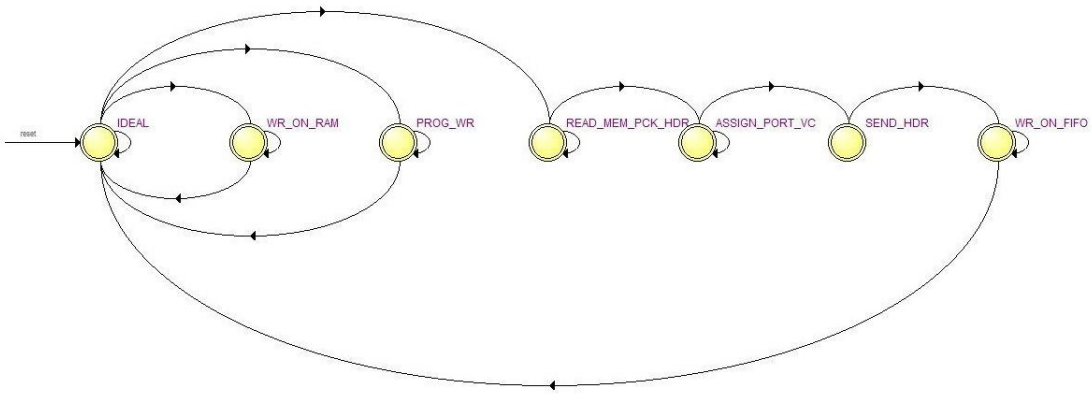


FIGURE 6.6: Finite state machine NI controller

Figure 6.5 shows the Verilog HDL modules inclusions. In sub-module noc2wb all the NoC-to-Node pipeline is implemented and in sub-module wb2noc all the Node-to-NoC pipeline is implemented. The decoupling of the two pipelines makes them easy to extend and optimize. The model can be adapted for other bus interfaces simply changing the wrapper modules.
Figure 6.6 show the finite state machine of the NI controller, which show how the data packet travels through out the network interface.

6.4.1 The Packet Buffer

The Packet Buffer represents a temporary storage for incoming packets from the bus that are waiting to be injected into the NoC. It is composed of several buffers, each of them can store the longest packet of the network. When a message is received by the WISHBONE Slave Wrapper, it is packaged and stored in a free buffer. If no free buffer is available, the WISHBONE Wrapper stops the communication and will restart it only when space is freed. In this buffer the packet waits for virtual channel allocation and transmission of flits. Figure 6.7 show the block diagram of packet buffer generated by Xilinx.

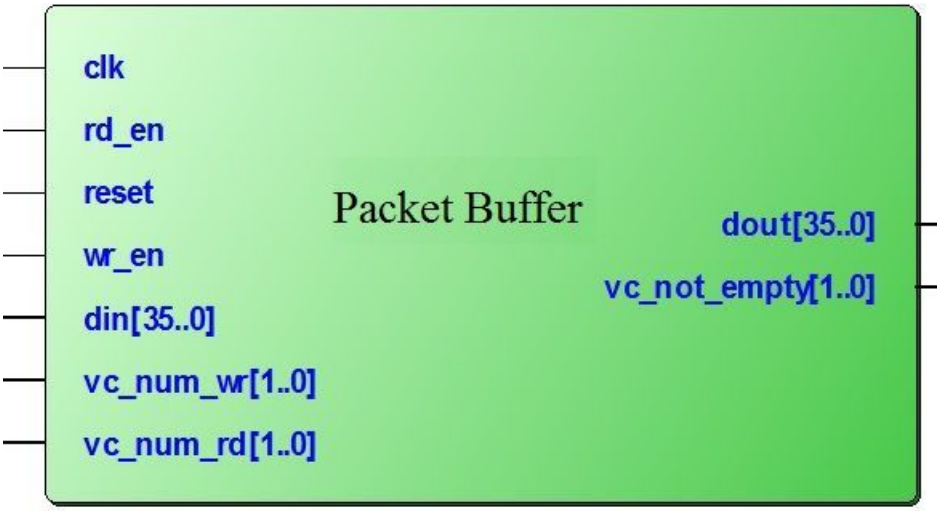


FIGURE 6.7: Symbol of Packet Buffer Module

6.4.2 The Virtual Channel Allocator

Traditionally, the Virtual Channel Allocator stage contains the critical path of the router pipeline. In the Network Interface this module is considerably less complex since a single output port is available.
An N : 1 arbiter is used for each virtual network. Each arbiter receives N requests, where N is the maximum number of packets that can be stored in the Packet Buffer, and allocates an available virtual channel only for one of them, if any. Note that this architecture does not underutilized the link bandwidth, since even if only one it per cycle wins the VA stage, this it will be transmitted on the only link available. Figure 6.8 show the block diagram of Virtual Channel Allocator.

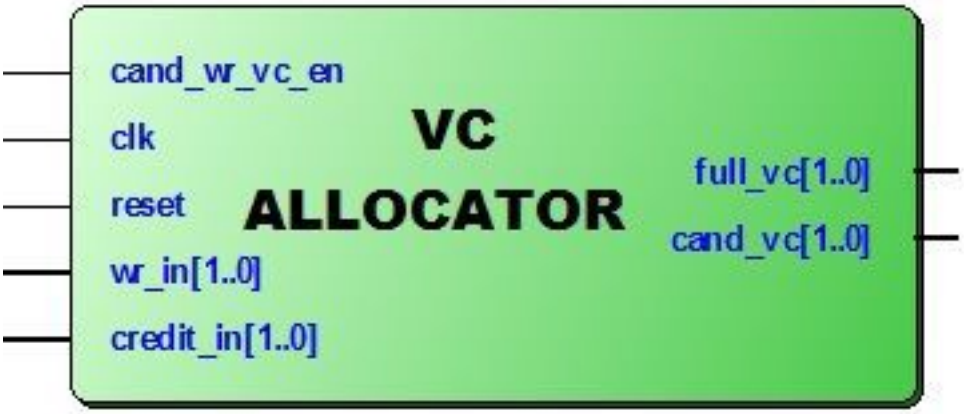


FIGURE 6.8: Symbol of Virtual Channel Allocator Module

6.4.3 The Link Allocator

The Link Allocator is an arbiter N : 1, where N is the maximum number of packets that can be stored in the Packet Buffer. It receives the request transmission from the its that have already won the VA stage. Only one of them can win and use the link to the router in the next cycle. The Link Allocator architecture is same as the architecture of the Arbiters as shown in figure 6.9.

FIGURE 6.9: Symbol of the Link Allocator of Module

6.4.4 The WISHBONE Wrapper

The implemented NI exposes the standard interface of a WISHBONE master and a WISHBONE slave IP, since it must act as a slave and a master from both the bus and the NoC perspective. Internally, to keep logically separated the reception and transmission of a message, the Slave Wrapper always receives messages, and the Master Wrapper always sends messages. This implies that the Slave Wrapper must receive the reply of a slave node to a read transaction started by the NI MasterWrapper, and the MasterWrapper must transmit the reply of a read transaction for the NI Slave Wrapper. Figure 6.10 shows in details the wrapper interactions. The rest of the section discusses every possible transaction to clarify the implementation. When a packet is received, if it is a write transaction, it is simply sent by the Master Wrapper on the bus, since no reply must be sent back. If the packet contains a read transaction, the MasterWrapper starts the transaction and the reply is stored in the Slave Wrapper. The Slave Wrapper does not allow the Master to start a read transaction if no space is available in the Packet Buffer.

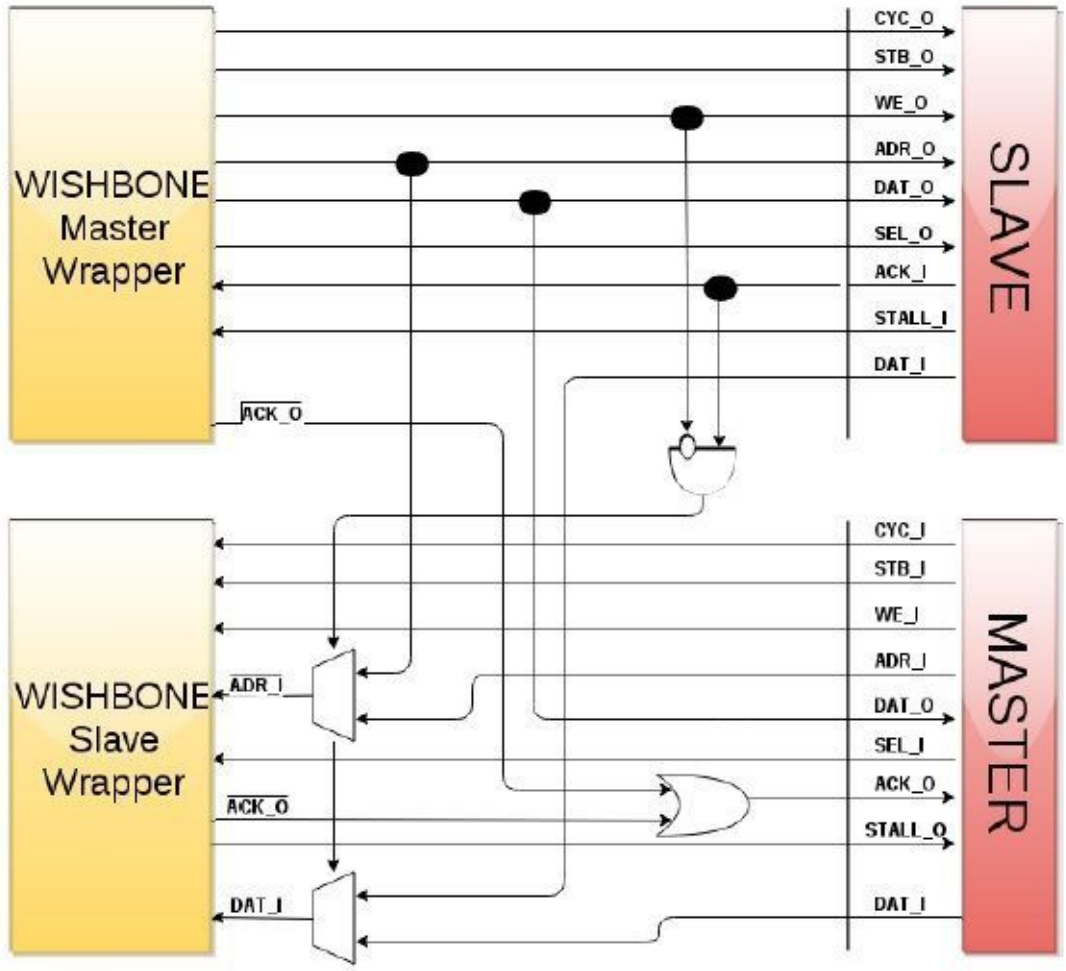


FIGURE 6.10: Wrappers architecture

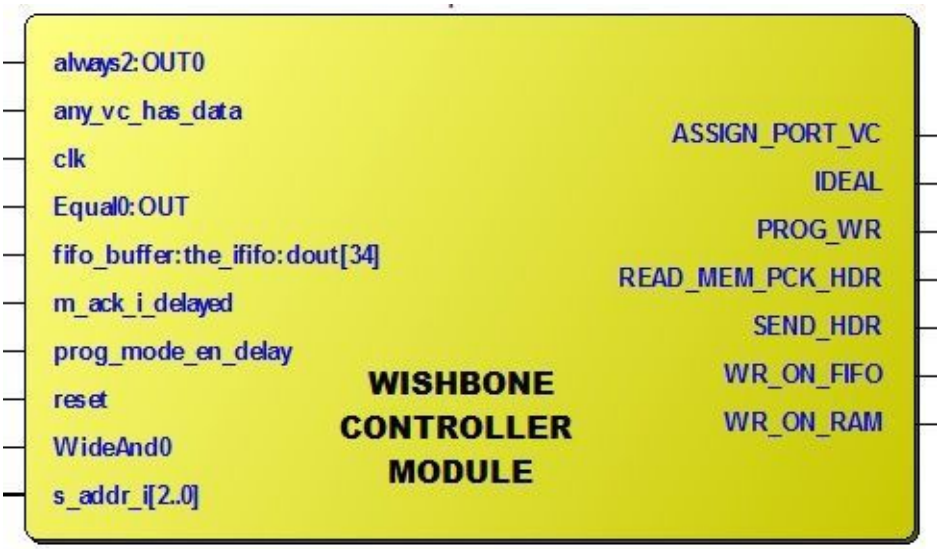


FIGURE 6.11: Wrappers module generated by Xilinx

When the NI acts as a slave two cases are possible. If it serves a write transaction, the Slave Wrapper receives the message and injects it in the NoC. If it serves a read transaction, the Slave injects the request in the NoC and saves an entry in the On-the-Fly table, with the

information of the waited message. When the Master receives a packet that contains the reply for the transaction, it replies for the Slave and closes the transaction. This kind of architecture is required to overcome the absence of split transaction in the WISHBONE bus. If this feature had been available it would not be need of the table, since in this case a master always sends and a slave always receives messages. Figure 6.11 shows the Block Diagram generated by the software itself.

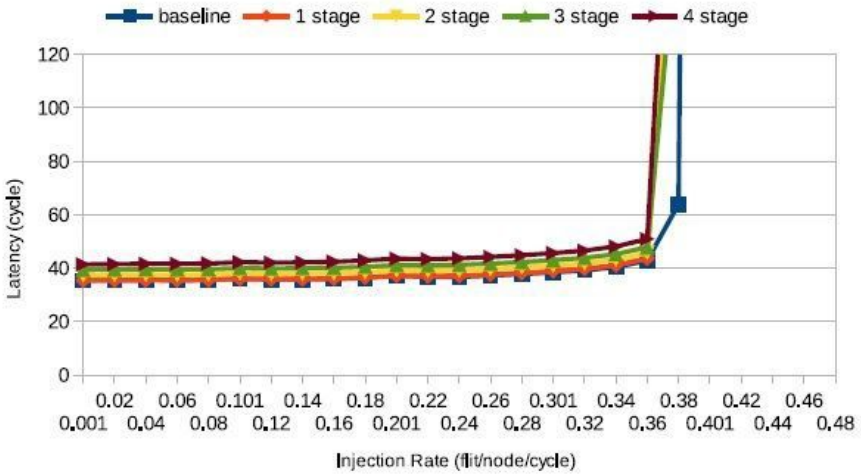
6.5 Results and Conclusion

In this chapter, we have proposed and implemented a DMA based Network-Interface for connecting the NoC router to a processor element. The NI is implemented using Verlog HDL code with the help of Xilinx 14.7 and the synthesis report is shown in table 6.3.

