EXPLORING THE SIMULATION OF DYNAMIC PARTIAL RECONFIGURATION FOR NETWORK ON CHIP (NOC)-BASED FPGA

By

Amr Hassan Ali Baddar

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

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

EXPLORING THE SIMULATION OF DYNAMIC PARTIAL RECONFIGURATION FOR NETWORK ON CHIP (NOC)-BASED FPGA

By

Amr Hassan Ali Baddar

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

Under the Supervision of

Prof. Hossam A. H. Fahmy

Professor

Electronics and Communications
Engineering Department.
Faculty of Engineering, Cairo University

Dr. Hassan Mostafa

Assistant Professor

Electronics and Communications
Engineering Department.
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

EXPLORING THE SIMULATION OF DYNAMIC PARTIAL RECONFIGURATION FOR NETWORK ON CHIP (NOC)-BASED FPGA

By

Amr Hassan Ali Baddar

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

Approved by the Examining Committee

_____

Prof. Hossam A. H. Fahmy **(Thesis Main Advisor)**
Computer Professor, Faculty of Engineering, Cairo University

_____

Prof. Ahmed Hussein Khalil **(Internal Examiner)**
Electronics Professor, Faculty of Engineering, Cairo University

_____

Prof. Mohamed Abdelghany Salem **(External Examiner)**
Electronics Professor, Faculty of Engineering, German University Cairo

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2019

Engineer's Name:	**Amr Hassan Ali Baddar**
Date of Birth:	24/11/1989
Nationality:	Egyptian
E-mail:	amr_hassan@ymail.com
Phone:	+201061488997
Address:	10b St., El-Maadi, Cairo, Egypt
Registration Date:	01/10/2012
Awarding Date:	/ /2019
Degree:	**Master of Science**
Department:	**Electronics and Communication Engineering**

Supervisors:

Prof. Hossam A. H. Fahmy
Dr. Hassan Mostafa Hassan

Examiners:

Prof. Hossam A. H. Fahmy
(Thesis main advisor)
Computer Professor, Faculty of Engineering, Cairo University

Prof. Ahmed Hussein Khalil
(Internal Examiner)
Electronics Professor, Faculty of Engineering, Cairo University

Prof. Mohamed Abdelghany Salem
(External Examiner)
Electronics Professor, Faculty of Engineering, German University Cairo

Title of Thesis:

Exploring The Simulation of Dynamic Partial Reconfiguration for Network on Chip-Based FPGA

Key Words:
Dynamic Partial Reconfiguration, Network-on-Chip, Fields Programmable Gate Array

Summary:
In this thesis, a literature survey of exiting Dynamic Partial Reconfiguration (DPR) techniques for conventional FPGAs is presented. Then, a comparative review of these techniques is provided with respect to reconfiguration time and area. Following that, different network parameters at the NoC-based FPGAs have been analyzed to estimate the impact on DPR performance using a state-of-art simulator "NoC-DPR", Finally, a case study is introduced to clarify the DPR performance gap between NoC-based FPGAs and conventional FPGAs.

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Amr Hassan Ali Date:

Signature:

Acknowledgements

I would like to express my utmost gratitude to Allah for giving me the strength to complete the work.

I would also like to sincerely thank my supervisors, Prof. Hossam Fahmy and Dr Hassan Mostafa, for their continuous support and guidance throughout my work.

Table of Contents

List of Tables

List of Figures

Nomenclature

ASIC	Application-Specific Integrated Circuits
BIST	Built-In Self-Test
CAD	Computer-Aided Design
CLB	Configuration Logic Block
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DPR	Dynamic Partial Reconfiguration
FPGA	Field-Programmable Gate Array
ICAP	Internal Configuration Access Port
IP	Intellectual Property
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LUT	Look-Up Table
NI	Network Interface
NoC	Network on Chip
NRE	Non Recurring Engineering
P2P	Point to Point
PAR	Place and Route
PCIe	Peripheral Component Interconnect Express
PE	Process Element
RTL	Register Transfer Level
RU	Resource Usage
SDR	Software Define Radio
SEU	Single Event Upset
SoC	System on Chip
SRAM	Static Random Access Memory
TDM	Time Division Multiplexing
VC	Virtual Channel
WSF	West Side First

Abstract

System-on-Chip (SoC) designs are among the most widely used designs to implement computationally data-intensive applications, which are consist of several Processing Elements (PEs) and storage elements (SEs) communicating together through the aid of a network architecture. However, several Intellectual Properties (IPs) are implemented on a single chip due to the recent developments in the fabrication process of CMOS devices.

Meanwhile, Field Programmable Gate Arrays (FPGAs) are attracting more interest to exploit SoC designs due to the constraints of Application-Specific Integrated Circuits (ASICs) such as high Non-Recurring Engineering (NRE) cost and time to market. Recently, the number of PEs is increasing to maximise functionalities and capabilities of modern SoC designs. Accordingly, communication among those PEs becomes an essential factor in the design of large-scale systems. Consequently, numerous challenges of the communication amongst those PEs, while configured on FPGAs, are raised and thus, innovative solutions are needed. Therefore, Network-on-Chip (NoC) has been adopted for FPGA to address these communication challenges.

On the other hand, Dynamic Partial Reconfiguration (DPR) of SRAM- based FPGAs becomes a remarkable feature for many applications, as DPR represents the potential to add more flexibility during runtime. Moreover, adding the DPR feature to many applications is easier than before because of the recent developments FPGA's software designing tools such as Quartus for Intel (previously Altera) and ISE for Xilinx. On the contrary, DPR configuration technique such as Internal Configuration Access Port (ICAP) and Joint Test Action Group (JTAG) port, encounters a performance limitation because only one reconfiguration is permitted at a time.

In this thesis, a state-of-art NoC-based FPGA simulator is proposed, which supports DPR simulation. NoC-DPR simulator is used to estimate design limitations and performance degradations of using DPR for NoC-based FPGA. NoC-DPR simulator measures the reconfiguration time overhead, which caused by network's latency, and the concurrent reconfigurations on FPGA fabric. It is proven that the overhead of reconfiguration time increases exponentially with increasing the number of concurrent reconfigurations. However, further investigations show that the network of wormhole routers with virtual channels optimises the reconfiguration time with a factor up to 4x compared to the network of wormhole routers without virtual channels. Finally, a case study is introduced to clarify the DPR performance gap between NoC-based FPGAs and conventional FPGAs.

Chapter 1 : Introduction

This Chapter presents a short introduction to the importance of studying the impact of using the NoC approach on the DPR performance of FPGAs. Section 1.1 shows the motivation behind this research. In Section 1.2, the main thesis contributions are discussed. Section 1.3 provides the thesis outline and organization.

1.1. Motivation

The recent developments in a wide range of applications such as image and video signal processing, Software Defined Radio (SDR), and electronic measurement applications have led to adding more flexibility over runtime using software parameters or hardware reconfiguration, which increase the lifetime of the application. Therefore, reconfigurable devices such as FPGA offer a suitable approach over the traditional ASIC approach for these applications.

In addition to the significant advancements in FPGA's essential resources, most of the manufacturers are adding hard and soft blocks to the new FPGA fabric besides the essential logic elements, such as memories, processors, ALUs and Digital Signal Processing (DSP) blocks. Hence, FPGA has been commonly used over ASIC due to the enhancements in the performance of FPGA compare to ASIC, the fast time to market, the facile upgrading, and the low implementation cost.

Additionally, DPR is increasingly used for a variety of applications mapped on SRAM-based FPGAs technology to enforce the flexibility over runtime phase [1]. Furthermore, Partially Reconfigurable (PR) devices save chip area by programming only the necessary physical resources (such as LUTs, IP, and embedded blocks) in each corresponding execution phase. Accordingly, power is saved by programming just the needed blocks, which allows for static leakage reduction.

On the other hand, the system complexity increases because of the current developments in chip scaling process. Additionally, designs' layouts are developed from plain circuit layout into complex heterogeneous structures. Accordingly, the number of processing elements and the processing power have elevated; hence, data-intensive applications have materialized.

Thus, the challenge of the communication between those cores, when configured on FPGAs, has been triggered. Accordingly, Network-on-Chip (NoC) is considered as an effective alternative to traditional bus-based system-on-chip for FPGAs, which complicates the design process and reduces systems scalability and flexibility, to handle these communication challenges. FPGAs adopt freely the NoC solution [2], where FPGA's Configuration Logic Blocks (CLBs) are considered as computing cores (or PEs). CLBs are the basic logic blocks used to build any logic function. These cores use the hardwired network of routers (NoC) to communicate with each other. Consequently, those PEs can be dynamically reconfigured to adopt a new logic at the run-time without any change neither in network structure nor the other PEs.

Regardless of this increasing interest in both concepts, NoC and DPR, several researchers have focused on each concept individually rather than merging them.

Therefore, a tool called NoC-DPR, a cycle-accurate simulator for NoCs that supports DPR, is developed to investigate the impact of different network's parameters on DPR performance. The DPR performance is measured by the reconfiguration time. The

principal character of NoC-DPR simulator is foreseeing the dynamic behaviour of the NoC-based system before synthesis for the target architecture by estimating the suitable network size, area of each processing element, number of simultaneous reconfiguration process, and the target reconfiguration time.

1.2. Contribution

This work includes the following contributions:

• A comparative review of DPR techniques for the conventional FPGA such as ICAP and JTAG of Xilinx FPGA.

• Comparing several open-source NoC simulators to analyze the behaviour of the NoC by varying network parameters. Choosing the best match with an open-source library called ReChannel that simulates dynamic reconfiguration.

• Integrating NoCTweak with ReChannel library and evaluating the modification of network interface on the simulator and network performance.

• Using a state-of-art NoC-based FPGA simulator, which supports DPR simulation to measure the impact of varying network parameters such as network size, injection rate, buffer size, and different router types on DPR performance. In addition to that, NoC-DPR estimates the effect of simultaneous reconfiguration on the network's performance and reconfiguration time.

• Introducing two case studies, synthetic and embedded application, to compare between the conventional SRAM-based FPGA and the next-generation NoC-based FPGA performance of DPR.

1.3. Organization of the Thesis

To have an overview of the impact of NoC-based FPGAs on DPR performance at next-generation FPGAs, studying DPR at conventional SRAM-based FPGA plays an important role to have a clear overview. Hence, an introduction to SRAM-based FPGA and different configuration techniques, are illustrated in Chapter 2 along with NoC's background, structures, and the available NoC simulator. Chapter 3 provides a detailed survey and comparison between the different techniques of DPR on Xilinx FPGA. Chapter 4 presents the modifications in the NoCTweak simulator to support NoC-based FPGA simulation, which supports DPR simulation, along with a comparison of the NoC-DPR performance to NoCTweak simulator performance.

Since NoC design parameters such as (router type, virtual channel numbers, etc…) are the prominent factors that have a direct impact on the reconfiguration time, Chapter 5 presents the work done to study the reconfiguration time overhead. That is resulted in increasing simultaneous configurations on FPGA fabric using synthetic and embedded applications. The conclusion and the potential future work are discussed in the last Section.

Finally, Appendix A shows all options of NoC-DPR simulator, while Appendix B gives a detailed description of some created modules used for integration DPR simulation library.

Chapter 2 : Background

This chapter is organized as follows: Section 2.1 discusses the architecture of FPGA regarding logic and routing resources. Section 2.2 discusses the DPR definition, benefits, and techniques. NoC concepts are outlined in Section 2.3, while the simulation background is introduced in Section 2.4 and 2.5. Finally, the related work is discussed in Section 2.6.

2.1. FPGA Technologies

First, the logic block cells and the routing blocks of FPGAs are discussed, then a detailed architecture of DPR is presented, because DPR should not be addressed without having a background of the underlying FPGA architecture.

2.1.1. FPGA versus ASIC

FPGA is composed of a set of CLBs linked through a configurable interconnection network. Starting from the year 2000, FPGA features have scaled down from 130 nm to 20 nm as illustrated in Figure 2-1 for Virtex and Stratix FPGA family of Xilinx and Intel (Formerly Altera) Companies respectively, the most popular manufactures of FPGAs. Consequently, the logic density, the Lookup Tables (LUTs) of FPGA chip, has elevated by approximately 2000% over these years [3]. Moreover, the design of ASIC's layout becomes an extra complex challenge, and it takes a longer time to market and high Non-Recurring Engineering (NRE) cost; thus, the designers are motivated to use FPGAs for SoC implementations.

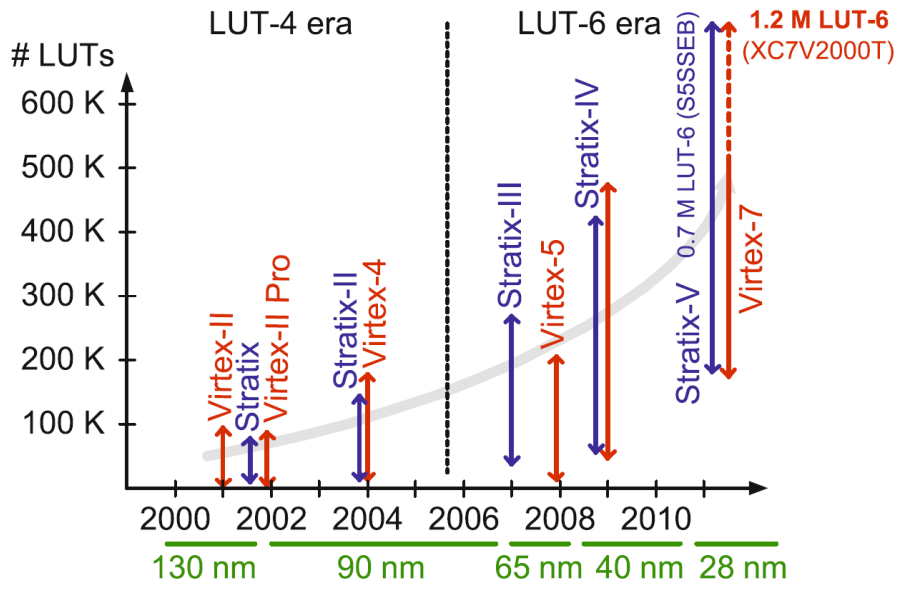


Figure 2-1: Device capacity regarding look-up tables of the Altera Stratix and Xilinx Virtex series FPGAs over the last decade [3]

2.1.2. FPGA architecture

The CLB is the main basic block of FPGA, which is responsible for implementing any logic function by using an N-input LUT. When the LUTs are programmed with a configuration data from the configuration plane as depicted in Figure 2-2, LUTs may represent any logic function up to N-inputs. This ability plays an essential role in building any desired logic function. Additionally, the data routing from Input-Output Block (IOB) to the logic block, through Switch Box, is used to build more complex blocks of the target function.

In the most recent FPGA architectures [5-15] the routing switches are distributed as a sea of routing resources, while the islands are the CLBs as illustrated in Figure 2-2(a). Accordingly, these boxes permit the communication easily along the rows and columns of CLBs. The CLBs are surrounding by the connection blocks, which attach the input and output signals to the routing switches. The switch boxes connect between the horizontal and vertical logic blocks. In this way, any arbitrary interconnections can be provided between the CLBs.

Note also the ports of configuration are shown in Figure 2-2, such as Joint Test Action Group (JTAG), SelectMap (SMAP), and Internal Configuration Access Port (ICAP). They are discussed in details in Chapter 3.

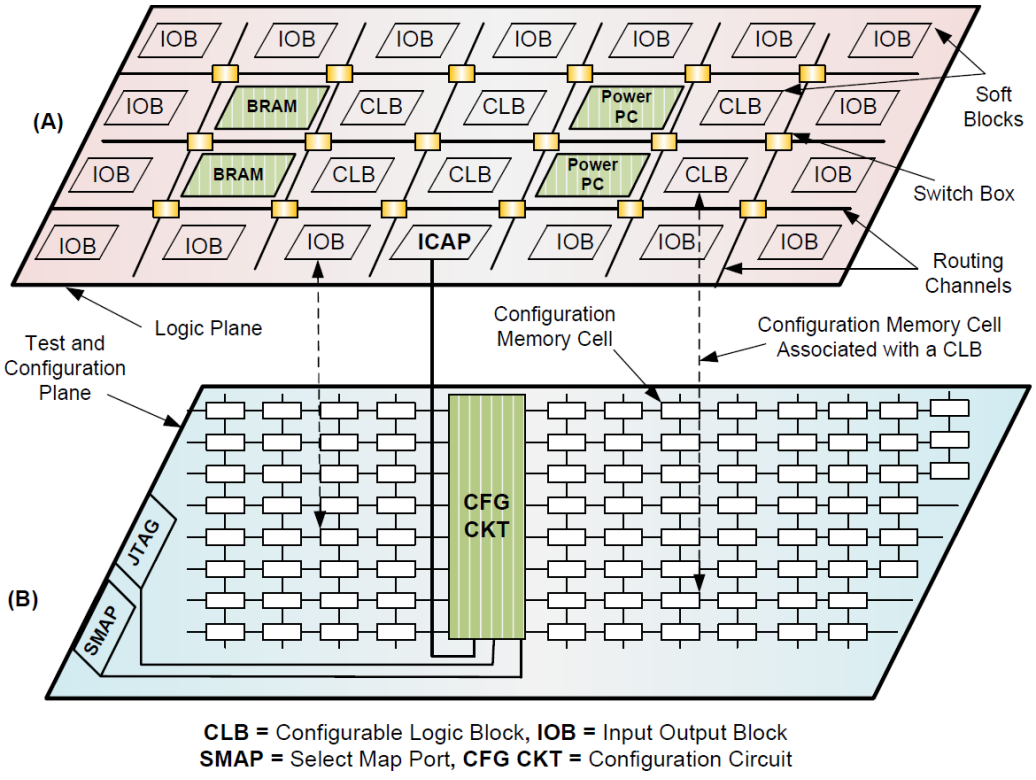


Figure 2-2: The Internal architecture of FPGA with two planes: (a) Logic plane, and (b) Test and Configuration plane [3]

2.1.3. Heterogeneous Arrays

As the technology node of Integrated Circuit (IC) layout downscales, more computing cores are implemented on the same dice area. Hence, the manufactures of FPGA provide a heterogeneous structure to improve the performance and versatility in computation. Consequently, the building blocks of FPGAs are not a set of switches boxes and CLBs anymore. The modern FPGA architectures also conduct hardwired blocks. Figure 2-3 highlights the evolution for Xilinx FPGA family, Virtex, over the last two decades.

The hard IP blocks such as RAMs, DSP, Digital Clock Managers (DCMs), Microprocessors, transceivers, and PCI Interfaces have been added to the Virtex families [6, 8, and 15]. In the conventional FPGA architectures, application IPs are implemented using the hardwired IPs or built using LUTs, are called soft implementation. However, the inter-IP communication architecture (i.e., NoC), which handle the communication between modules, is soft IP.

The next generation of FPGAs are discussed in this thesis, which are NoC-based FPGA platform. The new FPGAs have hardwired intercommunication modules of NoC.

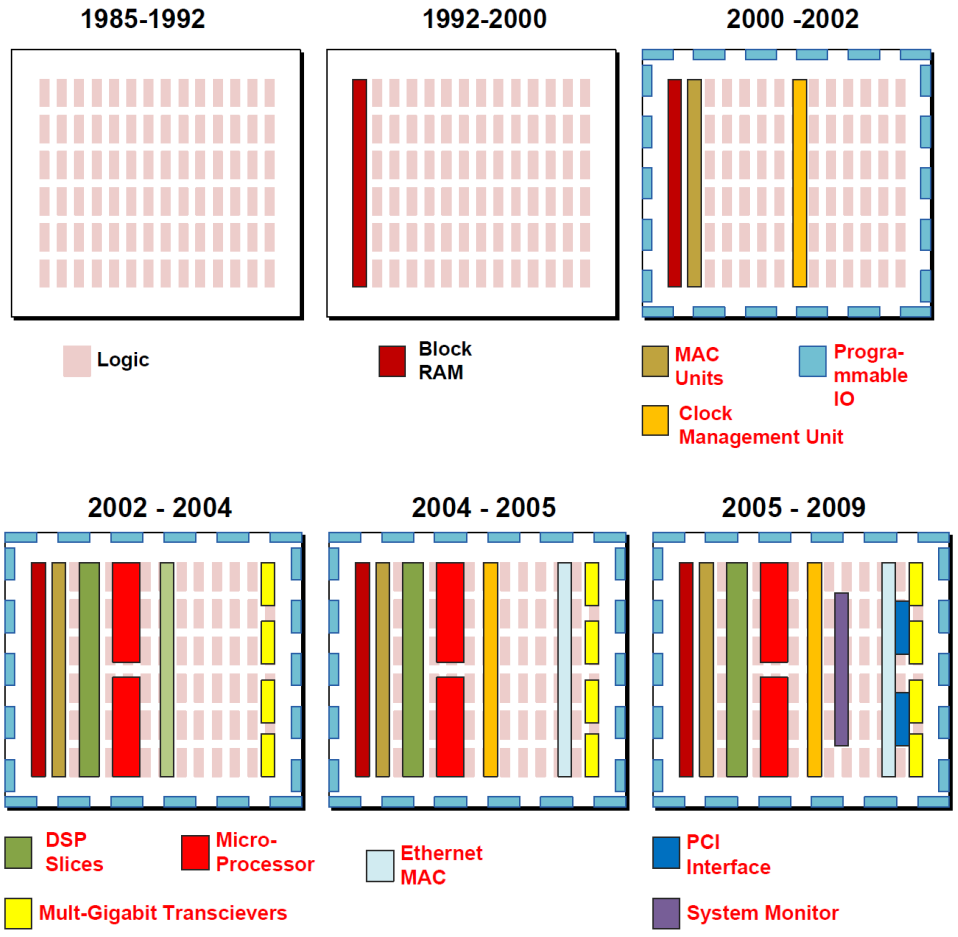


Figure 2-3: FPGA Virtex Family Architectural Evolution over the Years [5-15].

2.2. Dynamic Partial Reconfiguration.

PR is an act of reconfiguring a section of an FPGA after its preliminary configuration. Partial reconfiguration is classified into two types: DPR and Static Partial Reconfiguration. Static partial reconfiguration allows reconfiguration while the device is non-active. In DPR, the device is partially reconfigured while the rest of FPGA is still operating.

2.2.1. Reconfigurable Models

The configuration schemes that are used with reconfigurable systems as illustrated in Figure 2-4, are divided into (a) Single Context, (b) Multi-context and (c) Partially Reconfigurable [17]. The preliminary FPGA structures was a single context, where only a full configuration is allowed. However, this style was found limiting the implementation for run-time reconfiguration. In the recent years, Multi-Context Reconfiguration and DPR have been introduced to add more flexibility to FPGA designs.

2.2.1.1. Single Context

In the early FPGAs, all CLBs were programmed using a single bitstream of configuration information each time. That is because only entire FPGA access is allowed. Therefore, any required change to a certain CLB requires a complete reprogramming of the whole chip as illustrated in Figure 2-4(a). Moreover, the single context configuration simplifies the external reconfiguration process; it introduces a high configuration time overhead. Therefore, this mode is more acceptable for applications that do not require run-time reconfiguration.

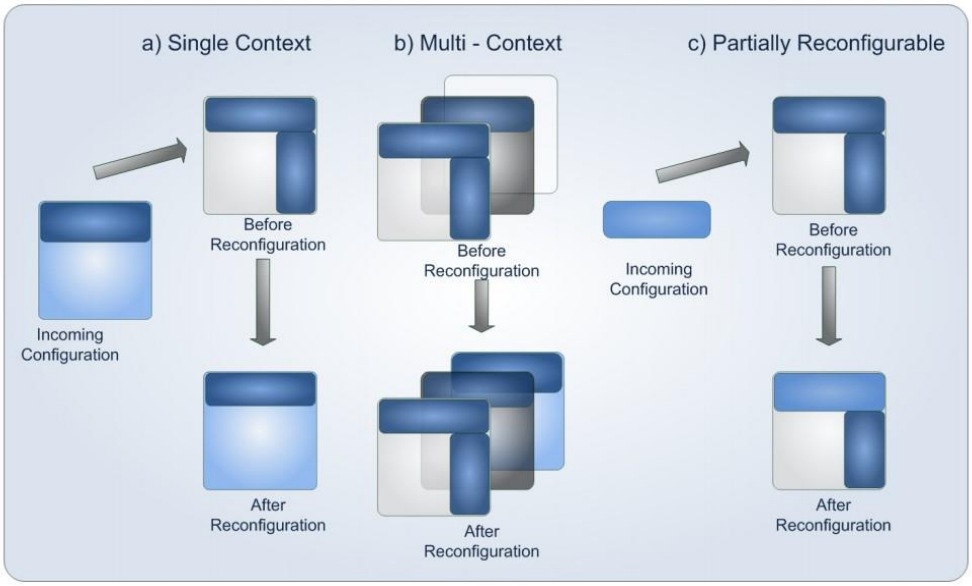


Figure 2-4: Different reconfiguration models: (a) Single Context, (b) Multi-Context, and (c) Partially Reconfigurable [17]

2.2.1.2. Multi-Context

A Multi-context FPGA consists of various configuration memory bits per programming bit place. Those memory bits are considered as Multi-Context of configuration, as shown in Figure 2-4(b). An example of multi-context FPGA is Tabula [4], which is a time-multiplexed FPGA. Hence, one plane of configuration is activated at a time. However, the FPGA switches between several configuration planes or contexts.