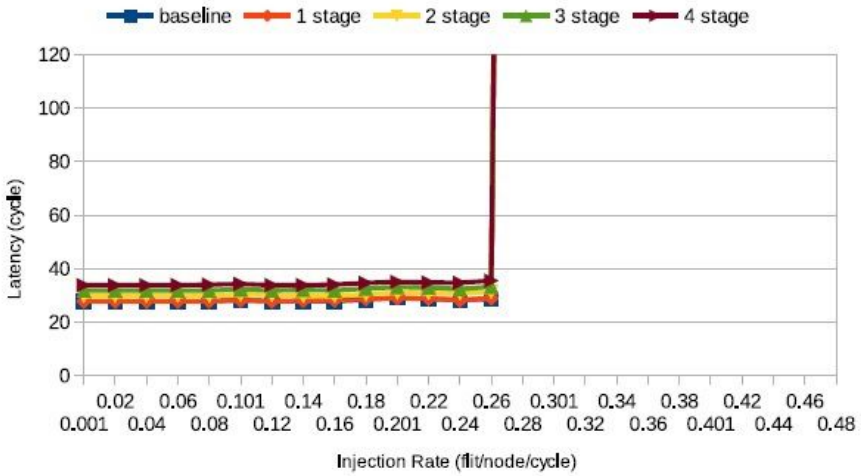
TABLE 6.3: The NI Status fields descriptions

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
32x34-bit dual-port RAM = 1	11.804 ns	4.418 W
12-bit adder = 1	(32.361% logic, 67.638% route)	
13-bit adder = 2	(3.82 ns logic,7.984 ns route)	
4-bit adder = 4		
5-bit adder = 4		
5-bit substractor = 4		
1-bit register = 36		
12-bit register = 1		
13-bit register = 1		
17-bit register = 1		
2-bit register = 4		
13-bit register = 1		
3-bit register = 1		
34-bit register = 1		
5-bit register = 2		
12-bit comparator equal = 1		
5-bit comparator greater = 4		
5-bit comparator lessequal = 2		
1-bit 2-to-1 multiplexer = 69		
12-bit 2-to-1 multiplexer = 1		
17-bit 2-to-1 multiplexer = 1		
2-bit 2-to-1 multiplexer = 11		
3-bit 2-to-1 multiplexer = 1		
Finite State Machine = 1		

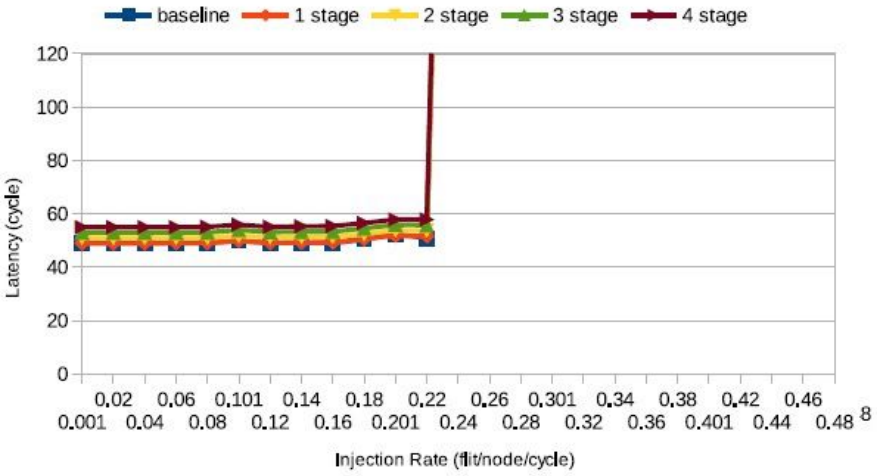
Figure 6.12 allows a categorization of the applications based on the injected traffic. The remaining benchmarks any one could be identified as medium traffic applications. Figure 6.12 shows Average packet latency with data packet size of one flit. Different synthetic traffic pattern are used: uniform random, tornado and bit complement. Figure 6.12 also shows the results with synthetic traffic where only single flit packets are used.



(A) Uniform Random



(B) Tornado



(C) Bit Complement

FIGURE 6.12: Average packet Latency analysis with data packet size of three flits

The 1 stage NI architecture provides almost the same latency of the baseline for low

traffic. This is due to the single pipeline stage and the single it message that are not impacting both the contention metric of the node bus. Moreover, the 2 stages introduces 2 more latency cycles with respect to the 1 stage, due to an additional pipeline stage in each traversed NI. Similarly, the 3 stages adds 4 more latency cycles, and the 4 stages architecture adds 6 cycles, with respect to the 1 stage, due to their additional pipeline stages in each traversed NI.
Power analysis is again performed by the Xpower Analyzer and graphs are plotted by Xpower estimator of the Network Interface module as shown in figure 6.13.
Considering the current trend towards multi-node multi-core architectures,where each node is composed of a multi-core and multiple nodes and glued into a single system towards a dedicated intra-node interconnect, the thesis explores different trades-issues and solutions for the implementation of the Network Interface (NI) controller for such systems. The NI represents the junction point between the inter-node and intra-node interconnect, thus playing a crucial rule in the performance of the whole system. Moreover, the resource allocation and the NI design represent a critical decision point for the power-performance-area optimization.

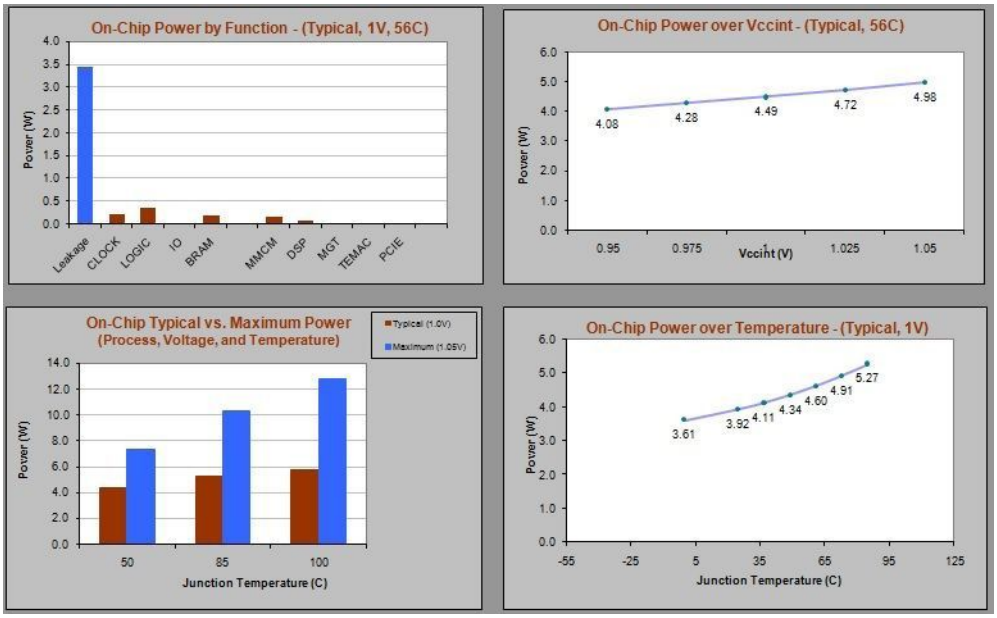


FIGURE 6.13: Power analysis report of NI module

A design exploration of the architecture is required to dimension the shared resources of the Network Interface, like the number of message buffers or the bus data width. Most of these design decisions in influence the area and latency trade-off.

Chapter 7

Virtual Channel Allocators and Switch Allocators

7.1 Summary

Network throughput can be increased by dividing the buffer storage associated with each network channel into several virtual channels (VCs). Each physical channel is associated with several small queues, virtual channels, rather than a single deep queue. The virtual channels associated with one physical channel are allocated independently but compete with each other for physical bandwidth. Virtual channels decouple buffer resources from transmission resources. This decoupling allows active messages to pass blocked messages using network bandwidth that would otherwise be left idle.

The Wormhole flow-control based conventional bidirectional router requires two input buffers to receive two packets from different directions at the same time. In the proposed router architecture, we provide the link utilization flexibility by sharing the access authority of two inputs multi-synchronous FIFOs for the two in-out ports at the same direction, where two input FIFOs can be multiplexed on two physical channels in each direction. Actually, virtual channels behave like an architecture which has numerous wormhole links existing in parallel. Therefore, this approach can possibly improve performance by decreasing the Head-of-Line Blocking effect of the channels. Since in the proposed router, two physical channels shared the virtual channels in each direction at a time, then total number of virtual channels is equivalent to the conventional bidirectional virtual channel router.

Once a packet has completed Virtual Channel (VC) allocation, its flits can be forwarded to the selected destination port subject to buffer space availability. In conventional router architecture, for each flit to be transferred, a crossbar connection between the corresponding input and output ports must be established for one cycle. The switch allocator is responsible for scheduling such crossbar connections; in particular, it generates matching between requests from active VCs at each of the router's P input ports on the one hand and crossbar connections to its P output ports on the other hand. In the proposed router architecture, since the channels are bidirectional links bandwidth is doubled from P to 2P in each output direction. The grant signals generated by the switch allocator are used to set up the registers that control crossbar connectivity. In addition, the switch allocator notifies the winning VC at each input port, causing the latter to prepare its front most flit for crossbar traversal.

In this chapter we have implemented VC allocation using separable allocators as Separable input-first VC allocator, Separable output-first VC allocator, and Wavefront-based VC allocator. We have also implemented switch allocation using separable allocators as Separable input-first switch allocator, Separable output-first switch allocator, and Wavefront-based switch allocator. The RTL model is implemented using Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. Their performance are evaluated in terms of power, area and delay.

7.2 Introduction

In Virtual Channel (VC) flow control, when the head flit of a packet arrives at a router, it must acquire one of the VCs associated with the physical channel that connects to its destination output before it can proceed. To achieve this, the head flit sends a request to the VC allocator once it reaches the front of its input VC. The VC allocator generates a matching between any such requests from the input VCs on the one hand and those output VCs that are not currently in use by another packet on the other hand.
In the general case, a router with P ports and V VCs per port therefore requires a VC allocator that can match P×V agents (all input VCs at all input ports) to P×V resources (all output VCs at all output ports). The VC allocator architecture in our proposed router is the same as the available VC flow control based router except there is no overhead associated with this stage (VA stage), since in our router design, there are two bi-directional links in each direction among VC buffer and here we used multi-synchronous FIFOs for each virtual channel.
Virtual channels is analogous to adding lanes in a street allowing cars(packets) to flow in parallel without interfering with each other. The added lanes/channels are virtualized since they do not physically exist but appear on the one physical channel in a time-multiplexed manner [89] as shown in figure 7.1.
Allowing for flow separation and isolation needs the dedication of multiple resources either

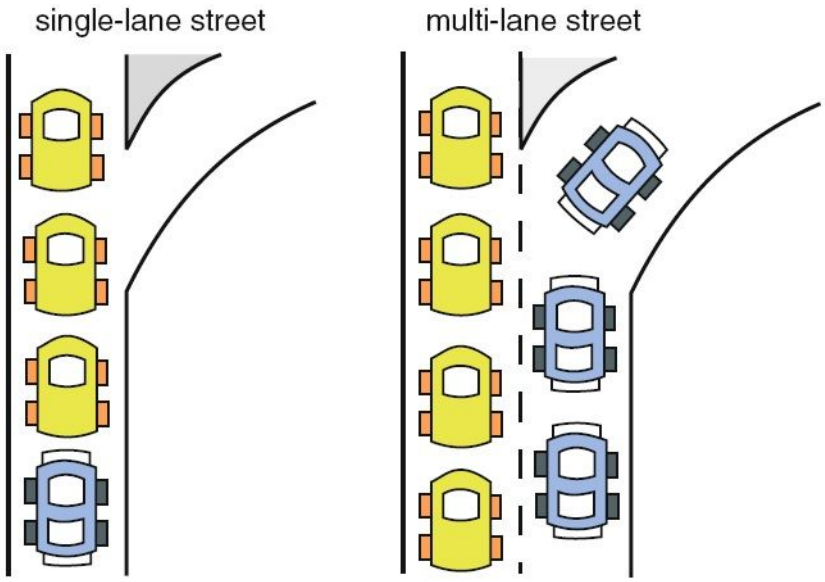


FIGURE 7.1: Illustration of Virtual Channel use [89]

in space (more physical lanes by adding extra wires on the links) or in time (more virtual resources interleaved on the same physical resources in a well defined manner). This chapter deals with virtual channel allocation that represent an efficient flow control mechanism for adding lanes to a street network in an efficient and versatile manner, as illustrated in figure 7.1. Adding virtual channels to the network removes the constraints that appear in single-lane streets and allow otherwise blocked packets to continue moving by just turning to an empty (less congested) lane of the same street [24, 146]. Since the additional lanes are virtually existent their implementation involves the time multiplexing of the packets that belong to different lanes (virtual channels) on the same physical channel. Briefly, virtual channels behave similar to having multiple wormhole channels present in parallel. However, adding extra lanes (virtual channel) to each link does not add bandwidth to the physical channel. It just enables better sharing of the physical channel by different flows [147-148].

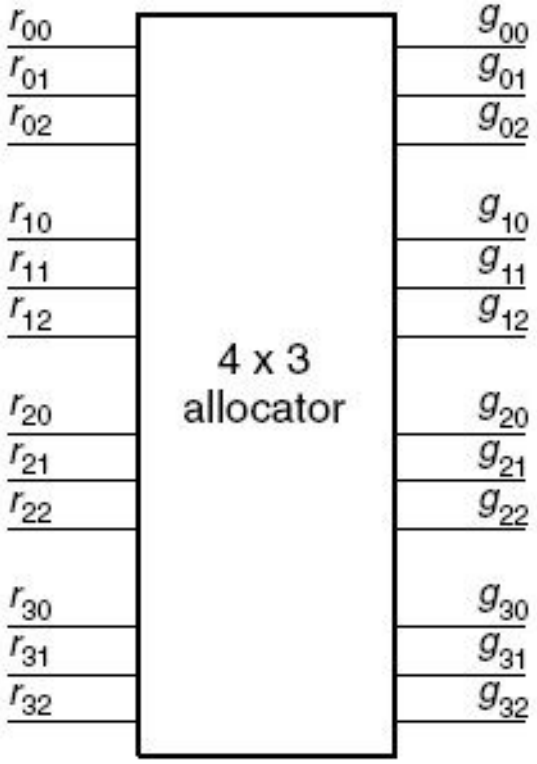
Architectures supporting the use of VCs may reduce also on-chip physical routing congestion, by trading off physical channel width with the number of VCs, thereby creating a more layout-flexible SoC architecture. Instead of connecting two nodes with many parallel links that are rarely used at the same time, one link can be used instead that supports virtual channels, which allows the interleaving in time of the initial parallel traffic, thus saving wires and increasing their utilization.
The switch allocator is responsible for determining in each cycle the connections between the input and the output ports of the switch. Since now the input buffers are organized in multiple-independent queues, each input can send multiple requests per clock cycle.
The grant signals generated by the switch allocator are used to set up the registers that control crossbar connectivity. In addition, the switch allocator notifies the winning VC at each input port, causing the latter to prepare its frontmost flit for crossbar traversal.

7.2.1 Related Work

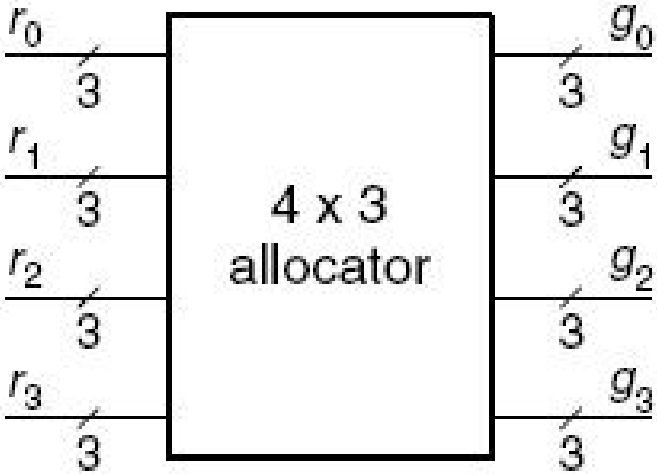
VC allocation in interconnection networks has been addressed in a number of prior research contributions: Peh and Dally [149-150] present an analytical delay model for separable VC allocators. Their model is derived from gate-level schematics using the logical effort method [151]; as such, it is primarily geared towards full-custom implementations that optimize for minimum critical path delay regardless of the implications for area and power. Furthermore, the models do not account for wire delay, which has become a critical factor in modern submicron semiconductor processes. Mullins et al. [97] propose a technique for reducing the delay of separable input-first VC allocators by precomputing arbitration decisions and by using a free VC queue at each output port, the front-most element of which is assigned to incoming requests. Kumar et al. [152] describe a scheme that combines VC and switch allocation into a single step. Finally, Zhang and Choy [153] investigate approaches for reducing the complexity of separable VC allocators based on utilization statistics for their individual constituent arbiters and present detailed delay, area and power results.
Switch allocation as a means of improving network performance was originally proposed by Peh and Dally [149-150]. The authors present an analytical delay model for a separable implementation, but do not investigate area or power; since the model does not account for wire delay, its results are overly optimistic for modern process technologies. Mukherjee et al [154] compare the performance of several switch allocator implementations in the context of a system-level interconnection network. Their study assumes that routers are deeply pipelined, and two of the allocators considered require multiple cycles to produce each matching; both factors would be undesirable in latency-sensitive NoCs. Furthermore, the different allocators are compared purely in terms of network performance; in particular, the analysis does not consider area or power, both of which represent first-class design considerations in NoCs. Mullins et al. [97] reduce the pipeline latency of a VC router by precomputing arbitration decisions in a separable allocator one cycle in advance. While this scheme effectively removes the switch allocation stage from the router's critical path for buffered flits, it is less effective in the absence of congestion, as conflicts between newly arriving flits can result in unused crossbar time slots. The row/column decoupled router introduced by Kim et al. [155] features an efficient mirror allocation scheme; however, this scheme is not directly applicable to generic router designs. Kumar et al. [152] describe a scheme for combined VC and switch allocation that dynamically transitions between input- and output-first operation based on network load. Their design explicitly prioritizes subsequent flits from the same packet, and uses FIFO queues to implement both input-side arbitration and VC selection.