In this way, the multi-context FPGA is regarded as a multiplexed planes of a single device, which needs a full reconfiguration to perform any modification as detailed in Figure 2-5. Therefore, instead of having a single 16-adder as shown in Figure 2-5(a), it is divided into four 4-bit adders as shown in Figure 2-5(b) and cycle over the four configurations using time multiplexed configuration SRAM cell. Furthermore, this category of FPGAs has generally required a complicated storage technique, a multiplexed configuration SRAM cell, as shown in Figure 2-5(d). Moreover, time-multiplexed FPGA is considered non-suitable for many applications due to performance degradation as a result of reconfiguration time overhead.

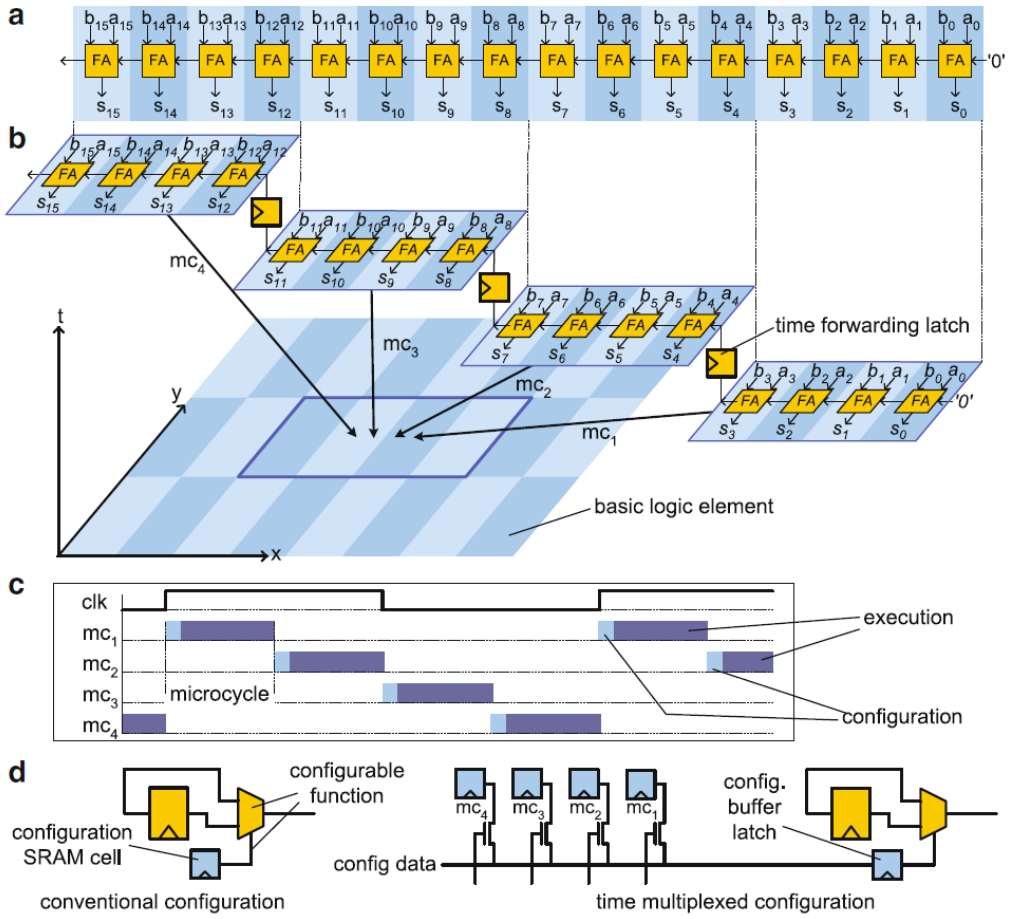


Figure 2-5: Time multiplexed FPGA; (a) 16-bit adder on a conventional FPGA; (b) time multiplexed implementation using four micro-reconfigurations (mc1,..,mc4); (c) timing; (d) configuration storage [18]

2.2.1.3. Partially Reconfigurable

PR of the FPGA is required if a section of the configuration needs alteration. In the modern FPGA, the programming plane is similar to the RAM device. The target section of the configuration data is accessed by addresses, which permits the reconfiguration of only the required part of the FPGA.

Due to the fact that any part of the FPGA can be reconfigured, the full programming of FPGA is not necessitated. Moreover, several applications demand to reprogram part of the configured logic, while the remaining blocks are not changed, as shown in Figure 2-4(c).

Hence, the address information is provided with the bitstream file; the configuration data might be more significant than the corresponding bitstream file size of the single context. Consequently, data overhead is examined, and full reconfiguration consumes significant time compared to the single context. Moreover, a partially reconfigurable design is optimally used when the reconfiguration data is less significant compared to the total design.

2.2.2. Benefits of using Dynamic Partial Reconfiguration

Figure 2-6: DPR saves FPGA's area [19]

The most beneficial advantage of DPR is that it provides more flexibility to the hardware designs. DPR allows the implementation of elaborated circuits within an affordable area and reduces static power consumption. Thus, DPR innovates the concept of virtual hardware [1], which is similar to virtual memory. Therefore, the contexts are mapped and reconfigured on the physical FPGA as they are cited.

DPR is used in applications that require a high level of flexibility like SDR and some embedded FPGA applications; video processing, cryptography, and genomic sequence alignment. DPR also has an essential role in implementing adaptive hardware algorithms and improving FPGA fault tolerance [2].

The DPR could be implemented to minimize the size of used resources on the FPGA, and it's entire power consumption as illustrated in Figure 2-6. DPR enables designers to achieve capability bigger than the physical capability of the FPGA. Furthermore, DPR offers other positive aspects [20]:

- **Task speed**: As the constraints of the component size is solved by using DPR, any design is accelerated by dividing it into smaller blocks and mapping them individually on FPGA.

- **Power and area reduction**: Since the rarely used functions is swapped out of the FPGA, smaller FPGAs are used. Hence, the power consumption and the permanently logic which implements the unused functions are extracted.
- **Behaviour change**: The design is reconfigured for a different function without stopping its operation, for example SDR applications.
- **Hardware virtualization**: DPR enables managing a collection of hardware components as a library by having more hardware available than that physically existing in the FPGA.

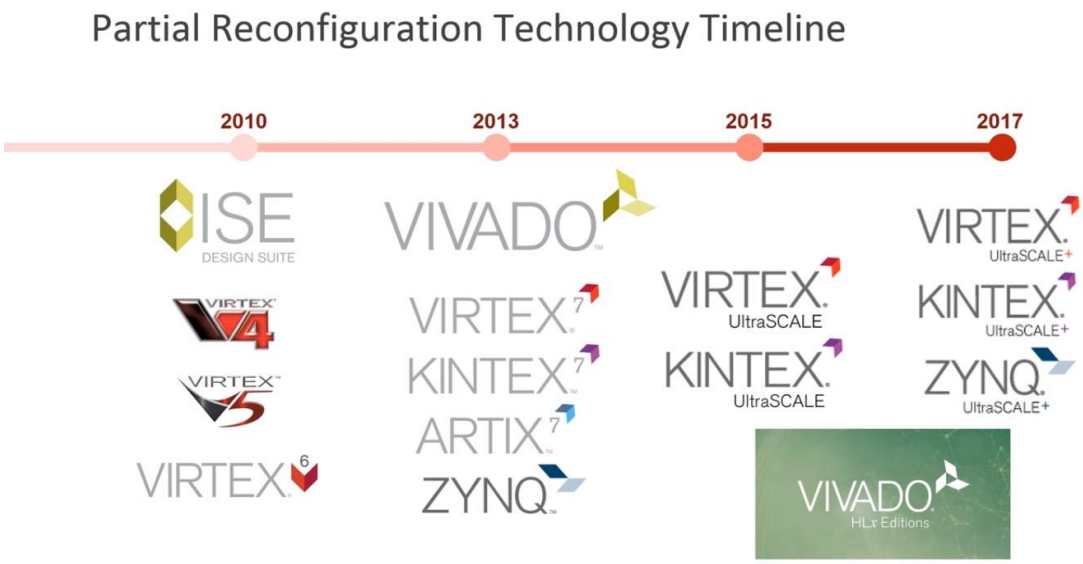


Figure 2-7: Evolution of DPR support of Xilinx FPGA and software [19]

Despite these advantages, DPR is attracting considerable interest lately due to the complicated design flow, and the lack of supporting software tools in the early stages. Consequently, the most known FPGA vendor who supports DPR in their FPGAs and the associated software is Xilinx. The evolution of their software products from complete design suite ISE for Virtex 4, 5 and 6 to VIVADO design editor is shown in Figure 2-7. A separate license is required to support DPR designs for Virtex, Kintex and Zynq UltraSCALE and UltraSCALE$^+$ FPGAs, which indicate that the DPR becomes a vital feature. Nevertheless, DPR adds more complexity to the system design. Consequently, system designers have to gain more understanding of the target device structure, the floor-planning, and identifying reconfigurable regions manually.

2.3. Network on Chip

NoC is the communication medium that is responsible for connecting the PEs through the routers. PEs are composed of logic and memory blocks.

2.3.1. Interconnection challenge

The conventional FPGA is viewed as a network of processors, compared to the multiprocessor system, where the CLBs are the processing elements. However, the conventional FPGA differs from the multiprocessor system. The conventional FPGAs have single bit processing elements and instruction control, as they are configurable with a single instruction bit per processing element.

Within the conventional FPGA layout, the PEs are disturbed in an array on the FPGA's underlying plane. However, the interconnection dominates the FPGA's area in conventional devices as shown in Figure 2-8, which represents each size of component such as interconnect, PE, and configuration memory theoretically. Since full connectivity would grow in complexity as $O(n^2)$, most FPGA's manufacturers use some advanced connection schemes to minimizing the usage of interconnects' resources [16].

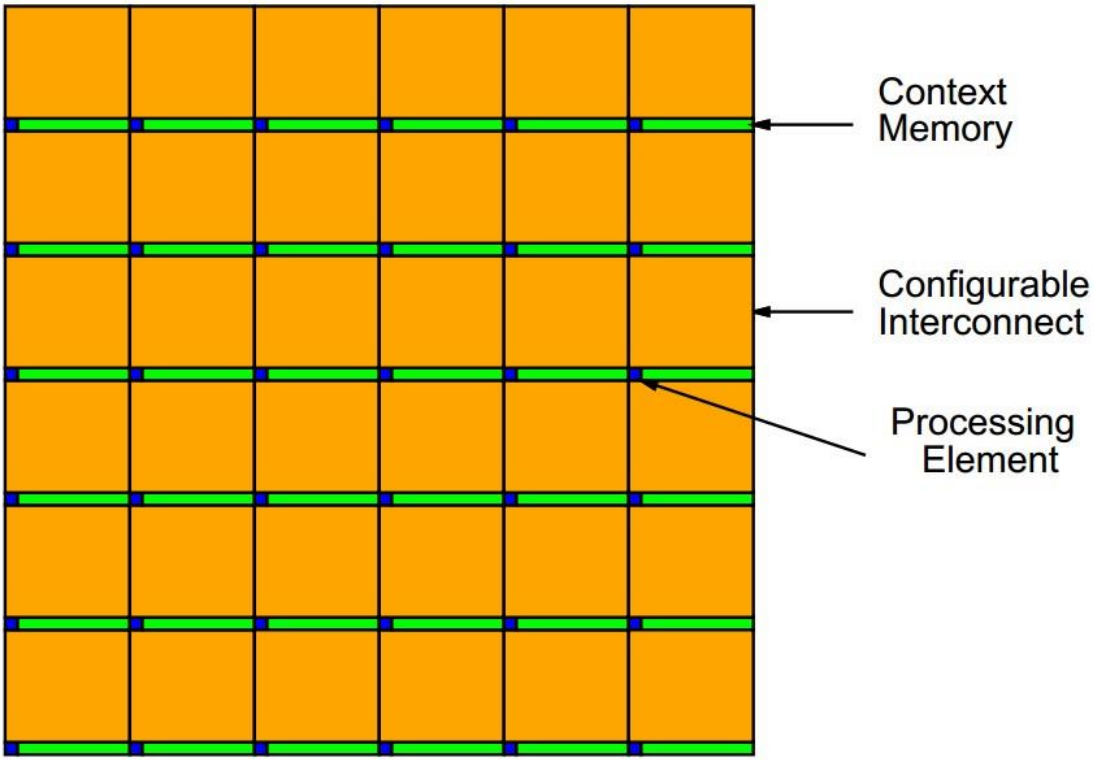


Figure 2-8: FPGA architectural as multiprocessor Array [16]

Within the last few years, communication among these PEs was destined to become a vital factor in the design of large-scale systems. Consequently, the challenges of the communication among these PEs have become significant and require innovative

solutions. Therefore, NoC has been adapted for FPGAs to handle these PEs communication challenges

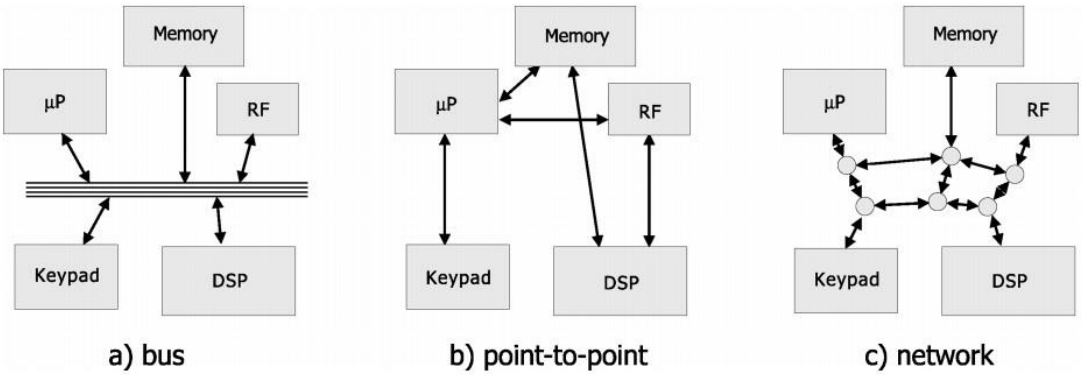


Figure 2-9: Different Communication Structures: a) bus-based, b) point-to-point, and c) Network-on-Chip [21]

2.3.2. Bus versus Network

Bus communication architecture enables the PEs to share a unique medium of communication as illustrated in Figure 2-9(a). Every PE sends or receives data, have to reserve the bus. Data can be transmitted on the bus while asserting the proper address on the corresponding bus, only when no other PE uses the bus. As listed in Table 2-1 [22], considerable arguments between the bus and network structure are discussed.

Table 2-1: Bus-versus-Network Arguments [22]

Bus Pros and Cons			Network Pros and Cons
Performance degradation	-	+	Performance is not degraded
Bus timing	-	+	Wires are pipelined
Bus arbitration	-	+	Routing decisions are distributed
Bandwidth is limited	-	+	Aggregated bandwidth
Bus latency	+	-	Internal network contention
Simple	+	-	Need re-education

2.3.3. NoC parameters

Four main metrics are considered to choose the most convenient NoC architecture. These metrics are the area, power consumption, latency, and throughput. There are other metrics which are used in evaluation such as packet loss or wire length.

Area and power consumption are related to layout implementation of network hardware components such as router modules and network interface.

The latency is the time elapsed between the emission of the head packet into the network and the time of arrival of the tail packet at the end node.

Throughput is the maximum traffic accepted by the network at a particular time, and it is measured by flits/cycle/node.

In this study, latency and throughput are used by NoC-DPR simulator to calculate the time overhead of the dynamic partial reconfiguration.

2.3.4. NoC Architecture

The NoC consists of routers, network interface, PEs, and connection links:

- Router: routes the data from its input ports to output ports according to the routing strategy.
- Network Adapter (Interface): provides an interface between the router and the processing element. Its primary task is to handle the communication between the network and the PEs.
- Process Element (Core): the main application that uses the network.
- Connection Links: the channels of communication of data between the various components of the network.

2.3.5. Topology

The topology is defined as the structure of connection between the routers and the PEs. Topologies are classified as regular and irregular, based on the location of the routers in the network. Figure 2-10 shows the following topologies:

a) Ring
b) Mesh
c) Star
d) Fully-connected
e) Mesh torus
f) Hypercube.

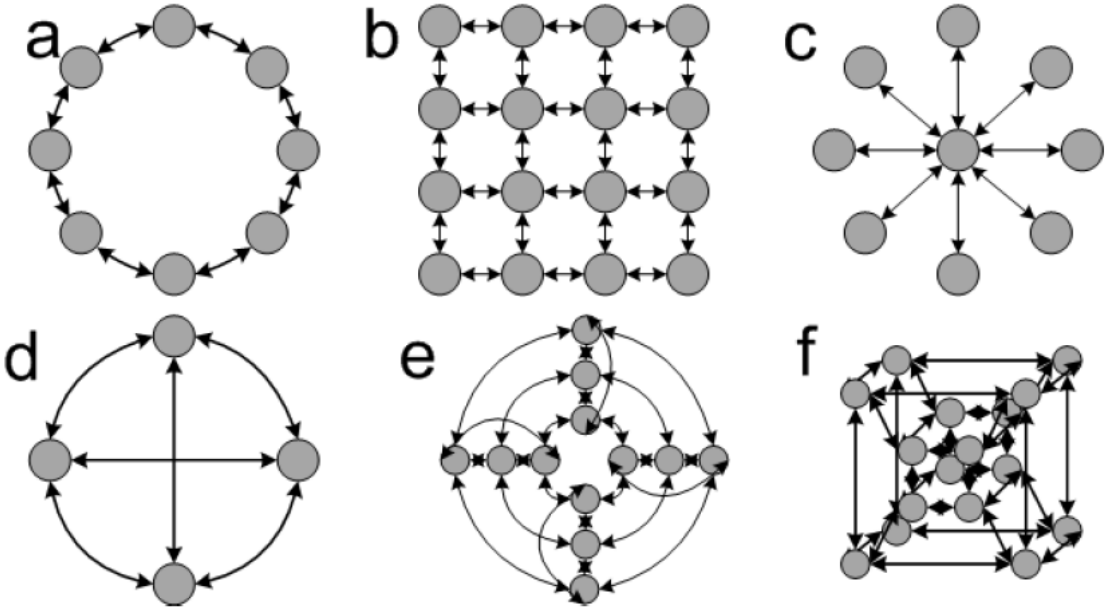


Figure 2-10: topologies for the different network [21]

2.3.6. Routing

The definition of routing is to transfer data from node to node with a predetermined algorithm. Routing is classified into the following categories:

- Distributed or centralized: distributed routing is where the flow decision is calculated locally at each node. However, routing is called centralized when the routing decisions are computed in one centralized node.
- Deterministic or adaptive: routing is called deterministic when the path is determined by the location of the source and destination node only. However, routing is called adaptive when the propagation of data between two nodes can be determined through multiple paths. The implementation of adaptive routing algorithms is resulting in complicated nodes, but it provides adequate performance of the NoC.
- Circuit switching or packet switching: The routing is circuit switching routing if the route between the start and end nodes is allocated while the data is transferred. Although, packet switching all nodes share the same path and the data is split into packets that contain the routing information.

2.3.7. Switching techniques

The main module of the router is a switch module that identifies which and when the inputs ports of a router are connected to outputs ports [23]. There are several switching techniques:

- Store-and-forward: The router splits the incoming stream into packets, at each node within the route to the destination, the router saves the packet in the input buffer, and afterwards the routing information is calculated to

identify the corresponding output port, which results in high per-packet latency.

- Virtual cut-through: this type is same as the store-and-forward but instead store all packet, it identifies the output port as soon as the first bytes of the packet (Header) is received. However, if out port is busy, the router will save the packet in a buffer [24, 25].
- Wormhole: Here the router divides the packets into flits (Flow Control Unit). The routing information is stored in the header flit. In this manner, a single packet is transferred through different nodes. That reduces the latency over the store and forward method but may cause many bottlenecks in the network.

2.3.8. Virtual channel Flow Control

Flow control addresses the issue of validating the transmitting and receiving of the packets in the network. Additionally, it resolves the problems of optimal usage of the network's resources and provides a consistent performance to all network resources.

Virtual channels (VCs) are the multiplexing of a physical channel by several logically separate channels with different buffers as presented in Figure 2-11. VCs per physical channel are varied between 2 and 16. Their implementation outcomes in an area, power and latency overhead due to the cost of multiple buffer implementation. Nevertheless, VC routers are eliminating deadlocks, enhancing wire utilization and improving network performance [26].

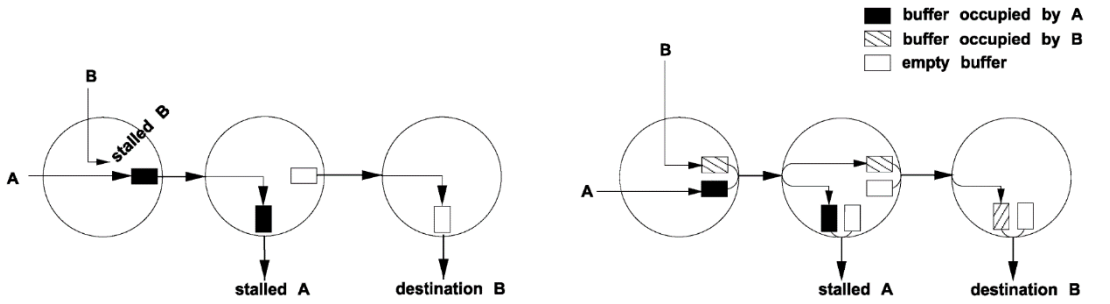


Figure 2-11: Virtual channels mechanism [21]

2.4. **Simulation Abstraction**

A simulation is to develop a model of an actual or theoretical physical system, and investigate the execution output, which is carried out on a computer. By this way, the researchers test the design space as well as evaluate the performance and efficiency for all of the new designs.

There are two types of simulation:

- Cycle-accurate simulation: In this type, the simulation is running on a cycle-by-cycle basis, which is impacting the simulation timing; as the simulator implement more details. Consequently, cycle-accurate simulation also has a cost on the development of the simulator. Cycle-accurate simulator is necessary when the actual router's RTL description is requested to be evaluated and verified.

- Event-driven simulation: This type is used when the given systems are consisting of several modules. In Event-driven simulation, events instead of accurate cycle sequence drive the flow of control within the system. The event-driven simulation uses events that occur at a various time and handles them in order of minimizing the simulation time.

2.5. **Programming Abstraction**

The following is the programming abstraction levels that should be considered when building a simulator depending on the requirements of design as shown in Figure 2-12.

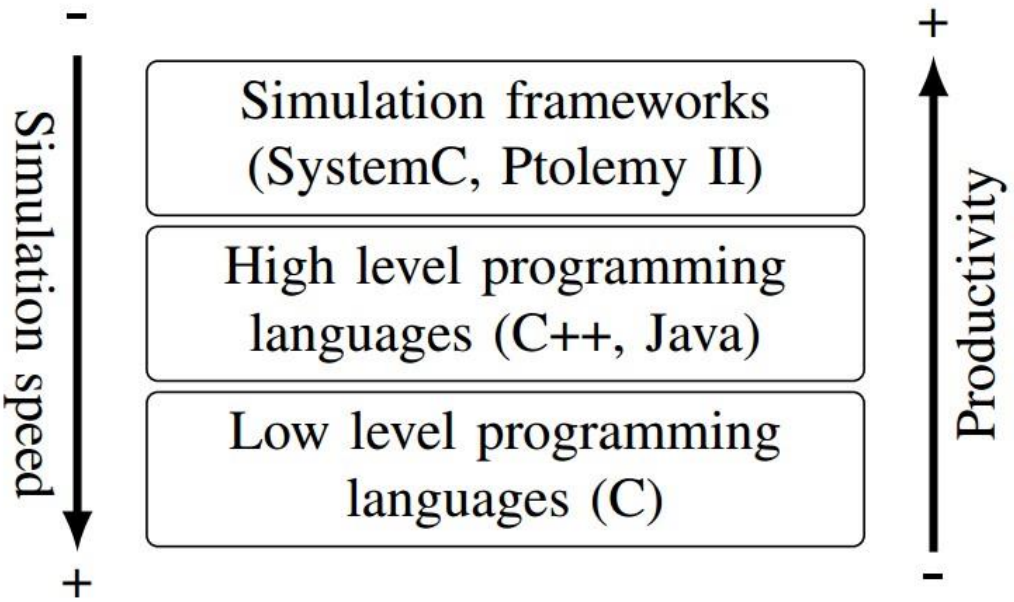


Figure 2-12: Simulation levels versus speed and productivity [27]

2.5.1. Low-level programming languages:

At low-level, the designer of the simulator may decide to use a programming language such as C. In this case; building a system using low-level programing language is a time-consuming task, due to the managing and controlling of the accurate time of concurrent processes. However, it may result in high simulation speed.