7.3 Fundamentals of Implemented Allocators

While an arbiter assigns a single resource to one of a group of requesters, an allocator performs a matching between a group of resources and a group of requesters, each of which may request one or more of the resources.



(A) Symbol for an allocator with individual request and grant lines shown



(B) Symbol with bundled requests and grants

FIGURE 7.2: An n $\times$ m allocator accepts *n m-bit* request vectors and generates *n m-bit* grant vectors [7].

An $n \times m$ allocator is a unit that accepts *n m-bit* vectors as inputs and generates *n m-bit* vectors as outputs, as shown in figure 7.2 for an n = 4 by m = 3 allocator. When request input r_{ij} is asserted, requester *i* wants access to resource *j*. Each requester can request any subset of the resources at the same time. For allocators used in router designs, the requesters often correspond to switch inputs and the resources correspond to switch outputs. So, we will usually refer to the requesters and resources as inputs and outputs, respectively.
The allocator considers the requests and generates the grant vectors subject to three rules:

1. $g_{ij} \Rightarrow r_{ij}$, a grant can be asserted only if the corresponding request is asserted.
2. $g_{ij} \Rightarrow \neg\, g_{ik}\ \forall$ k $\neq$ j, at most, one grant for each input (requester) may be asserted.
3. $g_{ij} \Rightarrow \neg\, g_{ik}\ \forall$ k $\neq$ i, at most, one grant for each output (resource) can be asserted.

The allocator can be thought of as accepting an n $\times$ m request matrix *R* containing the individual requests, r_{ij} and generating a grant matrix *G* containing the individual grants, g_{ij}. *R* is an arbitrary binary-valued matrix. *G* is also a binary valued matrix that only contains ones in entries corresponding to non-zero entries in *R*, has at most one one in each row, and at most one one in each column.

7.3.1 Separable Allocators

Most investigative allocators are based on a basic separable allocator. In a separable allocator, we perform allocation as two sets of arbitration: one across the inputs and one across the outputs. This arbitration can be performed in either order. In an input-first separable allocator, an arbitration is first performed to select a single request at each input port. Then, the outputs of these input arbiters are input to a set of output arbiters to select a single request for each output port. The result is a legal matching, since there is at most one grant asserted for each input and for each output. However, the result may not even be maximal, let alone maximum. It is possible for an input request to win the input arbitration, locking out the only request for a different output, and then lose the output arbitration. This leaves an input and an output, which could have been trivially connected, both idle.

7.3.1.1 Input-first Separable Allocator

A 4 $\times$ 3 input-first separable allocator is shown in figure 7.3. Each input port has separate request lines for each output. For example, for a flit at input 2 to request output 1, request line r_{21} is asserted. The first rank of four three-input arbiters selects the winning request for each input port. Only one of the signals x_{ij} will be asserted for each input port *i*. The results of this input arbitration, the signals x_{ij} , are forwarded to a rank of three 4-input output arbiters, one for each output. The output arbiters select the winning request for each output port and assert the grant signals g_{ij} . The output arbiters ensure that only one grant is asserted for each output, and the input arbiters ensure that only one grant is asserted for each input. Thus, the result is a legal matching.

7.3.1.2 Output-first Separable Allocator

A separable allocator can also be realized by performing the output arbitration first and then the input arbitration. A 4 $\times$ 3 output-first separable allocator is shown in figures 7.4. In this case, the first rank of three 4-input arbiters selects the winning request for each output port. Only one of the resulting signals y_{ij} will be asserted for each output port *j* . The four 3-input

input arbiters then take y_{ij} as input, pick the winning request for each input, and output this result on grant signals, g_{ij} ensuring that at most one of g_{ij} is asserted for each input i.

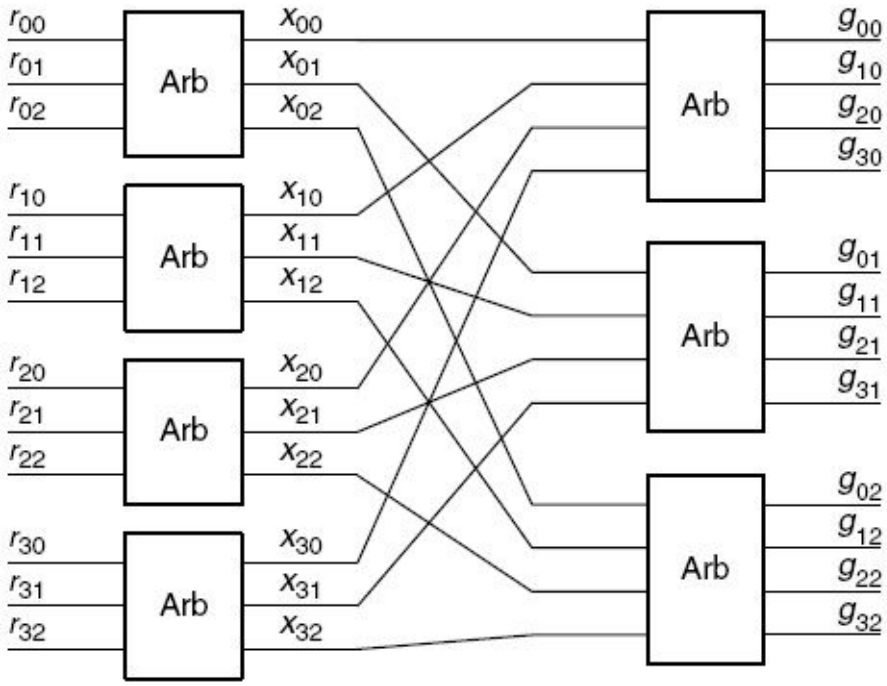


FIGURE 7.3: A 4 × 3 input-first separable allocator [7]

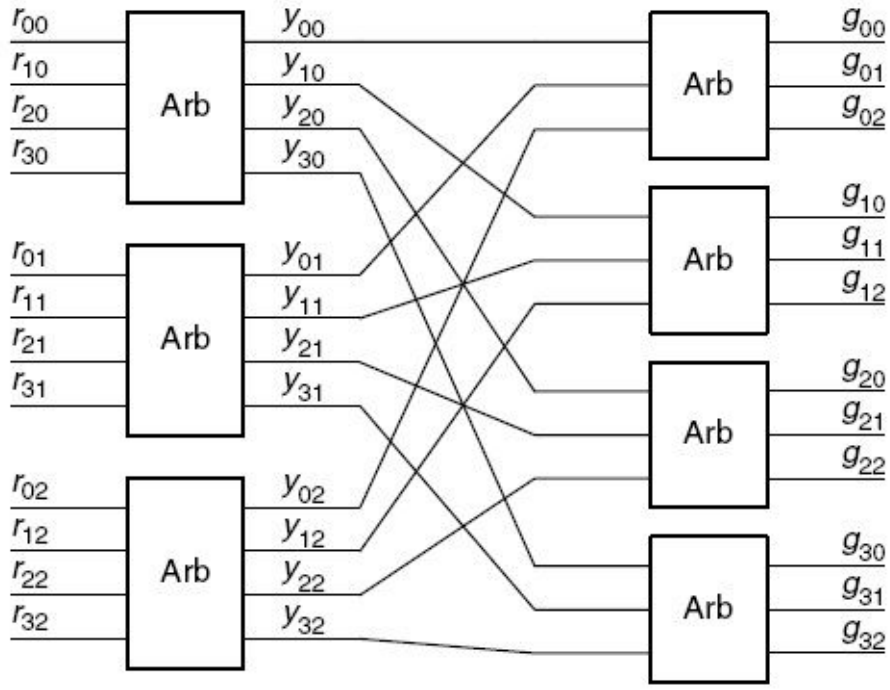


FIGURE 7.4: A 4 × 3 output-first separable allocator [7]

7.3.2 Wavefront Allocator

The wavefront allocator, unlike the separable allocators described above, arbitrates among requests for inputs and outputs simultaneously. The structure of the wavefront allocator is shown in Figure 7.5 and the logic of each allocator cell is show in Figure 7.6.

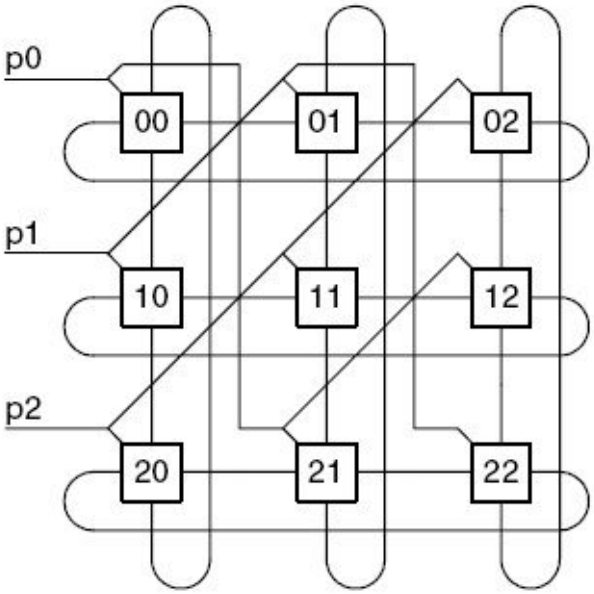


FIGURE 7.5: A wavefront allocator

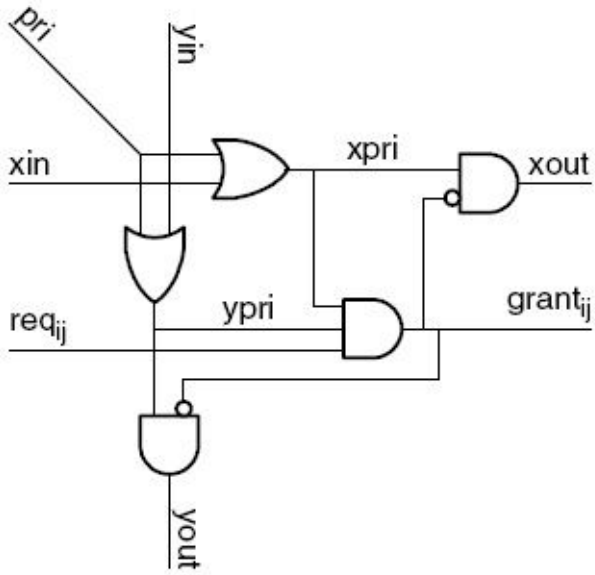


FIGURE 7.6: Logic diagram of a wavefront allocator cell

The wavefront allocator works by granting row and column tokens to a diagonal group of cells, in effect giving this group priority. A cell with a row (column) token that is unable to use the token passes the token to the right (down), wrapping around at the end of the array. These tokens propagate in a wavefront from the priority diagonal group, hence the name of the allocator. If a cell with a request receives both a row and a column token, either because

it was part of the original priority group or because the tokens were passed around the array, it grants the request and stops the propagation of tokens. To improve fairness, the diagonal group receiving tokens is rotated each cycle. However, this only ensures weak fairness.
In an n $\times$ n arbiter, diagonal group *k* contains cells x_{ij} such that *(i+j) mod n = k*. Thus, for example, in the 3 $\times$ 3 allocator of figure 7.5, priority group 0, selected by signal *p0*, consists of cells *x00*, *x21*, and *x12*. Because each diagonal group must contain exactly one cell from each row and from each column, all wavefront allocators must be square. A non-square wavefront allocator can be realized, however, by adding dummy rows or columns to square off the array. For example, the 4 $\times$ 3 allocator of our examples above can be realized using a 4 $\times$ 4 array. The priority groups in an allocator need not be diagonals as long as they each contain one element of each row and column.
The details of the allocator cell are shown in figure 7.6. When the cell is a member of the priority group, signal *pri* is asserted, which generates both a row token *xpri* and a column token *ypri* via a set of *OR* gates. If the cell is not a member of the priority group, row tokens are received via signal *xin* and column tokens via *yin*. If a cell has a row token *xpri*, a column token *ypri* and a request req_{ij} , it generates a grant $grant_{ij}$ via a 3-input *AND* gate. If a grant is asserted, it disables further propagation of the row and column tokens via a pair of *AND* gates. Otherwise, if a grant is not asserted, row (column) tokens are passed to the next cell to the right (down) via *xout* (*yout*).

7.4 Implementations and Results

7.4.1 Virtual Channel Allocator

In the most general case, during virtual-channel allocation, the head flits that are buffered at any input VC must compete for an available downstream VC, which belongs to the output port that is selected by the routing function. Thus, the VC allocator performs a matching between N$\times$V requestors and N$\times$V resources, subject to the constraint that any output VC is requested only by input VCs that need to be forwarded to the same output port. This allocation problem can be solved either by using a centralized approach, or by adopting a VC separable allocator with separate per-input and per-output arbiters.

7.4.1.1 Separable Virtual-channel Allocator

In this case, the organization of a virtual-channel allocator follows roughly the same structure as the separable switch allocator. Each VC of each input port tries to get access to a specific output port. Since more than one VC is associated with each output port, each input VC should decide whether it should request all of them at once or if it should select only a subset of the available output VCs. Its decision is guided solely by the routing function. In any case, the design of the virtual-channel allocator requires one arbiter per VC per output port. Thus, N$\times$V arbiters are required in total at the outputs, with each one accepting N$\times$V requests.
Depending on how many output VCs each input VC is allowed to request two alternatives are possible. In the case of input first allocation, each input VC selects only one of the available VCs returned by the routing function for the selected output, as shown in figure 7.7. Then, it is up to the output arbiters to grant one of the requesting input VCs. In the symmetric case of output-first allocation, shown in figure 7.8, each input VC requests all available VCs that belong to the output selected by the routing function. Since each input VC asks for more than one output VC it is possible that it receives a grant from more than one output arbiter. Thus, an additional stage of arbitration is needed to decide which grant to accept.

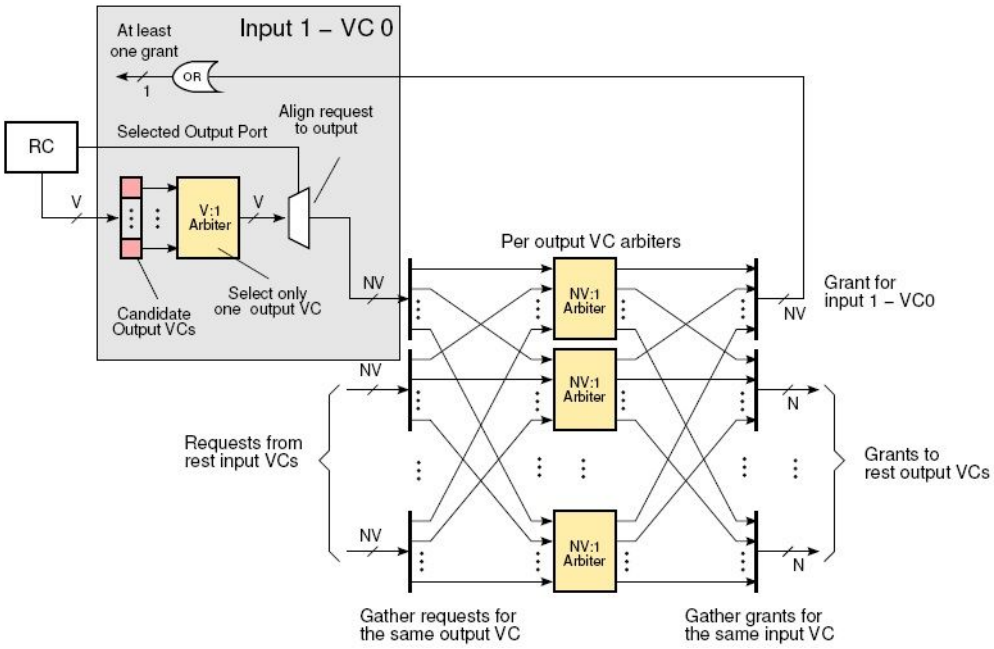


FIGURE 7.7: Input-first separable virtual channel allocator [156]

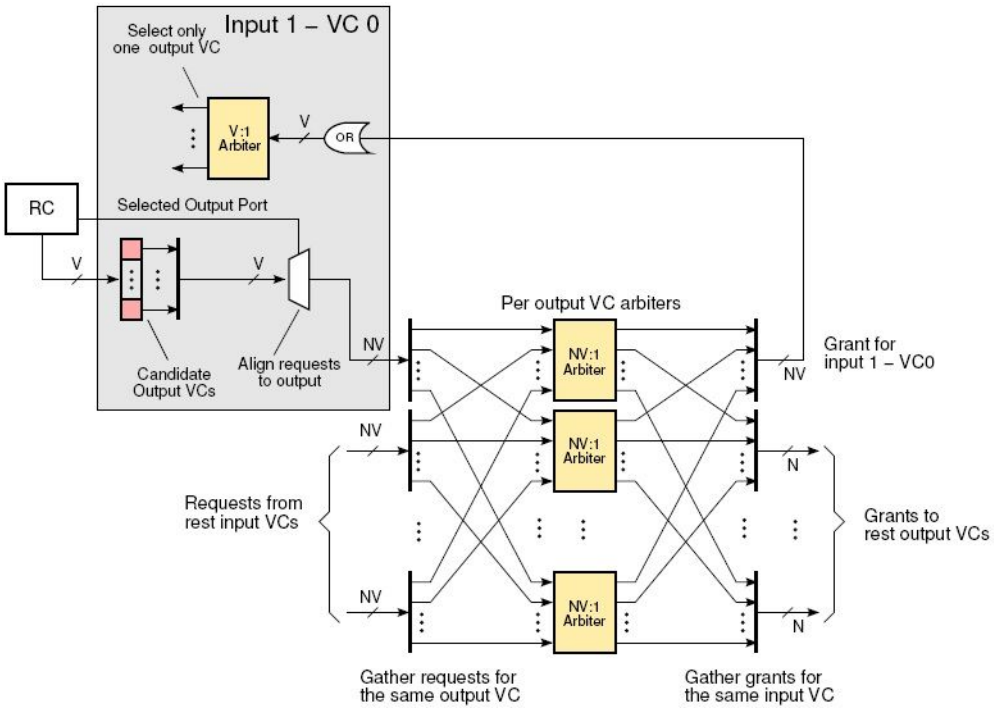


FIGURE 7.8: Output-first separable virtual channel allocator [156]

We have developed a parameterized RTL model using Verilog HDL and synthesized it in a commercial FPGA design flow. Synthesis is performed using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. In this sub-section we have implemented virtual channel controller in two ways as using separable input-first allocation and using separable output-first allocation . A series of figures (figure 7.9 - 7.10) and tables (table 7.1-7.2) show the synthesis report as well as power analysis report.

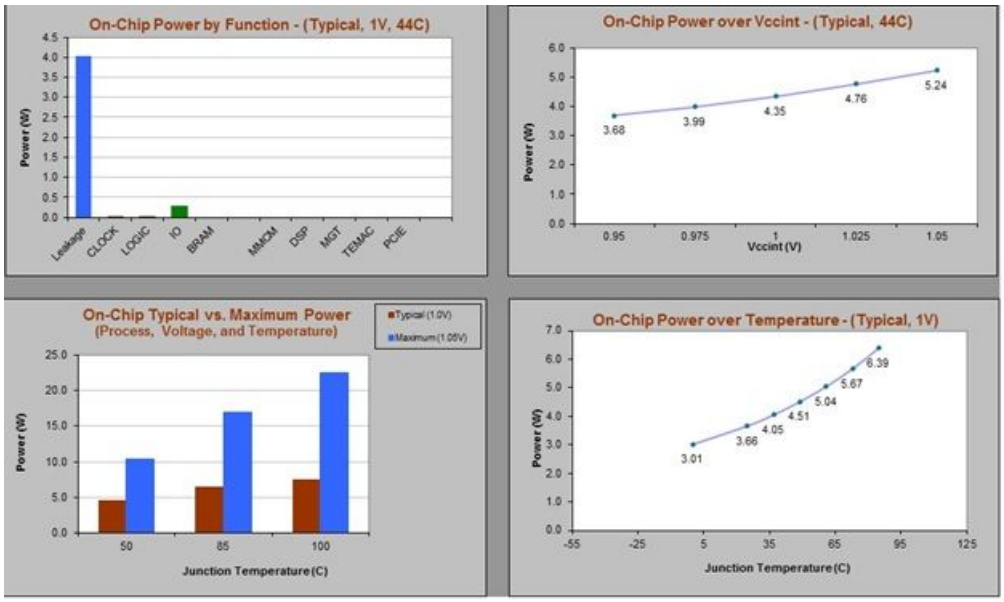


FIGURE 7.9: Power report of virtual channel allocator using separable input-first allocation

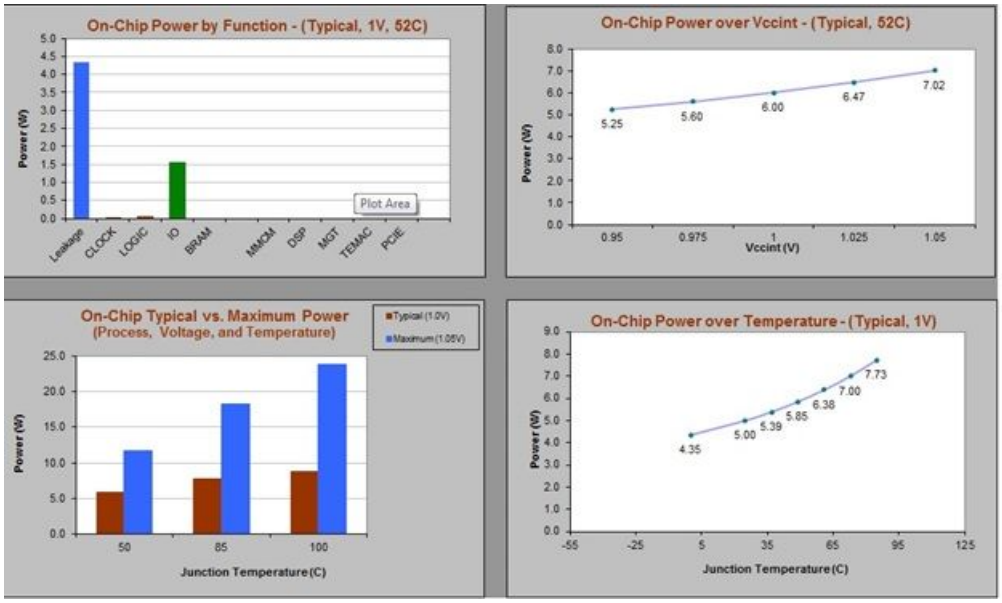


FIGURE 7.10: Power report of virtual channel allocator using separable output-first allocation

Input-first allocation provides slightly better matching than output-first allocation, as it performs the narrower V : 1 arbitration at the input side before the wider P×V : 1 arbitration at the output side; because an *n*-input arbiter grants only one of up to n requests, eliminating up to *n-1* others, this allows more requests to be propagated from the first arbitration stage to the second one than in the output-first case.
When separable VC allocation returns poor matchings, we could choose to perform multiple iterations as in the case of switch allocation. Nevertheless, as shown in [157], this option is not needed since the matching quality of the VC allocator does not determine the network performance of the switch.

TABLE 7.1: Synthesis report of virtual channel allocator using separable input-first allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
8x5 bit single-port RAM : 20	6.894 ns	6.182 W
1-bit register : 100	0.555 ns logic, 6.339 ns route	
3-bit register : 20	8.1% logic, 91.9% route	
1-bit 2-to-1 mux : 100		
2-bit 2-to-1 mux : 100		
3-bit 2-to-1 mux : 20		
5-bit 2-to-1 mux : 20		
Finite State Machine = 1		

TABLE 7.2: Synthesis report of virtual channel allocator using separable output-first allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
8x5 bit single-port RAM : 20	5.063 ns	7.73 W
1-bit register : 160	0.494 ns logic, 4.569 ns route	
3-bit register : 20	9.8% logic, 90.2% route	
2-bit 2-to-1 mux : 100		
3-bit 2-to-1 mux : 20		
5-bit 2-to-1 mux : 20		

7.4.1.2 Wavefront Virtual-channel Allocator

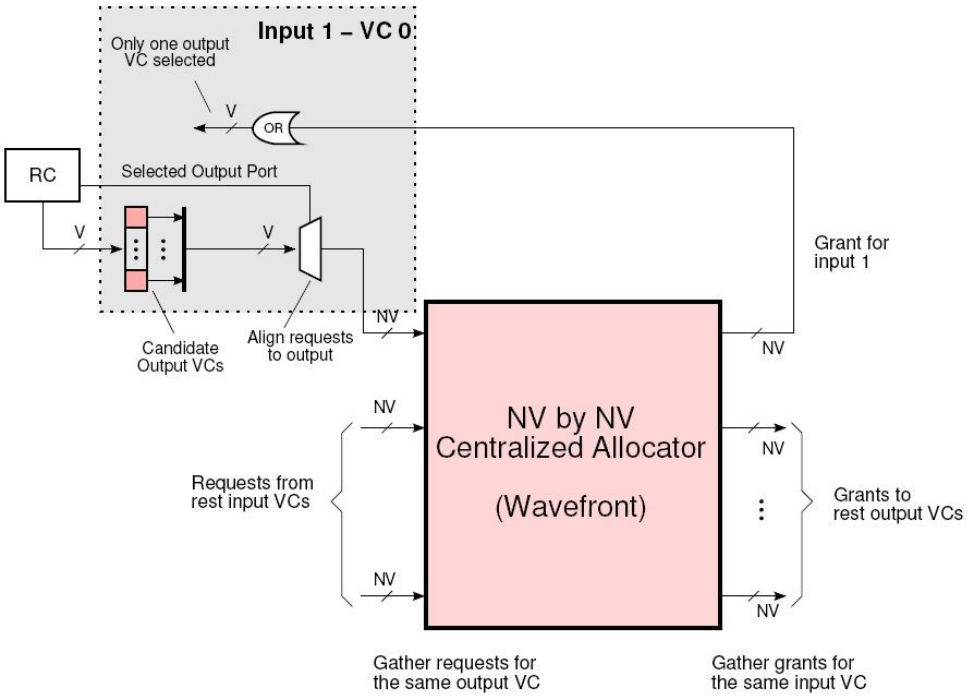


FIGURE 7.11: Wavefront virtual-channel allocator

In a Wavefront virtual-channel allocator the requests of each input VC are expanded to a N $\times$ V-wide request vector depending on the output port selected by the packet and the choices enforced by the routing logic, as shown in figure 7.11. The wide request vector cannot have more than V active requests, while all requests concern VCs of the same output. After a matching between input and output VCs is developed, the $\times$ grant signals coming from all output VCs are gathered per-input VC. The OR function at each input VC just detects whether at least one of the V returning lines corresponds to a grant.
As shown in figure 7.11, a wavefront-based VC allocator consists of a canonical P$\times$V -input wavefront allocator, with additional logic for generating the P$\times$V -wide request vector for each input VC as in the separable output-first case, and for reducing the P$\times$V -wide grant vectors to V -wide vectors as in the input-first case. Availability masking can be performed at the inputs to the wavefront allocator, while output-side grants are generated by combining the grant signals for all P$\times$V input VCs for each individual output VC.
We have developed a RTL model using Verilog HDL and synthesized it in a commercial FPGA design flow. Synthesis is performed using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. In this sub-section we have implemented wavefront virtual channel controller and figure 7.12 shows the power analysis report and tables 7.3 shows synthesis report.

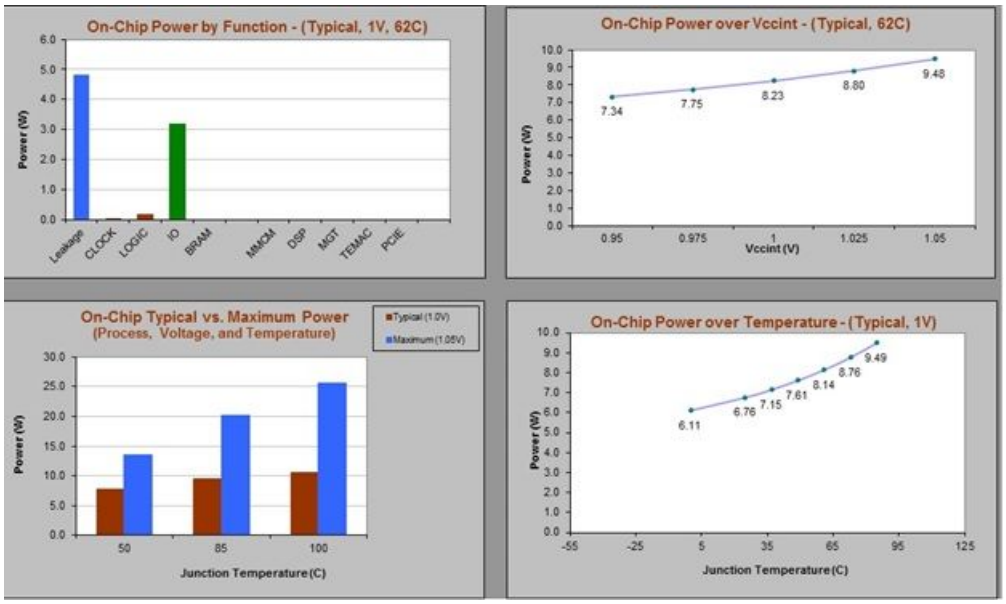


FIGURE 7.12: Power report of virtual channel allocator using wavefront allocation.

TABLE 7.3: Synthesis report of virtual channel allocator using wavefront allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
16x10 bit single-port RAM : 20	10.814 ns	9.49 W
1-bit register : 160	1.131 ns logic, 9.683 ns route	
4x4-bit multiplier : 2	10.5% logic, 89.5% route	
8x4-bit multiplier: 2		
4-bit register : 2		
4-bit 2-to-1 mux : 2		

Continued on next page

Table 7.3 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1000-bit shifter logical left : 2		

For all design points considered, results show that the separable input-first implementation achieves better delay, area and power dissipation than the other implementations under investigation. As the choice of VC allocator does not significantly affect network performance, it thus represents the preferable overall design choice for this use case.

7.4.2 Switch Allocator

The switch allocator is responsible for determining in each cycle the connections between the input and the output ports of the switch. Since now the input buffers are organized in multiple-independent queues, each input can send multiple requests per clock cycle.
Once a packet has completed Virtual Channel (VC) allocation, its flits can be forwarded to the selected destination port subject to buffer space availability. For each flit to be transferred, a crossbar connection between the corresponding input and output ports must be established for one cycle. The switch allocator is responsible for scheduling such crossbar connections; in particular, it generates matching between requests from active VCs at each of the router's P input ports on the one hand and crossbar connections to its P output ports on the other hand. The quality of the generated matchings directly affects the router's latency and throughput under load.
With VC flow control, flits may only be sent downstream if sufficient buffer space is available at the receiving router. To this end, routers maintain a set of credit counters at each output port that track the number of available buffer entries for each downstream VC. A given input VC can only request access to the crossbar if its destination VC has at least one credit available.
The grant signals generated by the switch allocator are used to set up the registers that control crossbar connectivity. In addition, the switch allocator notifies the winning VC at each input port, causing the latter to prepare its front-most flit for crossbar traversal e.g., by initiating a read access to the input buffer and to decrements its credit count proxy. Finally, the output side credit counter for each winning flit's destination VC is updated to reflect the fact that a credit has been consumed.