2.5.2. Object-oriented languages:

In this approach, the developer is using some high-level programming languages such as object-oriented C++ or Java. However, the developer has to manage and control the logic related to time and concurrent processes. Nevertheless, the high-level concepts of the language such as classes and inheritance facilate the designing stage.

2.5.3. Simulation frameworks:

In the simulation frameworks such as SystemC, all the administration of timing and parallel processes is handled by the framework; thus, more models are simulated within less time.

Similarly, hardware description languages provide high accuracy in addition to the shown in Figure 2-12. However, they often cause low productivity and low simulation speed.

2.6. NoC simulator comparison

Many surveys have been conducted to compare the simulations of NoC. The attempts of implementing network simulator are varying through different parameters such as the used programming languages, availability of the source code, the supported topologies and heterogeneous support of different modules at the processing element of a NoC. The summary of comparison results is listed in Table 2-2.

Several NoC simulators are developed recently, and a comparison between them is done at [31]. Some of the simulators are developed in C++ like Booksim by Jiang and al. [29]. Currently, Booksim 2.0 adds more features to perform modelling of the router microarchitecture.

Other simulators developed in SystemC like Noxim, which is developed by Palesi and al. [28], and Nirgam [32]. NoCTweak wrote in SystemC by Anh and Bevan [30] which supports router type wormhole over both synthetic traffic and embedded application patterns.

Another attempt to use high-level programming languages and higher framework such as Java and OMNeT++ which may lack the support of integration with other system written in systemC such as gpNoCsim and HNOCs [33, 34].

Table 2-2: NoC simulator comparison

Simulator	Framework	Parallelism	Topologies	Open Source	Heterogeneous Support	Synchronous/ Asynchronous
Noxim[28]	SystemC	-	Mesh	+	-	Synchronous
BookSim [29]	C++	-	Many	+	-	Synchronous
NoCtweak [30]	SystemC	-	Mesh	+	+	Synchronous
Nirgam [32]	SystemC	-	All	+	-	Synchronous
gpNoCsim [33]	Java	-	All	+	-	Both
HNOCS [34]	OMNeT++	+	All	+	+	Both

Another architecture with similar features is ReNoC which is developed by Stensgaard and Sparse [36]. The latter architecture allows the configuration of NoC's resources. Thus, the mapped application can customize the topology according to its requirements on SoC platform.

Other solutions endeavour to use NoC as a backbone in FPGAs system to overcome communication challenges, such as Ehliar and Liu [35] that proposes an open source FPGA based NoC architecture with low area overhead, high throughput, and low latency compared to the general NoC performance.

Chapter 3 : Configuration Techniques for Xilinx FPGAs

3.1. Introduction

In this chapter, the different interfaces to configure Xilinx FPGA with a partial bitstream are studied. Xilinx offers four methods to send or receive a partial bitstream from non-volatile memory into the reconfiguration memory. A comparison between these techniques is provided concerning the area and the reconfiguration time using part of an SDR system as a benchmark for DPR.

3.2. Methodology

Xilinx FPGAs have four various techniques to carry out DPR. Figure 3-1 demonstrates the different reconfigurations techniques to perform DPR using an encoder inside the communication chain of an SDR system.

All procedures require a controller of PR. The controller is placed outside the FPGA, such as external PC connected to JTAG or Serial port as illustrated in Figure 3-1(c).

Nevertheless, the controller is placed inside the FPGA's fabric, such as MicroBlaze soft microprocessor IP to manage ICAP or selectMap as shown in Figure 3-1(a) and Figure 3-1(b).

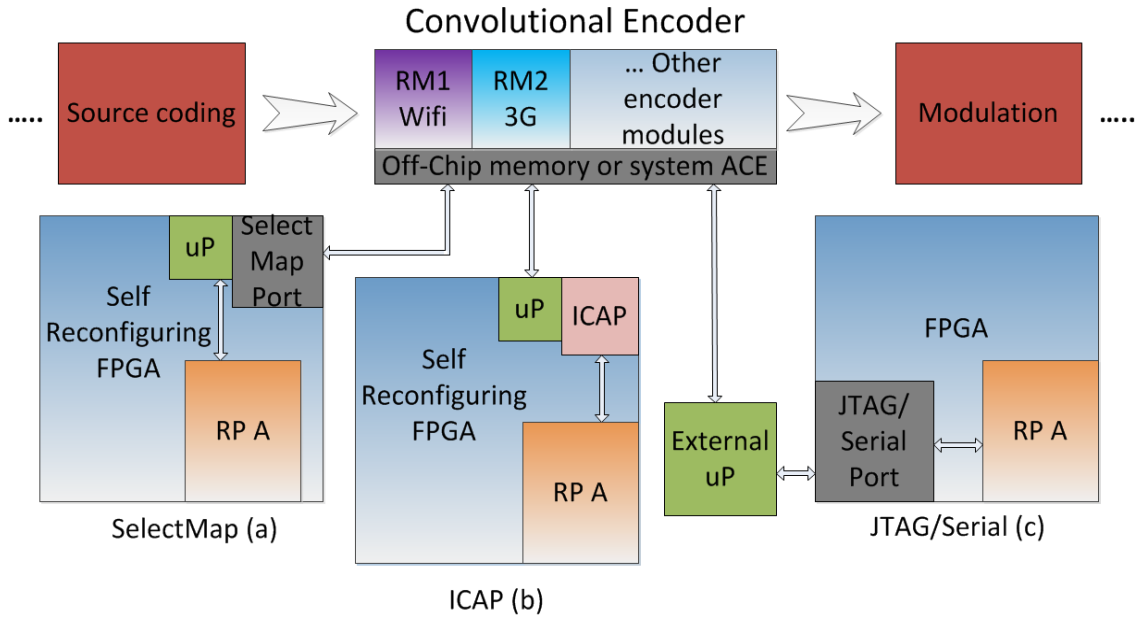


Figure 3-1: Reconfigurations techniques of convolutional encoder inside the communication chain [1]

3.2.1. JTAG

The JTAG is an acronym for standard Group named Joint Test Action Group. JTAG sends data out through I/O ports for testing connections on board level testing. Therefore,

it is widely used as an essential debugging tool. The JTAG sends signals inside the chip for testing device behaviour, these patterns of test aim to detect shorts and opens at the board and device levels.

The JTAG configuration is conducted using the iMPACT tool and Xilinx programming cable in Figure 3-1(c). The partial configuration is completed by obtaining the bitstream file located on the computer [37].

3.2.2. Serial Mode

Throughout slave serial configuration mode, the configuration data is loaded one bit per Configuration Clock (CCLK) cycle. The CCLK in the serial slave mode must be driven from exterior control logic. The Serial Slave mode generally used in the configuration of the single device from an external microprocessor as illustrated in Figure 3-1(c) or configuring multiple devices in a daisy chain.

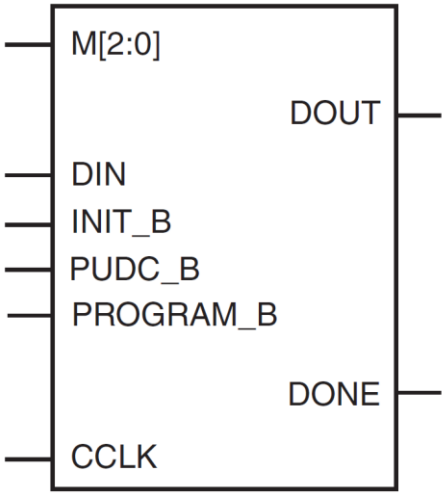


Figure 3-2: Virtex-5 FPGA Serial Configuration Interface [38]

Six pins are required to accomplish the reconfiguration procedure using Serial Slave mode as displayed in Figure 3-2. A single configuration is employed to configure multiple devices arranged in a daisy chain [39]. Each device receives the configuration data via its DIN pin and crosses it to the next device through its DOUT pin till the last device in the chain is configured, and all the devices discharge their DONE pins.

3.2.3. SelectMAP

SelectMAP is a configuration interface that provides an 8-bit, 16-bit, or 32-bit bidirectional data bus interface to FPGA's fabric, which is often used for both configuration and readback. It also operates in two modes; a master mode that drives configuration clock, or slave mode which is driven by an external configuration clock. Read-back is applicable only to Slave SelectMAP mode.

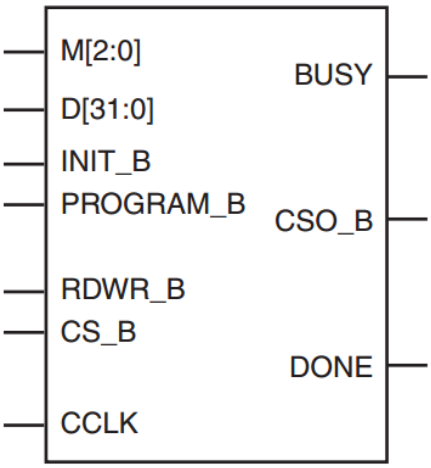


Figure 3-3: Virtex-5 FPGA SelectMAP Configuration Interface [38]

There are various setups for SelectMAP like single device slave SelectMAP that includes a processor providing data and clock. Alternatively, a CPLD is used as a configuration manager [38]. Another installation is multiple device daisy chains that can be used to configure various FPGAs in series with different bitstreams from a nonvolatile memory or processor.

Slave SelectMAP is the only mode that allows performing partial configuration in all Xilinx FPGA's as master modes are clearing all FPGA's configuration memory Figure 3-1(a).

The reconfiguration process using SelectMAP in slave mode is carried out using 38 pins as shown in Figure 3-3. Multiple FPGAs can be connected on SelectMAP bus which shares some pins with others FPGAs.

3.2.4. ICAP

ICAP is Xilinx primitive that offers direct access to the configuration logic at the FPGA fabric Figure 3-1(b). At runtime, the ICAP interface permits the configuration data to be loaded into or downloaded from the configuration memory of the FPGA. Additionally, it enables reading the status registers of the configuration logic.

The ICAP interface is similar to the SelectMAP slave mode interface but with separate 8-bit, 16-bit, or 32-bit data bus for reading and writing configuration data [39].

Configuration data is written to the FPGA’s memory with a fixed clock. Even though there are two ICAP primitives available starting from Virtex-5, the two ports cannot be operated concurrently. Consequently, the design has to start with the top ICAP, and then alternate between the two ports. ICAP caches the configuration bits into BRAM before they are loaded to the FPGA configuration memory.

Xilinx provided IP core known as OPBHWICAP, which is connected on the OPB bus, it enables the processor to access the configuration memory through the ICAP using the library and software routines that have been implemented by EDK toolkit. In Virtex-4 and Virtex-5 FPGAs, the XPSHWICAP then AXI_HWICAP has been released which works similarly with the OPBHWICAP, however the IP is connected to the PLB and AXI bus respectively [40]; thus, a lower-latency reconfiguration is obtained.

3.3. Software Defined Radio design

Lately, SDR turned to be a trending application for DPR. As wireless technologies maintain their growth and developing, more standards will be released. Therefore, the demand to preserve these entire standards in one device is required. Hardware designs that are trying to offer compatibility with the current standards, if even possible, will most likely become outdated after a short while. Nevertheless, the SDR system maintains the flexibility to control the same hardware resources via software for these multi-communication devices.

SDR systems make use of the reconfigurability of FPGAs because it provides good assets to load the desired standard. Practically, the ability to reconfigure a specific block, while all other blocks are working regularly, provides an opportunity to create a flexible and compact design.

The benefits of SDR system is noticed undoubtedly by implementing DPR approaches on Convolutional Encoder block. This encoder is accountable for generating FEC coding schemes. These coding schemes allow reducing channel noise. Convolutional encoder outputs are not impacted by the code schemes used in the existing standard but also several parameters (n, k, l), which are being used for explaining convolutional codes. Where n represents the input encode elements, k represents the output encode elements, and l represents the number of shift registers.