7.4.2.1 Separable Switch Allocators

Separable switch allocation is organized in two phases since both a per-output and a per-input arbitration step is needed [158]. An example organization is shown in figures 7.13 and 7.14. In the first case, each input is eligible to send to the outputs only one request (figure 7.13). In order to decide which request to send, each input arbitrates locally among the requests of each VC to forward their requests to the output arbiters (figure 7.14).
In this way, it is possible that two or more VCs of the same input will receive a grant from different outputs. However, only one of them is allowed to pass its data to the crossbar. Therefore, a local arbitration needs to take place again that will resolve the conflict.
In either form of separable switch allocation, in order to ensure fairness and deterministic service the priority vector of any arbiter should be updated only if the grant that it produced is also accepted in the second arbitration stage. Appropriate priority selection can also increase switch allocator's throughput by biasing the input/output pairs that correspond to heavily backlogged flows. In this way, flits of the same packet (or flows in general) are kept together as much as possible. Low-latency approaches that follow this principle, that are also easy to

implement in the context of NoCs, have been presented in [152] and [159].

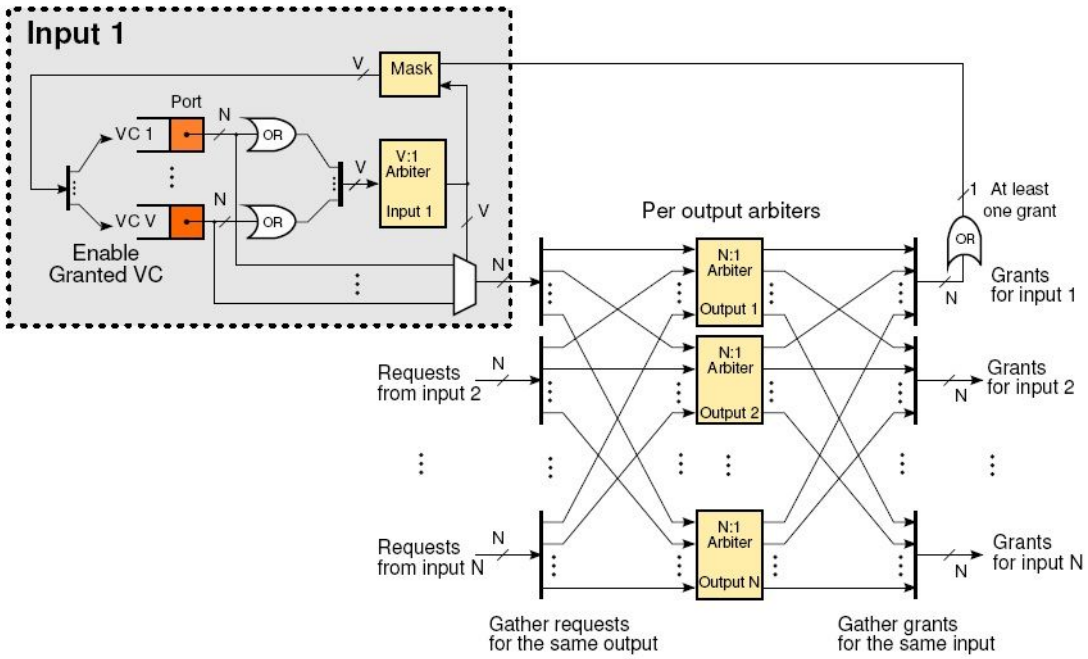


FIGURE 7.13: Input-first separable switch allocator

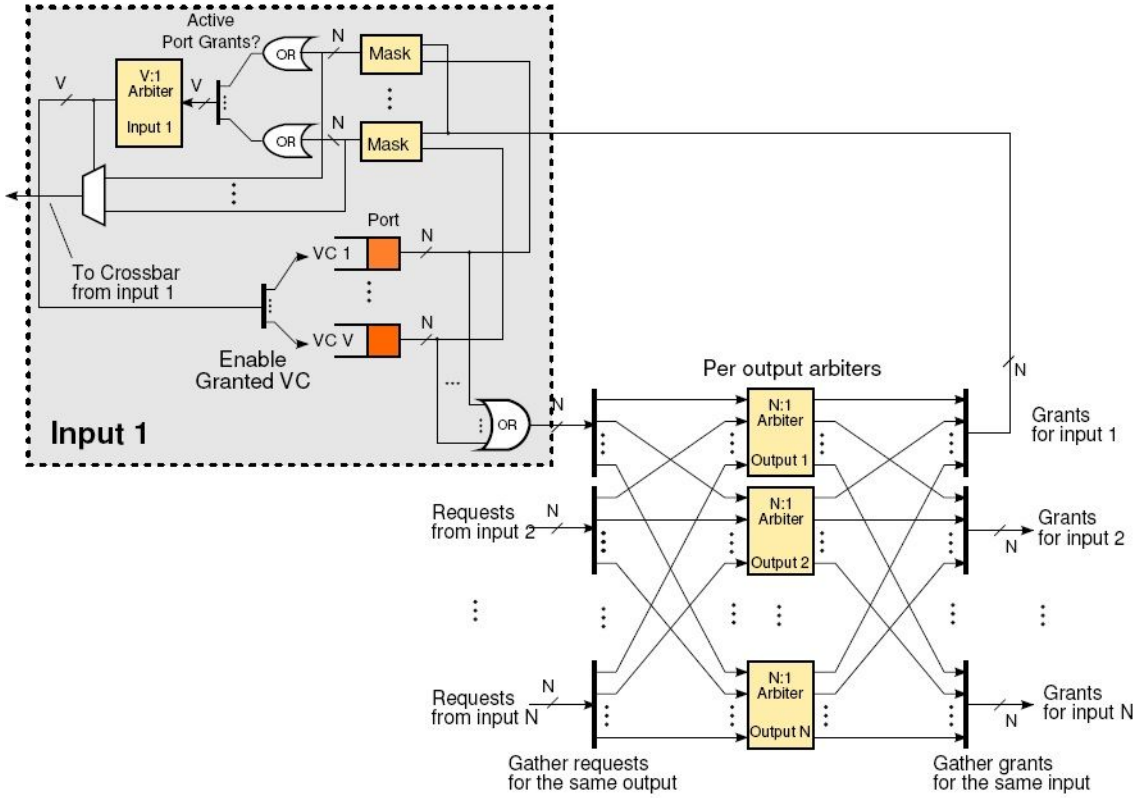


FIGURE 7.14: Output-first separable switch allocator

Even though the arbiters operate independently, their eventual outcomes in switch allocation are very much dependent. For example, when two or more input arbiters request the same output simultaneously, a conflict occurs. Only one input will be matched, while others will be left idle. There is nothing better these idle inputs could have done if they had HOL (Head of Line blocking) flits only for the requested output. However, it would have been much better for them and for the collective throughput of the switch if they had tried to bind to some other output. Such conflicts are unavoidable, when every port has an opportunity of only one request in every clock cycle. In order to improve their efficiency, separable switch allocators have two generic options. They can either try to "desynchronize" their bindings, so that each input (output) requests a different output (input) on every new scheduling cycle, or they can attempt multiple requests per port, per scheduling operation.
Similarly as virtual channel, we have developed a RTL model using Verilog HDL and synthesized it in a commercial FPGA design flow. Synthesis is performed using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. In this sub-section we have implemented virtual channel controller in two ways as using separable input-first allocation and using separable output-first allocation . A series of figures (figure 7.15 - 7.16) and tables (table 7.4 - 7.5) show the synthesis report as well as power analysis report.

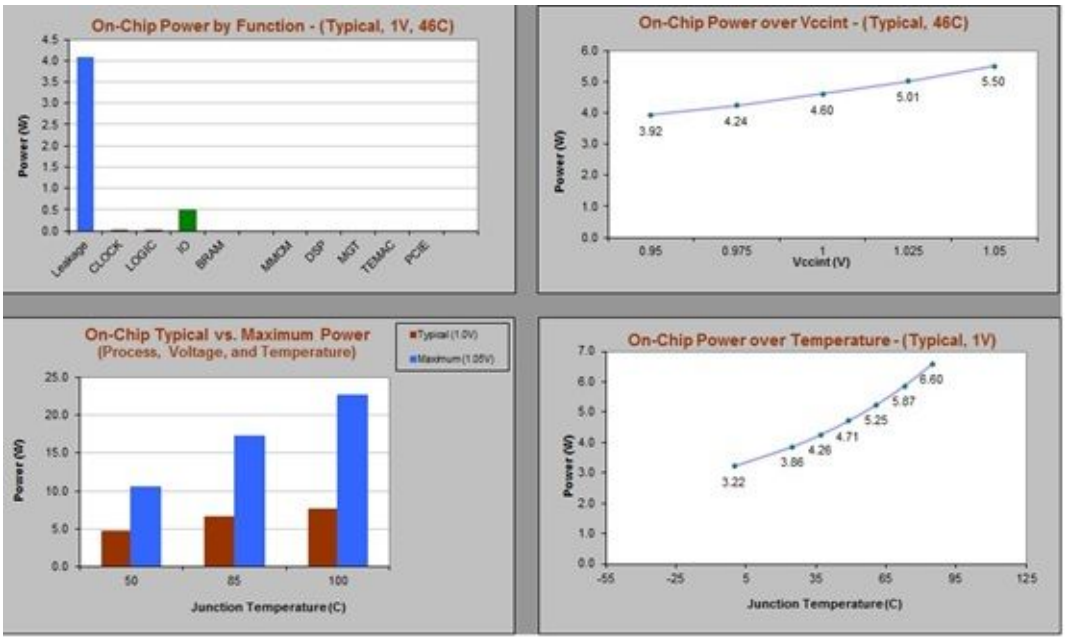


FIGURE 7.15: Power report of switch allocator using separable input-first allocation.

TABLE 7.4: Synthesis report of switch allocator using separable input-first allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
4x4 bit single-port RAM : 10	6.393 ns	6.60 W
8x5 bit single-port RAM : 10	0.555 ns logic, 5.838 ns route	
Flip-Flop : 50	8.7% logic, 91.3% route	
4-bit register : 2		
4-bit 2-to-1 mux : 2		

Continued on next page

Table 7.4 – *Continued from previous page*

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
1000-bit shifter logical left : 2		

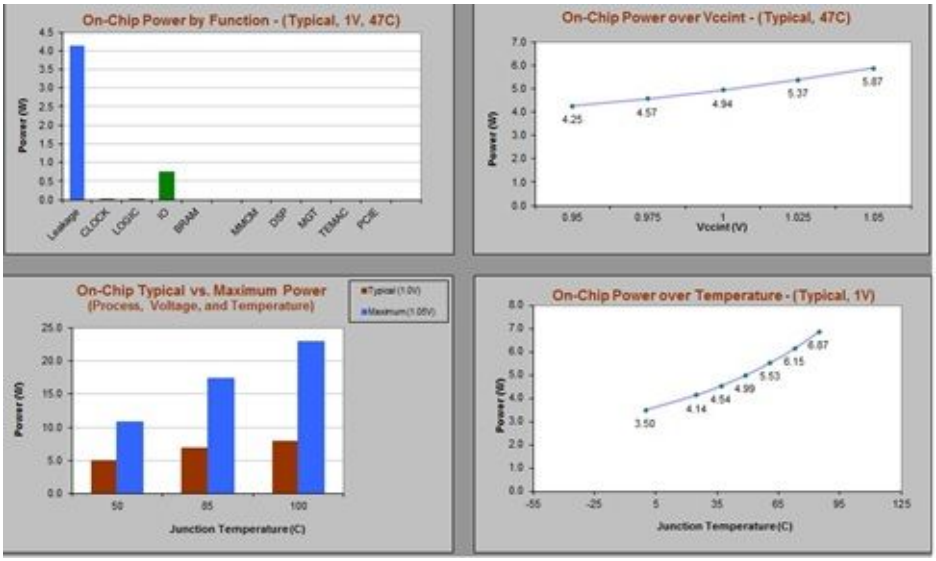


FIGURE 7.16: Power report of switch allocator using separable output-first allocation.

TABLE 7.5: Synthesis report of switch allocator using separable output-first allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
4x4 bit single-port RAM : 10	7.921 ns	6.87 W
8x5 bit single-port RAM : 10	0.799 ns logic, 7.122 ns route	
2-bit register : 10	10.1% logic, 89.9% route	
3-bit register : 10		
2-bit 2-to-1 mux : 10		
3-bit 2-to-1 mux : 10		
4-bit 2-to-1 mux : 15		
5-bit 2-to-1 mux : 10		

7.4.2.2 Wavefront Switch Allocators

A centralized switch allocator, such as the wavefront arbiter or the 2-D round-robin, handles all input requests at the same time and produces a global schedule for the whole crossbar. The schedule is maximal when it is not possible to insert a new input/output connection, without altering some of the connections that already exist in it. In general, centralized switch allocators produce maximal schedules and do not leave any outputs unoccupied if there is a request for them. The main drawback of the commonly used centralized schedulers is their delay.
The simplest form of a centralized switch allocator with N input/output ports receives directly the requests of all input VCs, that is N $\times$ V in total, and grants only N of them; one for each output. The scheduler in this case is asymmetric since it shares N outputs among N $\times$ V requests. In this way, we may end up granting two or more VCs from the same input that share a common crossbar port but have flits for different outputs. To avoid this conflict the

granted VCs of the same input pass from an additional V : 1 arbiter that selects only one of them.

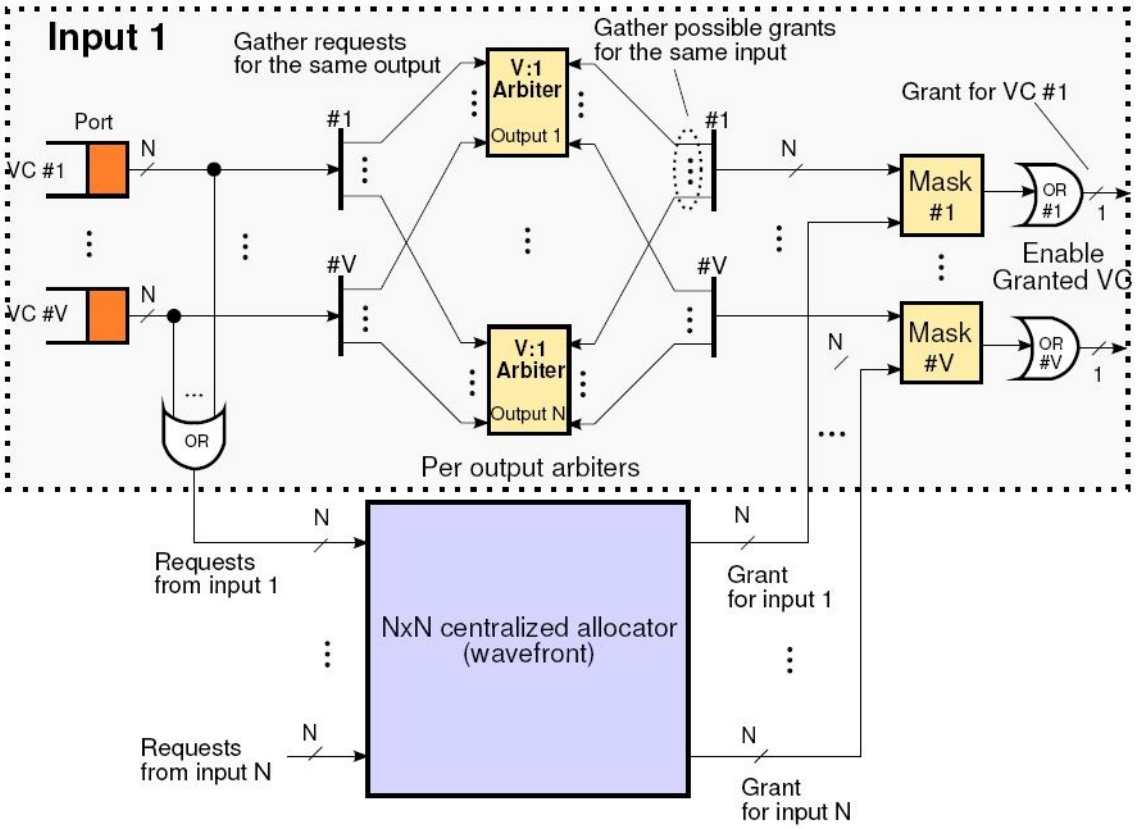


FIGURE 7.17: Wavefront switch allocator.

Of course, this per-input local arbitration stage (local V : 1 arbiter) could have been done at the beginning of switch allocation. Then, each input would transmit only one request to a N × N centralized allocator that again would produce one grant for each output. This grant can be consumed directly by the input port since the eligible VC is preselected.
Following this analysis, it is evident that both techniques besides the centralized allocator need an additional local arbiter for resolving output contention. In the baseline implementation this arbiter lies in the critical path of the switch allocator. However, based on the observation of [157] such a constraint can be removed, if we rely on a centralized allocator that gives to each input at most one grant. No grants are given to the same input for two distinct outputs; this constraint is already satisfied by the wavefront allocator. This faster alternative is depicted in figure 7.17.
At first, each input requests all outputs for which it has HOL flits irrespective of the input VCs that they belong to. The requests are received by a N × N centralized allocator that grants N requests, one for each output. Therefore, at this point we know the output to which each input will be connected to. The only thing that remains to be selected is the appropriate VC from each input. The good news is that this choice can be performed in parallel to the centralized allocator, using N V : 1 arbiters. Each arbiter corresponds to an output and selects one candidate VC for this output. At the end, from all derived (output, VC) pairs at each input, only one would be chosen according to the schedule of the wavefront allocator.
We have developed a RTL model using Verilog HDL and synthesized it in a commercial FPGA design flow. Synthesis is performed using Xilinx ISE 14.7 and FPGA Virtex 6 family

device XC6VLX760 is considered as target technology. In this sub-section we have implemented wavefront switch allocator and figure 7.18 shows the power analysis report and tables 7.6 shows synthesis report.

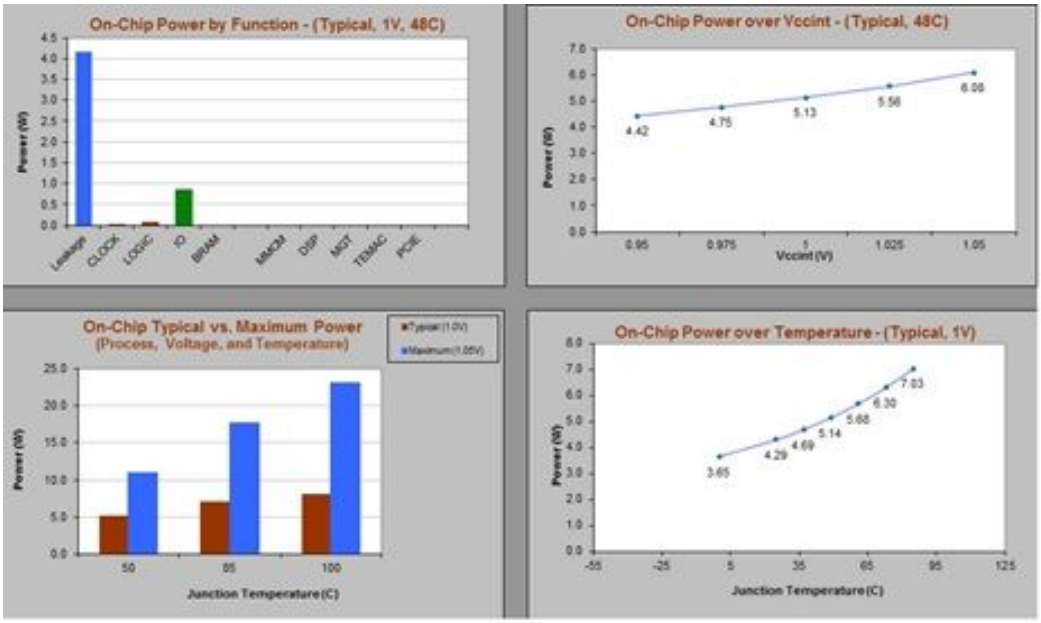


FIGURE 7.18: Power report of Switch allocator using wavefront allocation.

TABLE 7.6: Synthesis report of switch allocator using wavefront allocation.

Device Macro Statistics	*Timing Parameter*	*Power Dissipation*
4x4 bit single-port RAM : 10	10.598 ns	7.03 W
8x5 bit single-port RAM : 02	0.982 ns logic, 9.616 ns route	
3x3-bit multiplier : 2	9.3% logic, 90.7% route	
6x3-bit multiplier : 2		
Flip-Flops : 26		
2-bit 2-to-1 mux : 10		
3-bit 2-to-1 mux : 02		
4-bit 2-to-1 mux : 15		
125-bit shifter logical left : 2		

7.5 Conclusion

In the present chapter, we have discussed the key aspects of allocation and provided detailed descriptions of representative hardware implementations.

Additionally, we have explored the design space for VC allocators in the context of NoC routers. In particular, we have presented practical hardware implementations for three exemplary VC allocator architectures based on the elementary designs. Overall, our results suggest that the separable input-first implementation provides the optimal delay, area and energy efficiency among the three designs considered in the present chapter.

we have also evaluated three exemplary switch allocator implementations and investigated several approaches for reducing the router's pipeline delay.

Chapter 8

Conclusion and Future Work

The Multi-Synchrnous Bi-directional NoC (MBiNoC) architecture, as presented in this thesis, can dynamically self-reconfigure its channel direction based on real-time traffic needs. The basic concept is to utilize the waste idle bandwidth by temporarily reversing the channel direction in MBiNoC to relieve the congestion in the opposite traffic direction.
In this thesis, we clearly described the bandwidth utilization problem of conventional NoC design and proposed a novel concept of using bidirectional channel which can be adjusted based on real-time traffic requirement. A novel router architecture named MBiNoC is provided with virtual-channel flow-control based mechanism. Then, an inter-router transmission scheme is provided first to achieve bidirectional data transmission. To avoid deadlock and starvation, a priority-based design of ASM is introduced.

In this thesis we considered the analysis and implementation of Multi-Synchronous Bi-Directional NoC Architecture with Dynamic Self Reconfigurable Channel for the GALS Infrastructure. To enhance the performance of on-chip communications of Globally Asynchronous Locally Synchronous Systems (GALS), a dynamic re-configurable multi-synchronous router architecture is proposed to increase network on chip (NoC) efficiency by changing the path of the communication link in the runtime traffic situation. In order to address GALS issues and bandwidth requirements, the proposed multi-synchronous bi-directional NoC's router is developed and it guarantees higher packet consumption rate, better bandwidth utilization with lower packet delivery latency. All the input/output ports of the proposed router behave as a bidirectional ports and communicate through a novel multi-synchronous first-in first-out (FIFO) buffer. The bi-directional port is controlled by a dynamic channel control module which provides a dynamic reconfigurable channel to the router itself and associated sub-modules. This proposed multi-synchronous bi-directional router architecture is synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology and its performance is evaluated in terms of power, area and delay.

In Chapter 1 we introduced the Networks on Chip main characteristics and explained motivations making NoC a promising solution for future System on Chip interconnects. In this chapter we also discussed State-of-Art of traditional NoC's architectures and Bi-directional NoC's architectures with objectives of thesis.

In Chapter 2 we discussed architectural design of proposed router and contributions to the available bi-directional router's architecture. All input/output ports of the proposed router behave as a bidirectional ports and communicated through a novel multi-synchronous first-in first-out (FIFO) buffer. The bi-directional port is controlled by a dynamic channel control module which provides a dynamic reconfigurable channel to the router itself and associated sub-modules. The flow direction at each channel is controlled by a dynamic channel control algorithm. Implemented with a pair of algorithmic finite state machines, this dynamic

channel control algorithm provides high performance, is free of deadlocks, and is starvation-free. This chapter describes in detail the bidirectional channel control scheme and its design mechanism. An inter-router transmission scheme is provided first to achieve bidirectional data transmission. To avoid deadlock and starvation, a priority-based design of ASM is introduced. This chapter mainly focused on contributions to the existing architecture and explained dynamic channel control protocol in detail.

In Chapter 3 we have discussed microarchitectural structures of FIFO in Multi-Synchronous Bi-Directional NoC's routers. This is placed at the input or the output interface of a router to accommodated incoming flits and outgoing flits which cannot be directly forwarded due to traffic situations. In GALS infrastructure multi-synchronous FIFO is used at the interface of the routers, which supports dynamic, extendable and power efficient multi-clock architectures. This projected architecture of buffer allowed the allocation of data amongst entirely separated clock domain modules with minimum cycles of latency between sender and receiver. The ready/valid handshake permits the sender and the receiver to prevent their operation for an arbitrary amount of time. An abstract FIFO may be tailored to the ready/valid protocol both inside the upstream and the downstream connections. In this chapter we developed a novel multi-synchronous buffer architecture that supports valid/ready flow control mechanism at all the different interfaces of the routers.

In Chapter 4 we have discussed different types of available arbiter architectures. The routers are the basic building blocks of interconnection networks and their design critically affects the performance of the whole system. The core function of any crossbar scheduler is arbitration that resolves conflicting requests for the output. Since the delay of the arbiters directly determine the operation speed of the scheduler, the design of faster arbiters is of paramount importance. The core of each NoC router involves arbiter and multiplier pairs that need to be carefully co-optimized in order to achieve an overall efficient implementation. Low transmission latency design is one of the most important parameters of NoC design.
In this chapter, we have used parametric Verilog HDL to implement designs and compared performance in terms of power, area, and delay of different types of arbiters used for NoC routers. The RTL implementation is performed using parametric Verilog HDL and analysis in term of power, area and delay is performed using Xilinx ISE 14.7 and Xpower Analyzer (XPA) with Xpower Estimator (XPE).

In Chapter 5 we have discussed and implemented routing computation modules using different popular routing algorithms. Routing algorithm is one of the most important design choices for NoC implementation, as it controls the path decision that a flit has to follow while traveling along the network.
In this chapter we have presented RTL Implementation and analysis of Synchronous Look Ahead Routing Computation using XY, Adaptive XY, and Dimension-Order algorithms dedicated to 2D-Mesh Topology for a Distributed Scalable Predictable Interconnect Network (DSPIN).
In this chapter we have summarized routing algorithms used for NoC by simply comparison and analysis,and give the merits and demerits of implemented routing algorithms.

In chapter 6 we have discussed and implemented a DMA-based Network Interface (NI). This implemented NI has three memory mapped registers, Read packet register, Write packet register, and Status packet register. This chapter has focused on implementation of proposed NI design and analyzed its performance in terms of power, area, and delay using RTL NI model. The RTL coding is performed using Verilog HDL and synthesized using Xilinx ISE

14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology.

In chapter 7 we have discussed and implemented Virtual Channel Allocators and Switch Allocators in three different ways. In this chapter we have implemented VC allocation using separable allocators as Separable input-first VC allocator, Separable output-first VC allocator, and Wavefront-based VC allocator. We have also implemented switch allocation using separable allocators as Separable input-first switch allocator, Separable output-first switch allocator, and Wavefront-based switch allocator. The RTL model is implemented using Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology. Their performance were evaluated in terms of power, area and delay.
In this thesis, a MBiNoC architecture using dynamic self-reconfigurable bidirectional channels is proposed and implemented. A new Dynamic Channel Control protocol that supports real-time traffic direction arbitration while avoiding deadlock and starvation has been presented. MBiNoC architecture can significantly reduce the packet delivery latency at all levels of packet injection rates. Compared to the conventional NoCs, the bandwidth utilization and traffic consumption rate of our MBiNoC also exhibited higher efficiency.

On-chip networks are communication infrastructure dedicated for on-chip multiprocessor systems. The performance of an on-chip interconnection networks affects the performance of the networked multiprocessor systems because communication overheads due to task-level parallelism affecting the total computational time of a complex parallel computing. Therefore, a high performance network-on-chip should be a general requirement to develop a networked multiprocessor system in the future.
In accordance with the current status of the research results presented in this thesis, further investigations to improve the flexibility, to increase performance, to reduce power dissipation and to optimize the logic area of the proposed NoC router microarchitecture will be still open as consider IP-router interface is completely asynchronous and router-router interface is synchronous.
As the bi-directional routers introduced in this thesis may increase the critical path length and power overhead, a more accurate hardware overhead evaluation of MBiNoC should also consider the physical VLSI implementation and optimization of the routers.

Publications

This section presents the publications performed during the elaboration of this thesis. The following is the list of publications organized from latest to oldest.

Journal

Title: Buffer Architecture Design for Multi-Synchronous Bi-Directional NoC's Routers.

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Journal: Indian Journal Science and Technology

Status: Accepted

Abstract: Microarchitectural structures of buffers in Multi-Synchronous Bi-Directional NoC's routers have a noteworthy impression on the overall performance of an on-chip network for Globally Asynchronous Locally Synchronous (GALS) infrastructures. This buffering can be at the input interface or the output interface of a router to accommodate incoming flits and outgoing flits which cannot be directly forwarded due to traffic situations. This projected architecture of buffer allows the allocation of data amongst entirely separated clock domain modules with minimum cycles of latency between sender and receiver. This paper developed a novel multi-synchronous buffer architecture that supports valid/ready flow control mechanism at all the different interfaces of the routers. This proposed buffer architecture is implemented using parametric Verilog HDL and synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology and its performance is evaluated in terms of power, area and delay. .

Quartile: 2 `http://www.scimagojr.com/journalsearch.php?q=21100201522&tip=sid&clean=0`

ISSN: 0974-5645

Journal

Title: A Multi-Synchronous Bi-Directional NoC (MBiNoC) architecture with dynamic self-reconfigurable channel for the GALS infrastructure.

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Journal: Alexandria Engineering Journal

Status: Published

Abstract: To enhance the performance of on-chip communications of Globally Asynchronous Locally Synchronous Systems (GALS), a dynamic reconfigurable multi-synchronous router architecture is proposed to increase network on chip (NoC) efficiency by changing the path of the communication link in the runtime traffic situation. In order to address GALS issues and bandwidth requirements, the proposed multi-synchronous bidirectional NoC's router is developed and it guarantees higher packet consumption rate, better bandwidth utilization with lower packet delivery latency. All the input/output ports of the proposed router behave as a bi-directional ports and communicate through a novel multi-synchronous first-in first-out (FIFO) buffer. The bidirectional port is controlled by a dynamic channel control module which provides a dynamic reconfigurable channel to the router itself and associated sub-modules. This proposed multi-synchronous bidirectional router architecture is synthesized using Xilinx ISE 14.7 and FPGA Virtex 6 family device XC6VLX760 is considered as target technology and its performance is evaluated in terms of power, area and delay.

Quartile: 1 http://www.scimagojr.com/journalsearch.php?q=13907&tip=sid

DOI: https://doi.org/10.1016/j.aej.2017.02.019

ISSN: 1110-0168

1. International Conference - RTEICT 2016

Title: Design and analysis of DMA based network interface for NoC's router.

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Publication: IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), May 20-21, 2016.

Abstract: Recently Multi-core Globally Asynchronous Locally Synchronous(GALS) architecture emerged as the de facto to solution to satisfy the performance requirement of mainly modern embedded applications. Due to scalability, flexibility, and high bandwidth properties, Network-on-chip (NoC) technique has been proposed as a promising solution for the communication-centric platform. The rapid introduction of NoC for the multi-core GALS architecture requires proper Network-Interface (NI) design to interface two different IP blocks via network routers within the same chip. This paper proposed a DMA based Network-Interface for connecting the NoC router to a processor element. This proposed NI has three memory mapped registers, Read packet register, Write packet register, and Status packet register. Considering the current trend towards multi-core GALS architectures and the critical role, the interconnect plays in the whole system, this paper focuses on proposed NI design with its performance analysis in term of power, area, and delay using RTL NI model. The RTL coding is performed using Verilog HDL and simulation with implementation is performed by the Xilinx 14.7 and FPGA Vertex 6 as the target technology.