In this experiment, two encoder schemes; 3G and Wifi communication systems, are used as a benchmark for DPR as revealed in Figure 3-1. This approach is called Single-Loaded Encoder Module (SLEM) wherever DPR is used to implement one encoder on the chip at a time.

3.4. Results and Discussions

The experiment is designed to apply DPR to the implemented SDR design using different configuration approaches and to compare them in terms of area and reconfiguration time. This design has been implemented using XUPV5-LX110T kit which includes Virtex-5 xc5vlx110tff1136-1 FPGA, SystemACE Compact Flash configuration controller to store bitstream files of PR regions, and UART interface to interact with MicroBlaze by sending reconfiguration commands.

3.4.1. Experiment setup

The DPR time is not associated directly with design resources, but it is related only to partial reconfiguration region selection which is translated to frame's number.

The frame is the minimum addressable configurable part of the FPGA, which spans multiple Programmable Logic Blocks (PLBs) in array usually the entire column of PLBs.

Previously, in Virtex and Virtex II families, frames consist of the whole column of FPGA, which are the minimum building blocks for PR region. Starting form Virtex 4, frames became a complete tile which includes a certain number of CLBs of an entire column, and this number is increasing in each new Xilinx family.

Therefore, the total design size, static and PR regions, has a significant effect on the reconfiguration time. Hence, the selected size for the SDR design is varied along the experiment to check the variation in performance of each configuration technique. The various selection is chosen to take into consideration a significant change in the partial bitstream file size that is to reflect in the estimated reconfiguration time.

Table 3-1: Reconfiguration speed for different reconfiguration techniquess

Config. Mode	Data Width	Max. Clock Rate	Max. Bandwidth
JTAG	1-bit	66 MHz	66 Mbps
Serial Mode	1-bit	100 MHz	100 Mbps
ICAP	8/16/32 bits	100 MHz	0.8/1.6/3.2 Gbps
SelectMAP	8/16/32 bits	100 MHz	0.8/1.6/3.2 Gbps

The figure of merit, which is chosen for the comparison between these different techniques, is the area multiplied by reconfiguration time. This metric is an excellent indicator of the performance variation from a particular design size to another. Also, the number of occupied LUTs is also considered as a significant indicator of the design area as shown at the vertical axis and the horizontal axis in Figure 3-4.

The theoretically estimated reconfiguration time is calculated according to (1), where Bs_{size}, is the bitstream file size of PR region, Clk_{max} is the maximum clock rate supported by reconfiguration interface, and Dw is interface data width. These values are listed in Table 3-1 for each interface.

$$ReconfigurationTime = Bs_{size}/(Clk_{max} * Dw) \qquad (1)$$

The reconfiguration region sizes are chosen in a way to occupy a certain number of frames completely. The selection is made to make use of the whole area without any change in the partial bitstream size and without affecting the estimated reconfiguration time.

3.4.2. 8-bits Data Width

Figure 3-4 shows the performance of the JTAG, Slave Serial mode, Slave SelectMAP 8-bit, and ICAP 8-bit data width using the SDR design with different selections of PR regions. It is shown that at small designs that need PR regions less than ~400 and ~750 LUTs for JTAG and Serial mode respectively, JTAG and Serial mode are better in performance than ICAP and SelectMAP which work with 8-bit width at 50 Mhz. These values decreases (~150 and ~300 LUTs) when ICAP and SelectMAP work at 100 Mhz taking into consideration that ICAP allows maximum frequency less than JTAG and Serial mode. This can be avoided when using SelectMAP as it has external CCLK port.

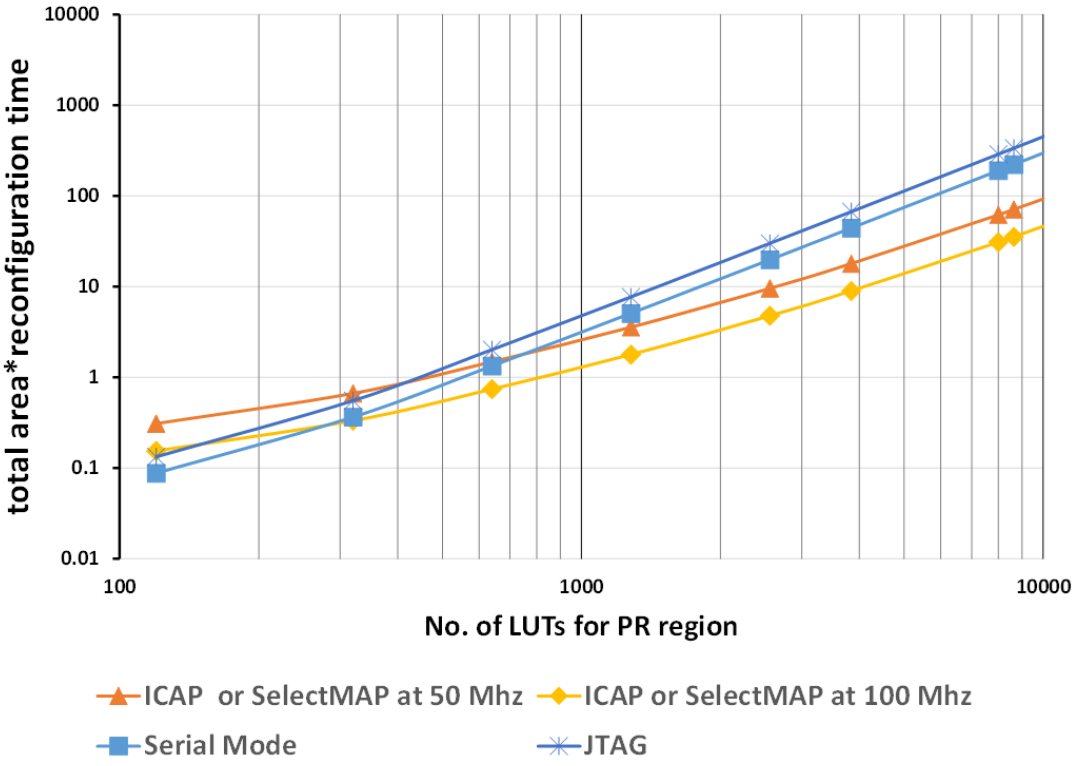


Figure 3-4: 8-bits ICAP and SelectMAP with Serial mode and JTAG

3.4.3. 16-bit Data Width

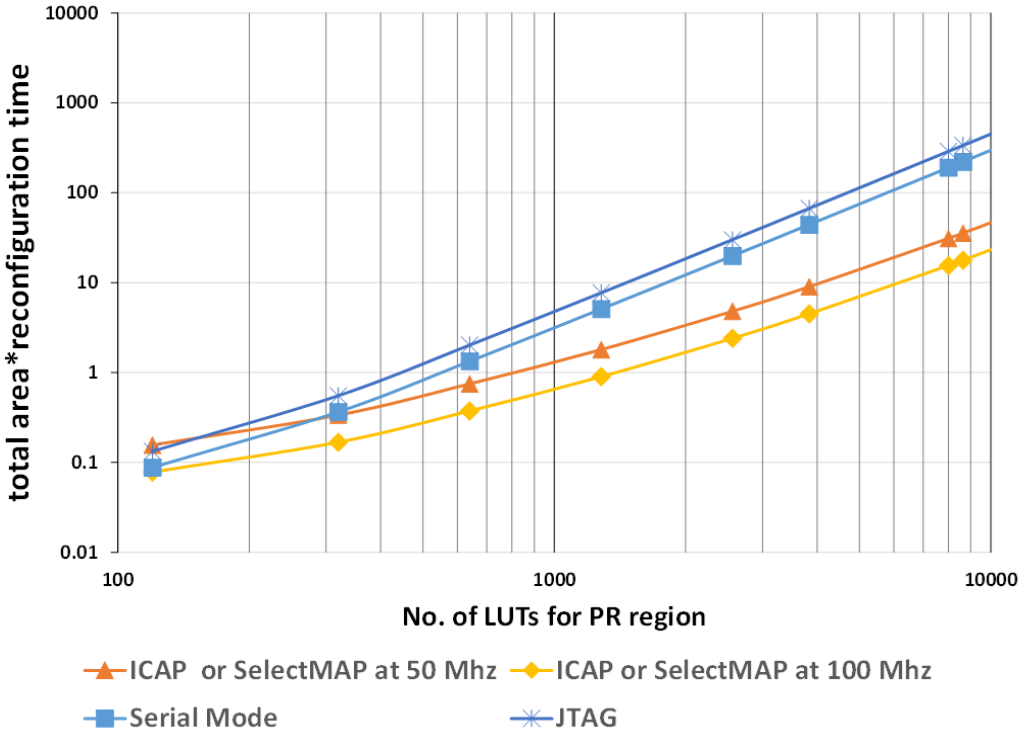


Figure 3-5: 16-bits ICAP and SelectMAP with Serial mode and JTAG

In Figure 3-5, the experiment is repeated with 16-bits data width for ICAP and SelectMAP. It is noted that the intersection points decreased more (~150 and 300 LUTs for JTAG and Serial mode respectively) compared with ICAP working at 50 MHz. These values decline because the comparison becomes unfair for serial interfaces like JTAG and Serial mode compared to 16/32-bits ICAP and SelectMAP as the parallel configuration always gives more capability to reach high configuration speed.

Designs which are using JTAG and Serial mode can save ~2400 LUTs compared to ICAP and SelectMap; this overhead is significant with small area designs. However, the reconfiguration speed of 16-bit ICAP and SelectMap is better with a factor of 24.2 and 16 than JTAG and Serial mode respectively.

3.4.4. 32-bits Data Width

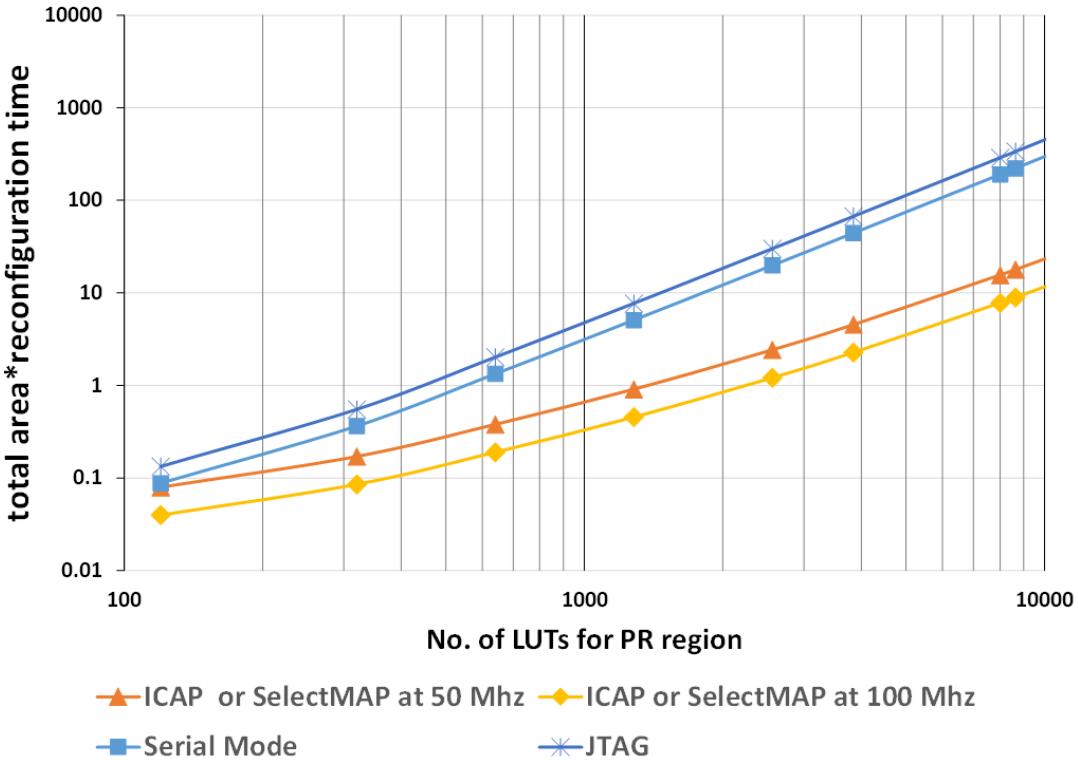


Figure 3-6: 32-bits ICAP and SelectMAP with Serial mode and JTAG

ICAP and SelectMAP will always be recommended if they operate at the full data width 32-bits over other reconfiguration techniques as shown in Figure 3-6. However, the drawback of this scheme is the used I/O pins. In SelectMAP, those pins have to be reserved the whole time for reconfiguration purpose only, whereas those pins can be used as general I/O after the reconfiguration has been done in ICAP.