URL: http://ieeexplore.ieee.org/document/7808019/

Print ISBN: 978-1-5090-0775-2

E-ISBN: 978-1-5090-0774-5

2. International Conference - ICCCA 2016

Title: RTL implementation and analysis of fixed priority, round robin, and matrix arbiters for the NoC's routers.

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Publication: IEEE International Conference on Computing, Communication and Automation (ICCCA), April 29-30, 2016.

Abstract: Networks-on-Chip (NoC) is an emerging on-chip interconnection centric platform that influences modern high speed communication infrastructure to improve the performance of many-core System-on-Chip (SoCs) designs. The core of each NoCs router involves arbiter and multiplier pairs that need to be carefully co-optimized in order to achieve an overall efficient implementation. Low transmission latency design is one of the most important parameters of NoC design. This paper uses parametric Verilog HDL to implement the designs and compares the performance in terms of power, area, and delay of different types of arbiters using for NoCs routers. The RTL implementation is performed using parametric Verilog HDL and analysis in term of power, area and delay is performed using Xilinx ISE 14.7 and Xpower Analyzer (XPA) with Xpower Estimator (XPE). The target device uses for these implementation is Vertex 6.

URL: `http://ieeexplore.ieee.org/document/7813949/`

Print ISBN: 978-1-5090-1667-9

E-ISBN: 978-1-5090-1666-2

3. International Conference - ICECS 2016

Title: Design and Analysis of Synchronous Look-Ahead Routing Computation using XY, Adaptive XY, and Dimension-Order algorithms for the 2D-Mesh Topology.

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Publication: 3rd IEEE International Conference on Electronics and Communication Systems (ICECS), February 25-26, 2016.

Abstract: Network-on-Chip (NoCs) provide a robust and scalable communication method for multi-processors system-on-chip (MP-SoCs) architecture with increasing number of cores and their performance is highly dependent on the throughput and latency properties of the micro-architecture of routers and its routing algorithms. Routing algorithm is one of the most important design choices for NoC implementation, as it controls the path decision that a flit has to follow while traveling along the network. In the case of look-ahead routing computation, each router pre-compute the preferred output ports based on its local blockage and transfer the preferred output ports to the adjacent routers. In this paper we represent RTL Implementation and analysis of Synchronous Look Ahead Routing Computation using XY, Adaptive XY, and Dimension-Order algorithms dedicated to 2D-Mesh Topology for a Distributed Scalable Predictable Interconnect Network (DSPIN). The RTL implementation is performed using parametric Verilog HDL and analysis in term of power, area and delay is performed using Xilinx ISE 14.7 and Xpower Analyzer (XPA) with Xpower Estimator

(XPE).

Note: Not yet publish on IEEE website.

4. International Conference - NGCT 2015

Title: Design of a dynamic depth high-throughput multi-clock FIFO for the DSPIN

Authors: Rajeev Kamal and Juan M. Moreno Arostegui

Publication: IEEE International Conference on Next Generation Computing Technologies (NGCT), September 4-5, 2015

Abstract: The clock distribution within Chip-Multiprocessors(CPMs) and System-on-chips (SoCs) come to be difficult as the number of processing elements increasing and the communication between those components are becoming even more critical. In recent years, researchers proposed Globally Synchronous Locally Synchronous (GALS) clocking scheme to reduce clock skew, power, and energy consumption in CPMs and SoCs. In this paper we have demonstrated dynamic depth multi-synchronous first-in first-out (FIFO) buffer which is useful for transferring data between two processing elements within a Distributed Scalable Predictable Interconnect Network(DSPIN).It also demonstrates dynamic calculation of FIFO depth using two clock frequency and packet size of in coming data.

URL: `http://ieeexplore.ieee.org/document/7375077/`

DVD ISBN: 978-1-4673-6807-0

E-ISBN: 978-1-4673-6809-4

USB ISBN: 978-1-4673-6808-7

Bibliography

[1] A. Jantsch and H. Tenhunen, "Networks-on-Chip," Norwell, MA: Kluwer, 2003.

[2] W. J. Dally and B. Towels, "Route Packets, Not Wires: On-Chip Interconnection Networks," in 38th ACM Design Automation Conference, pages 684–689, 2001.

[3] L. Benini and G. DeMicheli, "Networks on chips: A new SoC paradigm," in IEEE Computer, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[4] D. Geer, "Networks on Processors Improve On-Chip Communications," in IEEE Computer, vol. 42, no. 3, 17–20, March 2009.

[5] J. D. Owens, W. J. Dally, R. HO, D. N. Jayasimha, S. W. Keckler, and L. S. Peh, "Research Challenges for On-Chip Interconnection Networks," IEEE Micro, vol. 27, no. 5, pp. 96–108, Oct. 2007.

[6] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi, "Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC," CRC Press, Taylor & Francis Group, 2009.

[7] W. Dally and B. Towles, "Principles and Practices of Interconnection Networks," San Mateo, CA, USA: Morgan Kaufmann, 2003.

[8] N. E. Jerger and L. S. Peh, "On-Chip Networks," New York, NY, USA: Morgan Kaufmann, 2009.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et. al., "Large scale distributed deep networks," in Advances in Neural Information Processing System, pp. 1223-1231 2012.

[10] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *IEEE conference Acoustics*, Speech and Signal Processing (ICASSP), pp. 8595-8598, 2013 .

[11] A. Agarwal, C. Iskander, and R. Shankar, "Survey of Network on chip (NoC) Architecture & Contribution," in Journal of Engineering, Computing, and Architecture, vol. 3, no. 1, 2009.

[12] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Computer Surveys, vol. 38, no. 1, pp. 1–51, Mar. 2006.

[13] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," in proceedings of the IEEE, vol. 89, no. 4, pp. 490–504, April 2001.

[14] W. C. Tsai, Y. C. Lan, Y. H. Hu, and S. J. Chen, "Networks on Chips: Structure and Design Methodologies," Journal of Electrical and Computer Engineering Volume 2012 (2012), Article ID 509465, 15 pages.

[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 Proceeding 19th IEEE International Conference on Parallel and Distributed Processing Symposium, pp. 155-163, 2005.

[16] C. Albenes, ZFGME Santo, AA Susin, "ParlS: A parameterizable interconnect switch for Networks-on-Chips," in Proceeding ACM Conference, pp. 204-209, 2004.

[17] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J.V. Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," in IEE proceeding on Computers and Digital Techniques, vol. 150, no. 5, pp. 294-302, Sep. 2003.

[18] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: Concepts, architectures, and implementations," in IEEE Design & Test of Computers, vol. 22, no. 5, pp. 414-421, Sep. 2005.

[19] T. Bjerregaard and J. Sparsø, "Virtual channel designs for guaranteeing bandwidth in asynchronous Networkon- Chip," in Proceeding of IEEE Norchip Conference, pp. 269–272, Nov. 2004.

[20] T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless Network-on-Chip," in Proceeding of IEEE on Design Automation and Test, vol. 2, pp. 1226-1231, 2005.

[21] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," in Journal of Systems Architecture, vol. 50, no. 23, pp. 105-128, February 2004.

[22] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "Cost considerations in Network on Chip," inIntegration The VLSI Journal, no. 38, pp. 19-42, 2004, .

[23] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," in IEEE Circuits and Systems Magazine, vol. 4, no. 2, pp. 18-31, 2004.

[24] W.J. Dally, "Virtual-channel flow control," in IEEE Transactions on Parallel and Distributed Systems vol. 3, no. 3, pp. 194–205, 1992.

[25] D. Medhi and K. Ramasamy, "Network Routing: Algorithms, Protocols, and Architectures," Morgan Kaufmann Publishers, an imprint of Elsevier 2007.

[26] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in Proceeding 11th International Symposium on Asynchronous Circuits and Systems (ASYNC), pp. 54 – 63, 2005.

[27] I. Saastamoinen, M. Alho, and J. Nurmi, "Buffer implementation for Proteo network-on-chip," in IEEE International conference Proceeding on Circuits and Systems, vol. 2, pp. 113-116, May 2003

[28] E. Bolotin, A. Morgenshtein, I. Cidon, R. Ginosar, and A. Kolodny, "Automatic hardware-efficient SoC integration by QoS Network-on-Chip," in Proceeding 11th International IEEE Conference on Electronics, Circuits and Systems, pp. 479-482, 2004.

[29] H. Zimmer, S. Zink, T. Hollstein, and M. Glesner, "Buffer-architecture exploration for routers in a hierarchical network-on-chip," in Proceeding 19th IEEE International Symposium on Parallel and Distributed Processing, pp., 1-4, April 2005.

[30] A. T. Tran and B. M. Baas, "RoShaQ: High-performance on-chip router with shared queues," in IEEE 29th International Conference on Computer Design (ICCD), pp 232–238, 2011.

[31] International Technology Roadmap for Semiconductors: Executive Summary, Semiconductor Industry Association, 2007.

[32] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic Communication: A New Paradigm for Fault Tolerant Networks on Chip," VLSI Design, vol. 2007, Article ID 95348, pp. 1–17, 2007.

[33] 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," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 1, pp. 3–21, January 2009.

[34] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in Proceeding IEEE Proceeding on Computer Society, pp. 46-51, February 2004.

[35] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in Proceeding ACM Conference, pp. 224-229, 2005.

[36] H. Zimmer and A. Jantsch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in Proceeding First International IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis, pp. 188-193, 2003.

[37] D. Bertozzi, L. Benini, and G. D. Micheli, "Error control schemes for on-chip communication links: the energy reliability tradeoff," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 6, pp. 818-831, 2005.

[38] A. Radulescu, J. Dielissen, S. G. Pestana, and O. P. Gangwal, "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," in IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, vol. 24, no. 1, January 2005.

[39] P. Bhojwani and R. Mahapatra, "Interfacing cores with on-chip packet-switched networks," in Proceeding 16th International IEEE Conference on VLSI Design, pp. 382–387, 2003.

[40] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. V. Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks-on-Chip," in Proceedings of the Design Automation and Test in Europe Conference, pp. 350–355, March 2003.

[41] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, "RASoC: A router soft-core for networks-on-chip," in Proceeding Design Automation and Test in Europe Conference, vol. 3, pp. 198-203, 2004.

[42] N. Tushar, K. Jain, M. Ramakrishna, and P. V. Gratz, "Asynchronous Bypass Channels for Multi-Synchronous NoCs: A Router Microarchitecture, Topology, and Routing Algorithm," in IEEE Transactions on Computer-Aided Design Of Integrated Circuits And Systems, vol. 30, No. 11, November 2011.

[43] P. Golani, G. Dimou, M. Prakash, and P. Beerel, "Design of a High-Speed Asynchronous Turbo Decoder," in Proceeding IEEE 13th Intel Symp. Asynchronous Circuits and Systems, pp. 49-59, Mar. 2007.

[44] N. Onizawa, V. Gaudet, and T. Hanyu, "Low-Energy Asynchronous Interleaver for Clockless Fully Parallel LDPC Decoding," in IEEE Transaction Circuits and Systems, vol. 58, no. 8, pp. 1933-1943, Aug. 2011.

[45] A. Lines, "Asynchronous Interconnect for Synchronous SoC Design," IEEE Micro, vol. 24, no. 1, pp. 32-41, Jan./Feb. 2004.

[46] M. Horak, S. Nowick, M. Carlberg, and U. Vishkin, "A Low-Overhead Asynchronous Interconnection Network for Gals Chip Multiprocessors," in IEEE Transaction Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 4, pp. 494-507, Apr. 2011.

[47] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar, "An Asynchronous Router for Multiple Service Levels Networks on Chip," in Proceedign IEEE 11th Intel Symp. Asynchronous Circuits and Systems, pp. 44-53, Mar. 2005.

[48] J. Bainbridge and S. Furber, "CHAIN: A Delay-Insensitive Chip Area Interconnect," in IEEE Micro, vol. 22, no. 5, pp. 16-23, Sept./Oct. 2002.

[49] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, "A Telecom Baseband Circuit Based on an Asynchronous Network-on-Chip," in IEEE Intel Solid-State Circuits Conference Digest of Technical Papers, pp. 258-601, Feb. 2007.

[50] A. Alhussien, C. Wang, and N. Bagherzadeh, "A Scalable Delay Insensitive Asynchronous NoC with Adaptive Routing," in Proceeding IEEE 17th Intel Conf. Telecomm., pp. 995-1002, Apr. 2010.

[51] Y. Thonnart, P. Vivet, and F. Clermidy, "A Fully-Asynchronous Low-Power Framework for GALS NoC Integration," in Proceeding Design, Automation and Test in Europe Conference Exhibition, pp. 33- 38, Mar. 2010.

[52] W. Song and D. Edwards, "A Low Latency Wormhole Router for Asynchronous On-Chip Networks," in Proceeding 15th Asia and South Pacific Design Automation Conference, pp. 437-443, Jan. 2010.

[53] D. Gebhardt, J. You, and K.S. Stevens, "Link Pipelining Strategies for an Application-Specific Asynchronous NoC," in Proceeding IEEE/ACM Fifth Intel Symp. Networks on Chip, pp. 185-192, May. 2011.

[54] N. Onizawa, A. Matsumoto, T. Funazaki, and T. Hanyu, "High-Throughput Compact Delay-Insensitive Asynchronous NoC Router," in IEEE Transactions On Computers, Vol. 63, No. 3, March 2014.

[55] Sheibanyrad, A. Panades, and I. Greiner, "Systematic Comparison between the Asynchronous and the Multi-Synchronous Implementations of a Network on Chip Architecture," in Design, Automation & Test in Europe Conference & Exhibition, 2007.

[56] I. M. Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture," in Second ACM/IEEE International Symposium on Networks-on-Chip IEEE 2008.

[57] M. Millberg, E. Nilsson, R. Thid and A. Jantsch, "Guaranteed bandwidth using looped containersin temporally disjoint networks within the nostrum network-on-chip," in Proceedings of IEEE Design, Automation and Testing in Europe Conference, pp. 890–895, 2004.

[58] F. G. Moraes, N. L. V. Calazans, A. V. D. Mello, L. H. Möller, and L. C. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," in Integration, the VLSI Journal, vol. 38, no. 1, pp. 69–93, Oct. 2004.

[59] G. Reehal, M. A. Abd El Ghany, and M. Ismail, "Octagon architecture for low power and high performance NoC design," in IEEE National Aerospace and Electronics Conference (NAECON), July 2012.

[60] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda and L. T. Smit, "An energy-efficient reconfigurable circuit switched network-on-chip," in Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS) , 2005.

[61] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez and C. Zeferino, "SPIN: A scalable, packet switched, on-chip micro – network," in Design Automation and Test in Europe Conference and Exhibition (DATE), pp. 70–73, March 2003.

[62] M. A. A. Faruque, T. Ebi, and J. Henkel, "Configurable Links for Runtime Adaptive On-Chip Communication," in Proceedings of the Design Automation and Test in Europe Conference, pp. 256-261, April 2009.

[63] Y. C. Lan, H. A. Lin, S. H. Lo, Y. H. Hu, and S. J. Chen, "A Bidirectional NoC (BiNoC) Architecture with Dynamic Self-Reconfigurable Channel," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 3, Mar. 2011.

[64] Z. Qian, S. M. Abbas, and C. Y. Tsui, "FSNoC: A Flit-Level Speedup Scheme for Network-on-Chips Using Self-Reconfigurable Bidirectional Channels," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, Sep. 2015.

[65] J. Zhu, Z. Qian, C. Y. Tsui, "BiLink: A high performance NoC router architecture using bi-directional link with double data rate," in Integration, the VLSI journal, vol. 55, pp 30-42, Sep 2016.

[66] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas, "Oblivious Routing in On-Chip Bandwidth-Adaptive Networks," in Proceedings of the Parallel Architectures and Compilation Techniques, pp. 181-190, Sep. 2009.

[67] F. Karim , A. Nguyen and S. Dey, "An interconnect architecture for networking systems on chips," in IEEE Micro , Vol. 22, No. 5, pp. 36–45, Sep./Oct. 2002.

[68] Wen-Chung Tsai,Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen, "Networks on Chips: Structure and Design Methodologies," in Journal of Electrical and Computer Engineering Volume 2012 (2012), Article ID 509465, 15 pages.