Table 3-2 lists all experimental results and indicates where the serial configuration techniques such as JTAG and Serial Mode are used instead of parallel procedures such as ICAP and SMAP.

Table 3-2: Serial reconfiguration techniques recommendation over SMAP and ICAP

Configuration Mode	8-bit	16-bit
JTAG	<400 LUTs	<150 LUTs
Serial Mode	<750 LUTs	<300 LUTs

3.5. Summary

In this chapter, the four configuration methods, which are used with DPR in Xilinx FPGAs, are reviewed.

It is evident that JTAG reconfiguration is much slower than the other methods, but it does not add resources overhead like the other methods. Therefore, the performance with JTAG is better than the others with small designs where the area overhead is very noticeable. Despite that, the performance is not good with large designs where the space cost is not reasonable compared to the design area. On the other side, it allows sending internal signals through dedicated IO pins for debugging. Additionally, the methods that use a parallel port support a high-speed reconfiguration compared to the others, especially with large designs.

Chapter 4 : NoC-DPR Simulator Architecture

4.1. Introduction

The contribution of this chapter is proposing a cycle-accurate simulator for NoC, which is a state-of-art tool called NoC-DPR that is used to simulate DPR on NoC-based FPGA. Two open source SystemC components are linked; the first is a NoC simulator, which is known as NoCTweak, and the second is a SystemC Library (ReChannel), which simulates DPR for general purpose designs. All PEs of the network can be reconfigured dynamically to adopt a new design at the run-time.

4.2. NoC-DPR Simulator Architecture

NoC-DPR simulator is a command line-based tool that consists of a 2-D mesh network of routers, simulated by NoCTweak [29]. Each node includes a PE, Network Interface (NI), and an associated router. Each router connects with four nearest neighbouring routers forming a 2-D mesh network as illustrated in Figure 4-1. Using ReChannel [41] library, each PE is dynamically reconfigured by a particular type of data packet, generated from a specific node (master node 0, 0).

The primary consideration that must be taken, when merging DPR simulation library with NoC simulator, is all NoC modules should be well-defined through a definite hierarchy at SystemC.

Consequently, a separate NI is implemented with NoC-DPR simulator as displayed in Figure 4-2. Accordingly, DPR is performed on the PE, and NI supports the flow control over receiving and sending data during the DPR operation. However, the network's latency and throughput values have changed due to this modification.

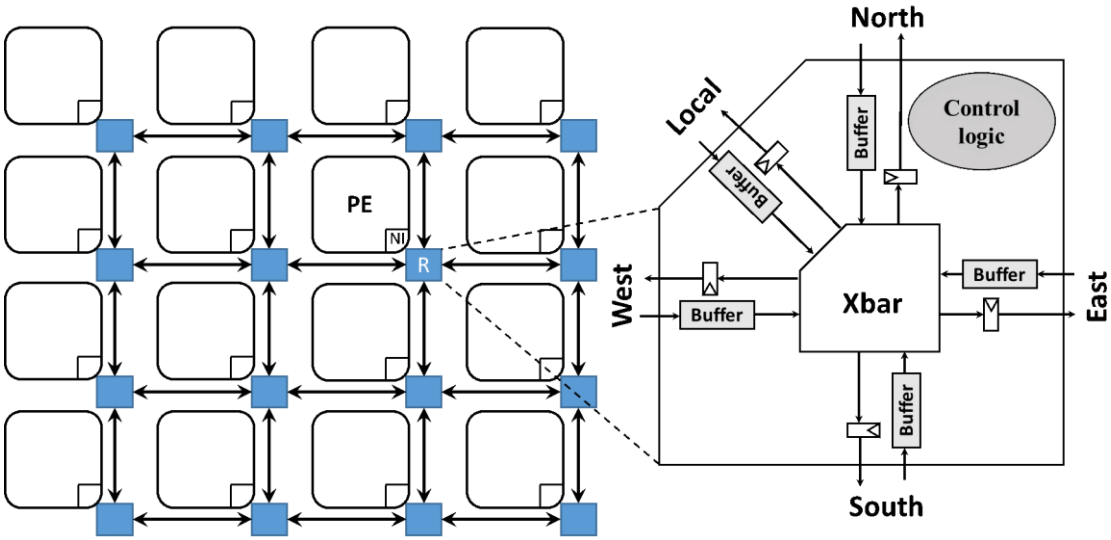


Figure 4-1: Network on a chip of NoCtweak simulator [52]

4.3. NoCTweak simulator

NoCTweak is an open-source 2-D mesh NoC simulator, which is designed for early exploration of the performance of on-chip networks. The simulator has been developed using SystemC, which allows accurate and fast modelling of concurrent hardware modules at the cycle-level accuracy [30].

The NoCTweak simulator is composed of a hierarchy of modules; a processor (core), a NI as drawn in Figure 4-5, and an associated router that implements different functions of the network and the simulation environment as portrayed in Figure 4-1.

Each of these modules has a clear interface that facilitates replacement and customization of module implementations without affecting other parts of the simulated system.

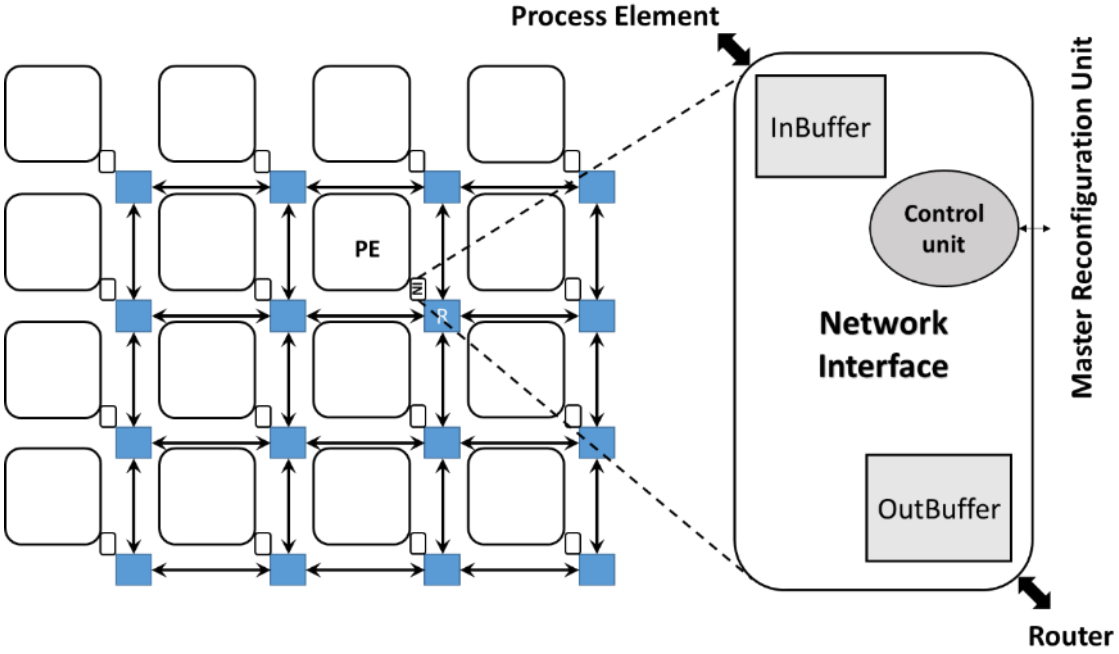


Figure 4-2: Network on a chip of DRP-NOC simulator [52]

4.3.1. Network Latency

Packet latency includes the travelling and waiting time from the source queue to the destination, the waiting time is caused by network congestion. In the simulation, when the PE receives a packet, it calculates the difference between the packet's creation time from the received time to get the packet latency. Thus, network latency is the average of latencies of all the packets transferred by the network.

Let L_{ij} be the packet latency of packet j and N_i be the number of packets received by processor i, then the average network latency is given by [30], where N is the number of processors in the platform:

$$L_{avr} = \frac{1}{N} \sum_{i=1..N} \left(\frac{1}{N_i} \sum_{\forall j} L_{ij} \right)$$

4.3.2. Network Throughput

Network throughput is the rate where the network efficiently accepts and delivers the injected packets. Let T_{sim} and T_{warm} be the simulation and warmup time, then the average network throughput (in packets per unit time per node) is given by [30]:

$$T_{avr} = \frac{1}{N(T_{sim} - T_{warm})} \sum_{i=1..N} N_i$$

The network latency is derived in cycles or seconds and the network throughput in flits per cycle or flits per second. All these kinds of terms are shown in the output results of NoC-DPR simulator.

4.4. ReChannel Simulation Library

Initial attempts to design for DPR was considered a slight complex task due to the lack of supporting tools and the requirement of full understanding of the FPGAs architecture. Therefore, FPGA designers use DPR simulators at early design stages as a proof of concept and to reduce the time to market.

Several approaches [42, 43, and 44] have been proposed to model dynamically reconfigurable systems at system-level using SystemC, which is C++-based description language used at higher abstraction levels to develop complex systems.

In [45], the OSSS+R framework is presented along with the design methodology for automatic modelling, synthesis, and simulation of partial run-time reconfiguration systems. However, these methodologies do not support functional verification of customized DPR system designs.

The modelling is performed using object-oriented techniques in ReConLib library [41]; nevertheless, the limitations of SystemC in modelling dynamic reconfiguration are avoided.

As the main challenge when modelling dynamic designs using SystemC is the inability of performing changes to the system's module topology during simulation. Consequently, this inability leads to difficulties in the modelling of reconfigurable systems using hardware description language (HDL) without modifications. The ReChannel library [41] is an extension to SystemC, not an adjustment to the SystemC's kernel as conducted in previous projects. Hence, ReChannel overcomes SystemC modelling limitation without actually changing the underlying simulation kernel.

On the other hand, several simulators simplify the DPR modelling process; consequently, the reconfigurable modules are switched between two states; activated and deactivated. That is achieved in ReChannel Library [41] through the concept of switches (portals) as shown in Figure 4-3. Portals allow the utilization of any SystemC channel in a reconfigurable context, which results in reconfigurable modelling systems with a highly flexible methodology. Furthermore, the first modification of the original system occurs within the interconnection between other modules of the system, i.e., between NI and PE interconnect signals as illustrated in Figure 4-4. Hence, portals are added to eliminate any required changes to the existing modules.

That facilitates the interface of reconfiguration parts to static parts. Reconfiguration properties, such as reconfiguration time, are added to those modules using an argument parsed by the command line to the NoC-DPR simulator.