[69] F. N. Najm, "Survey of power estimation techniques in VLSI circuits," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 4, pp. 446–455, 1994.

[70] S. Kumar, A. Jantsch, and J. P. Soininen, "Network-on-chip architecture and design methodology," in Proceedings of the International Symposium on Very Large Scale Integration, pp. 105–112, April 2000.

[71] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le ,et al., "Large scale distributed deep networks," in Advances in Neural Information Processing Systems pp. 1223-1231, 2012.

[72] Quoc V Le, "Building high-level features using large scale unsupervised learning," in IEEE International Conference on Acoustics, Speech and Signal, pp.8595–8598, 2013.

[73] D. Wentzlaff, P. Griffin, H. Hoffmann et al., "On-chip interconnection architecture of the tile processor," in IEEE Micro, vol. 27, no. 5, pp. 15–31, 2007.

[74] G. DeMicheli and L. Benini, "Networks on Chips: Technology and Tools," Morgan Kaufmann, Waltham, Mass, USA, 2006.

[75] A. Chattopadhyay and Z. Zilic, "GALDS: A complete framework for designing multi clock ASICs and SoCs," in IEEE Transaction VLSI System, vol. 13, no. 6, pp. 641–654, 2005.

[76] R. S. Ramanujam, V. Soteriou, B. Lin, and L. Peh, "Extending the Effective Throughput of NoCs With Distributed Shared-Buffer Routers," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 4, pp. 548–561, 2011.

[77] D. DiTomaso, R. Morris, A. K. Kodi, A Sarathy, and A. Louri, "Extending the Energy Efficiency and Performance With Channel Buffers, Crossbars, and Topology Analysis for Network-on-Chips," in IEEE Transactions on VLSI Systems, vol. 21, no. 11, pp. 2141 – 2154, 2013.

[78] V. Soteriou, R.S. Ramanujam, B. Lin, and L. Peh, "A High-Throughput Distributed Shared-Buffer NoC Router," in IEEE Computer Architecture Letters, vol. 8, no. 1, pp. 21 – 24, 2009.

[79] W. J. Dally and J. W. Poulton, "Digital Systems Engineering," in Cambridge University Press, 1998.

[80] J. Ebergen, "Squaring the FIFO in GasP," in Proceeding Asynchronous Circuits and Systems, pp. 194-205, 2001.

[81] C. E. Molnar, I. W. Jones, W .S. Coates, and J. K. Lexau, "A FIFO ring performance experiment," in Proceeding Advance Research in Asynchronous. Circuits and Systems, pp.194-205, 2001.

[82] M. R. Greenstreet, "Implementing a STARI chip," in Proceeding International conference on VLSI in Computers and Processors, pp. 38-43, 1995.

[83] A. Chakraborty and M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in Proceeding Asynchronous Circuits and Systems, pp. 78-88, 2003.

[84] J. N. Siezovic, "Pipeline synchronization," in Proceeding Advanced Research in Asynchronous Circuits and Systems, pp. 87-96, 1994.

[85] R. W. Apperson, Z. Yu, M. J. Meeuwsen, T. Mohsenin, and B. M. Baas, "A Scalable Dual-Clock FIFO for Data Transfers Between Arbitrary and Haltable Clock Domains," in IEEE Transaction on VLSI System, vol. 15, no. 10, pp. 1125-1134, 2007.

[86] I. M. Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in First International Symposium on NoC, pp. 83-94, 2007.

[87] T. Chelcea, S. M. Nowick, "A low-latency FIFO for mixed-clock systems," in Proceeding IEEE Computer Society Workshop on VLSI, pp. 119-126, 2000.

[88] C. Cummings, "Simulation and synthesis techniques for asynchronous FIFO design," in Synopsys Users Group 2002.

[89] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, "Microarchitecture of Network-on-Chip Routers, A Designer's Perspective," Springer 2015.

[90] P. P. Chu, "RTL Hardware Design Using VHDL, Coding for Efficiency, Portability, and Scalability," in John Wiley Sons 2006.

[91] R. Ginosar, "Metastability and Synchronizers: A Tutorial," in IEEE Design & Test of Computers, vol. 28, no. 5, pp. 23–35, 2011.

[92] R. Sharma, V. V. Joshi, and V. M. Rohokale, " Performance of router design for Network-on-Chip implementation," in International Conference on Communication, Information & Computing Technology (ICCICT), 2012.

[93] Yutian Huan and Andre DeHon, "FPGA optimized packet-switched NoC using split and merge primitives," in IEEE International Conference on Field-Programmable Technology (FPT), 2012.

[94] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz,D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," in IEEE Journal of Solid-State Circuits, vol. 43, pp. 6–20, Jan. 2008.

[95] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-chip Interconnection Networks of the TRIPS Chip," in IEEE Micro, pp. 41–50, Sep./Oct. 2007.

[96] L.S. Peh and W. J. Dally, "A Delay Model for Router Microarchitectures," in IEEE Micro, pp. 26–34, Jan./Feb. 2001.

[97] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in 31st Annual International Symposium on Computer Architecture (ISCA'04), 2004, pp. 188–198.

[98] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," in IEEE Transaction Parallel Distributive System, vol. 4, no. 1, pp. 13–27, 1993.

[99] C. Wissem, B. Attia, A. Noureddine, A. Zitouni, and R. Tourki, "A Quality of Service Network on Chip based on a New Priority Arbitration mech-anism," in 2011 International Conference on Microelectronics , pp.1-6 Dec. 2011.

[100] "Arbiter with dynamic priority scheme. DesignWare Building Block IP," in Synopsys 2009.

[101] F. E. Guibaly, "Design and analysis of arbitration protocols," in IEEE Transactions on Computers, vol. 38, no. 2, Feb. 1989.

[102] P. Gupta and N. McKeown, "Designing and Implementing a Fast Crossbar Scheduler," in IEEE Computer Society, vol. 19, no. 1 pp.20-28, Feb 1999.

[103] G. Dimitrakopoulos, N. Chrysos, and C. Galanopoulos, "Fast arbiters for on-chip network switches," in IEEE International Conference on Computer Design, pp.664-670, Oct. 2008

[104] J. J. Lecler and G. Baillieu, "Application driven network-on-chip architecture exploration & refinement for a complex SoC," in Design Automation for Embedded Systems vol. 15, no. 2, pp. 133–158, 2011.

[105] "A comparison of network-on-chip busses," in white paper Arteris, 2005. 105

[106] W. J. Dally, C. Malachowsky, and S. W. Keckler, "21st century digital design tools," in Proceedings of the 50th Annual Design Automation Conference (DAC 2013), pp 1–96, 2013.

[107] J. Duato, S. Yalamanchili, and L. Ni, "Interconnection networks - an engineering approach," Morgan Kaufmann Publishers 1997.

[108] J. Flich and J. Duato, "Logic-Based Distributed Routing for NoCs," in Computer Architecture Letters, vol. 7, no.1, pp.13-16, Jan. 2008.

[109] M. Galles "Spider: A high-speed network interconnect," in IEEE Micro, vol. 17, no.1, pp.34-39, Jan. 1997.

[110] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "LAPSES: a recipe for high performance adaptive router design," in Proceeding of Fifth International Symposium On High-Performance Computer Architecture, pp.236-243, Jan. 1999.

[111] A. Siebenborn, O. Bringmann, and W. Rosenstiel, "Communication analysis for network-on-chip design," in Proceeding IEEE International conference on Parallel Computing in Electrical Engineering, pp. 315-320, 2004.

[112] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in Proceeding IEEE Conference Design Automation and Test in Europe, vol. 1, pp. 234- 239, 2004.

[113] C. Neeb, M. Thul, and N. Andwehn, "Network on-chip-centric approach to interleaving in high throughput channel decoders," in Proceeding IEEE International Symposium on Circuits and Systems, pp. 1766–1769, 2005.

[114] F. Moraes and N. Calazan, "An infrastructure for low area overhead packet-switching network on chip," Integration - The VLSI Journal, vol. 38, Issue 1, pp. 69-93, October 2004.

[115] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in Proceeding 42nd Design Automation Conference,IEEE, pp.559-564, June 2005.

[116] N. Najib, A. Monemi, and M. N. Marsono, "Partially adaptive look-ahead routing for low latency Network-on-Chip," in IEEE Student Conference on Research and Development (SCOReD), pp.1-5, Dec. 2014.

[117] R. Manevich, I. Cidon, A. Kolodny, and I. Walter, "Centralized adaptive routing for NoCs," in Computer Architecture Letters, vol. 9, no.2, pp.57-60, Feb. 2010.

[118] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing," in 38th Annual International Symposium on Computer Architecture (ISCA), pp. 259-270, June 2011.

[119] S. Hu, W. Xu, J. Lin, and X. Lin, "Probabilistic odd-even: an adaptive wormhole routing algorithm for 2D-mesh network-on-chip," in The Journal of Supercomputing, pp. 123, Oct. 2014.

[120] M.Sheng, W. Zhiying, N. Enright Jerger, L. Shen, and N. Xiao, "Novel Flow Control for Fully Adaptive Routing in Cache-Coherent NoCs," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no.9, pp.2397-2407, Sept. 2014.

[121] Ruizhe Wu, Yi Wang and Dan Zhao, "A Low Cost Deadlock-free Design of Minimal-Table Rerouted XY-Routing for Irregular Wireless NOCs," in Fourth ACM/IEEE International Symposium on Networks-on-Chip, pp. 199 – 206 2010.

[122] M. Nickray, M. Dehyadgari, and A. Afzali-kusha, "Adaptive Routing Using Context-Aware Agents for Networks on Chips," in 4th International Design and Test Workshop (IDT), pp. 1-6, 2009.

[123] A. Rai, and P. Mathur, "Routing Algorithms and Performance Evaluation in Network on Chips," in International Journal of Innovative Research in Science & Engineering, Nov. 2016.

[124] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing", in Journal of the Association for Computing Machinery, vol. 41, no. 5, pp. 874-902, Sep. 1994.

[125] Ge-Ming Chiu, "The Odd-Even Turn Model for Adaptive Routing," in IEEE transactions on parallel and distributed systems, vol. 11, no.7, july 2000.

[126] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip," in IEEE Transactions on Computers vol. 57, no. 6, pp. 809–820, 2008.

[127] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS Infrastructure for a Massively Parallel Multiprocessor," in Design & Test of Computers,IEEE, vol. 24, no. 5, pp. 454-63, Sept. 2007.

[128] Ocp-ip, open core protocol specification, version 2.0. www.ocpip.org, 2003.

[129] Virtual component interface standard. http://www.vsi.org.

[130] Arm, amba 3.0 axi specification. 2004.

[131] G. Buzzard, D. Jacobson, S. Marovich, and J. Wilkes, "Hamlyn: A high performance network interface with sender-based memory management," in Proceedings of Hot Interconnects, 1995.

[132] D. J. Culler, J. P. Singh, and A. Gupta, "Parallel Computer Architecture: A Hardware/Software Approach," Morgan Kaufmann, San Francisco, CA, 1999.

[133] P. Steenkiste, "A high-speed network interface for distributed-memory systems: architecture and applications," ACM Transaction on Computer System, vol. 15, no. 1, pp. 75-109, 1997.

[134] A. A. Chien, M. D. Hill, and S. S. Mukherjee, "Design challenges for high-performance network interfaces," in Computer, vol. 31 no. 11, pp. 42-44, Nov 1998.

[135] S. C. Goldstein and T. Callahan, "Nifdy: A low overhead, high throughput network interface," in International Symposium on Computer Architecture, 1995.

[136] F. Steenhof, H. Duque, B. Nilsson, K. Goossens, and R.P. Llopis, "Networks on chips for high end consumer electronics tv system architectures," in Design, Automation and Test in Europe, vol. 2, pp. 1-6 March 2006.

[137] F. Angiolini, P. Meloni, S. Carta, L. Benini, and L. Raffo, "Contrasting a noc and a traditional interconnect fabric with layout awareness," in Proceedings of the Conference on Design, Automation and Test in Europe, pp. 124-129, Dec. 2006.

[138] S. P. Singh, S. Bhoj, D. Balasubramanian, T. Nagda, D. Bhatia, and P. Balsara, "Generic Network Interfaces for Plug and Play NoC Based Architecture," in Lecture Notes in Computer Science Springer :Reconfigurable Computing, pp. 287-298, Mar. 2006.

[139] M. Ebrahimi, M. Daneshtalab, N.P.Sreejesh, P.Liljeberg, and H.Tenhunen, "Efficient network interface architecture for network-on-chips," in Proceeding NORCHIP, Trondheim, pp. 1-4, Nov. 2009.

[140] W. Chouchene, B. Attia, A. Zitouni, N. Abid, and R. Tourki, "A low power Network Interface for network on chip," in A low power Network Interface for network on chip, pp. 1-6, Oct. 2011.

[141] S. E. Lee, J. H. Bahn, Y. S. Yang, and N. Bagherzadeh, "A Generic Network Interface Architecture for a Networked Processor Array (NePA)," in Architecture of Computing Systems (ARCS), pp. 247-260, Apr. 2008.

[142] P. Wielage and K. Goossens, "Networks on silicon: blessing or night-mare?," in Proceeding Euromicro Symposium on Digital System Design, pp. 196-200, 2002.

[143] A. Ferrante, S. Medardoni, and D. Bertozzi, "Network interface sharing techniques for area optimized noc architectures," in 11th Euromicro Conference on Design Architectures, Methods and Tools, pp. 10-17, Sep. 2007.

[144] H. M. Soliman, E. M. Saad, M. El-Bably, M. Hesham, and A. M. Keshk, "Designing a WISHBONE protocol network adapter for an asynchronous network-on-chip," in CoRR, abs 1203.4150, 2012.

[145] W. Chouchene, B. Attia, A. Zitouni, N. Abid, and R. Tourki, "A low power network interface for network on chip," in 8th International Conference on Systems, Signals and Devices (SSD), pp. 1-6, Mar. 2011.

[146] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," in IEEE Transaction on Parallel Distributive System, vol. 4, no. 4, pp. 466–475, 1993.

[147] Y. M. Boura and C. R. Das, "Performance analysis of buffering schemes in wormhole routers," in IEEE Transactions on Computers, vol. 46, pp. 687–694, 1997.

[148] T. Nachiondo, J. Flich, and J. Duato, "Destination-based hol blocking elimination," in International Conference on Parallel and Distributed Systems, IEEE Computer Society, pp 213–222, 2006.

[149] Li-Shiuan Peh, "Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks," PhD thesis, Stanford University, August 2001.

[150] Li-Shiuan Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," inn Proceedings of the Seventh International Symposium on High-Performance Computer Architecture, pp. 255–266, 2001.

[151] I. E. Sutherland, B. Sproull, and D. Harris, "Logical Effort: Designing Fast CMOS Circuits," Morgan Kaufmann Publishers Inc., San Francisco, CA, 1999.

[152] A. Kumar, P. Kundu, A. Singh, Li. S. Peh, and N. K. Jha, "A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS," in Proceedings of the 25th International Conference on Computer Design, 2007.

[153] M. Zhang and C. S. Choy, "Low-Cost Allocator Implementations for Networks-on-Chip Routers," VLSI Design, 2009.

[154] S. S. Mukherjee, F. Silla, P. Bannon, J. S Emer, S. Lang, and D. Webb, "A Comparative Study of Arbitration Algorithms for the Alpha 21364 Pipelined Router," in Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 223–234, 2002.

[155] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das, "A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks," in Proceedings of the 33rd International Symposium on Computer Architecture, pp. 4–15, 2006.

[156] J. Flich and J. Duato, "Designing Network On-Chip Architectures in the Nanoscale Era," CRC Press, Taylor & Francis, 2011.

[157] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–12, 2009.

[158] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High speed switch scheduling for local area networks," in ACM Transactions on Computer Systems, vol. 11, no. 4, pp. 319–352, 1993.

[159] N. Chrysos and G. Dimitrakopoulos, "Practical high-throughput crossbar scheduling," in IEEE Micro, vol. 29, pp. 22–35, 2009.