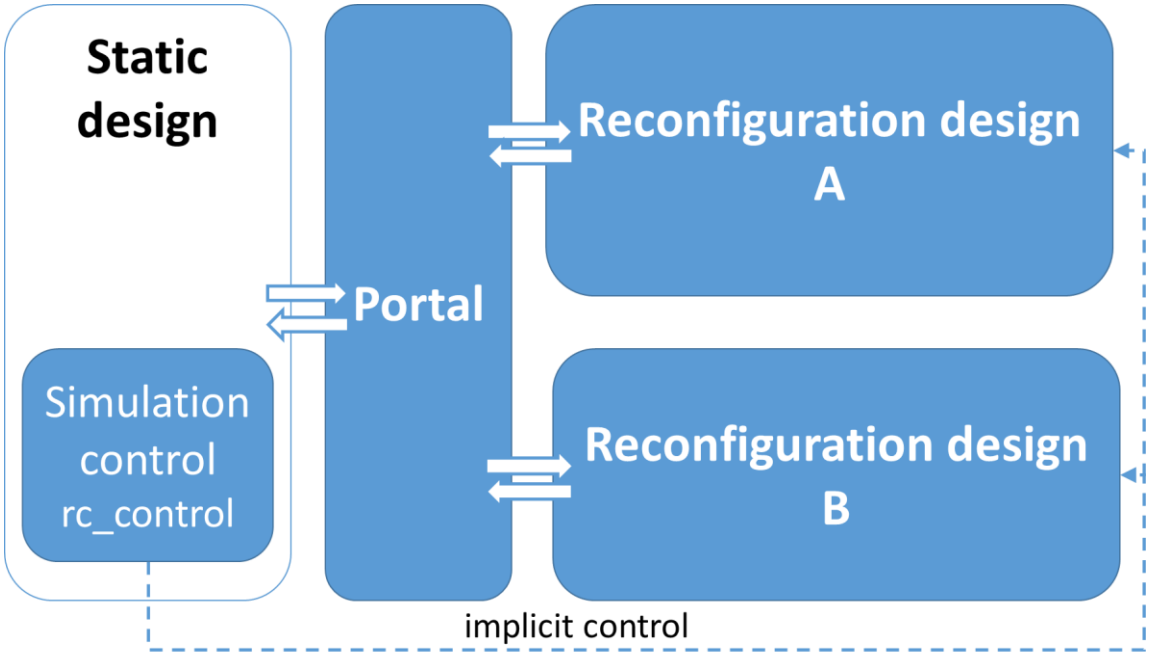


Figure 4-3: A portal connecting two reconfigurable modules to a standard channel SystemC

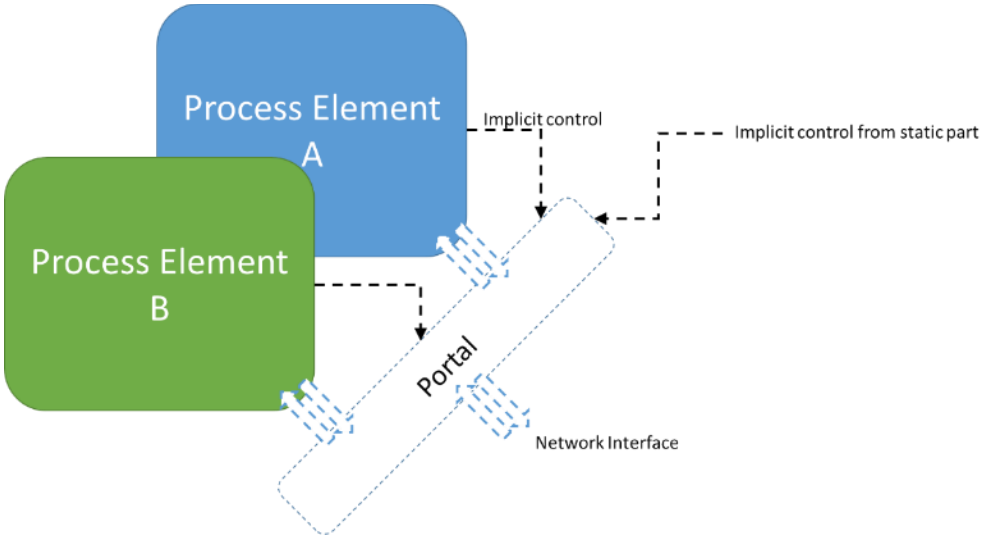


Figure 4-4: DPR simulation flow at systemC

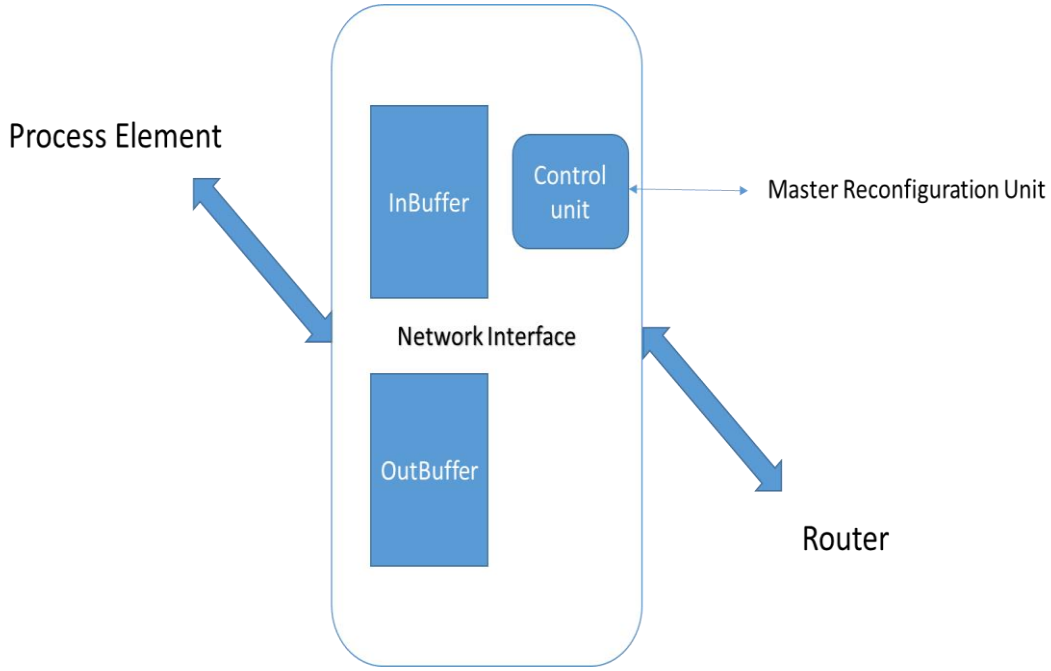


Figure 4-5: Network Interface of DRP-NOC simulator

4.5. Network Interface impact

Inserting an explicit network interface, as shown in Figure 4-2 and Figure 4-5, between the PE and the router, affects the network performance specifically on the latency and the throughput. NI is composed of two decoupling buffers that are responsible for storing and synchronizing flits (a flit stands for FLow control unIT, which is the minimum unit of the message).

The latency after inserting NI is measured and compared to the latency of NoCTweak. A network of wormhole routers with buffer size 2-flits per input port running at 100MHz is considered for this simulation along with different Flit Injection Rates (FIRs) on different network sizes. The five network sizes: 2x2, 4x4, 8x8, 16x16 and 18x18 are adopted to cover all performance variations. Figure 4-6 shows the methodology of NoC-DPR simulator. Network parameters are passed to the simulator through the command line, then the simulation using the parsed parameters is launched. Finally, all simulation results are collected using automation scripts, which run and manage the results over the different network sizes and the different injection rates to draw all following listed figures.

Figure 4-7 and Figure 4-8 show the difference between the latency of NoCTweak simulator and the proposed NoC-DPR simulator. NoCTeawk's latency reaches above 50,000 cycles while NoC-DPR's latency saturates at 23,000 cycles for the maximum supported network size 18x18. This enhanced latency value is due to the NI of NoC-DPR simulator, which is responsible for controlling packet generation from PE according to NoC state. Therefore, NI sends and receives control signals to PE so that it prevents network congestion and deadlocks.

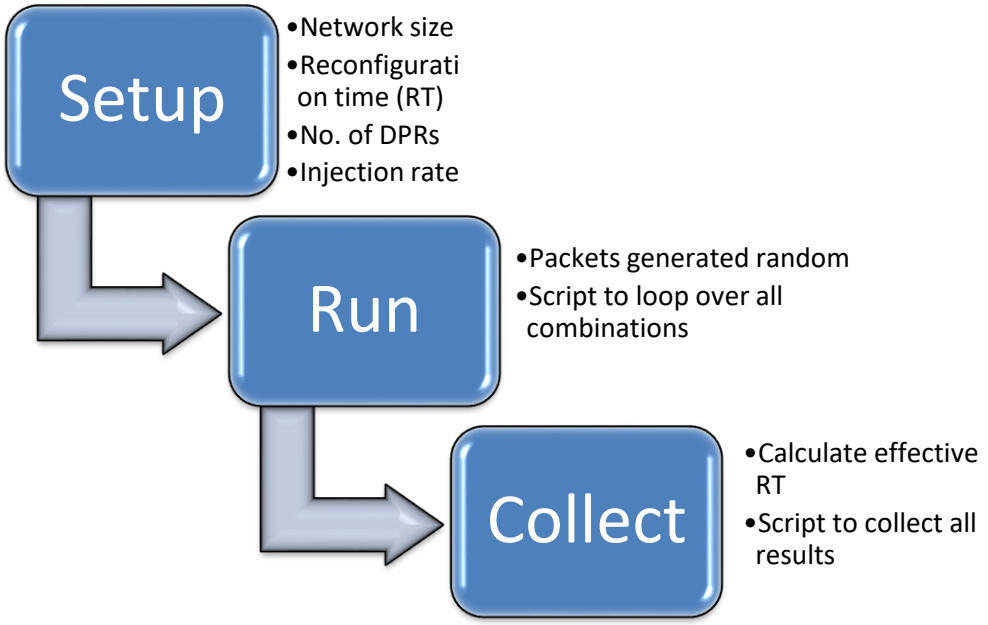


Figure 4-6: Methodology to setup, run and collect simulation results from NoC-DPR simulator

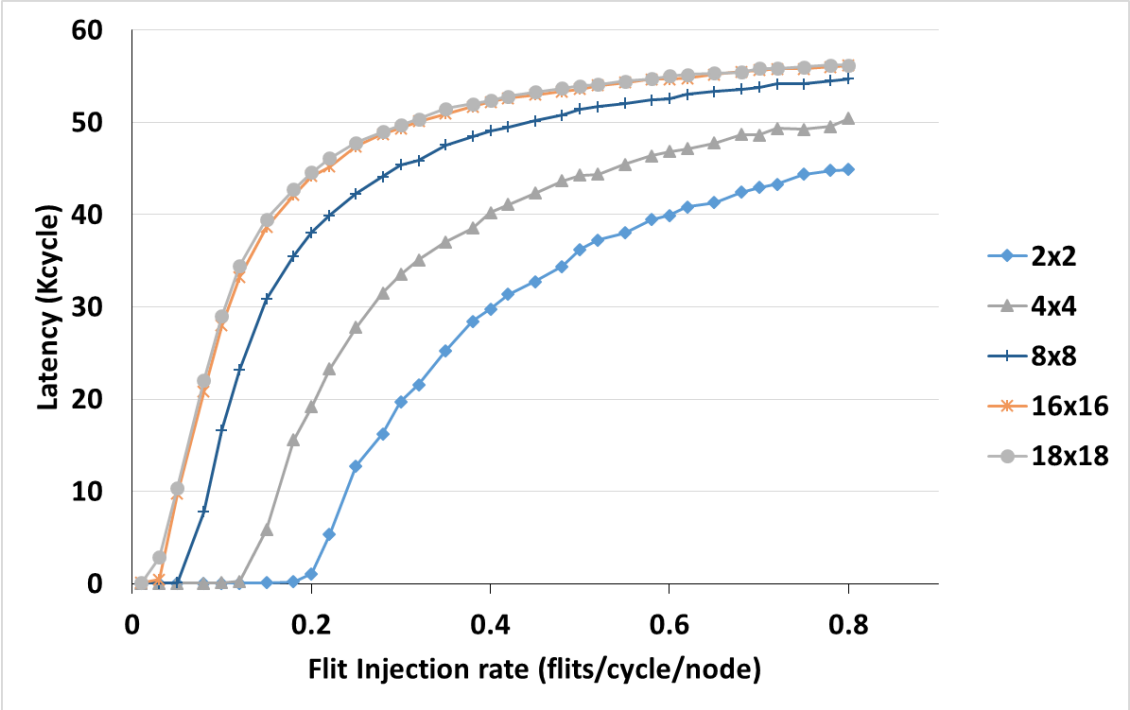


Figure 4-7: Average latency for 2-flits buffer depth for NoCTweak simulator without PE

On the contrary, in NoCTweak, the PE has no output buffer; therefore, PE injects flits directly into router input buffer without any flow control. Consequently, the latency increases as long as flits are injected as portrayed in Figure 4-7, unlike NoC-DPR which

saturates at lower latency 5,000, 7,500, and 14,000 cycles for 2-flits buffer size for 2x2, 4x4, and 16x16 NoC size respectively as shown in Figure 4-8.

On the other hand, in the NoCTweak simulator, the throughput is elevated at specific values as depicted in Figure 4-9, because of the direct injection to NoC; as the network is loaded with maximum accepted flits at higher FIR (Flits Injection Rate of each PE). However, In NoC-DPR simulator, the PE stops packet generation when network state is fully loaded, which causes a slight increase in the peak value of the throughput. The throughput decreases exponentially because the network is not loaded with maximum accepted flits as shown in Figure 4-10. The saturated values of the throughput are 0.21, and 0.14 (flits/cycle/node) in NoCtweak, and become peak values of 0.22, and 0.155 (flits/cycle/node) at 0.22 and 0.18 (flits/cycle/node) FIR in NoC-DPR simulator for 2x2 and 4x4 NoC size respectively.

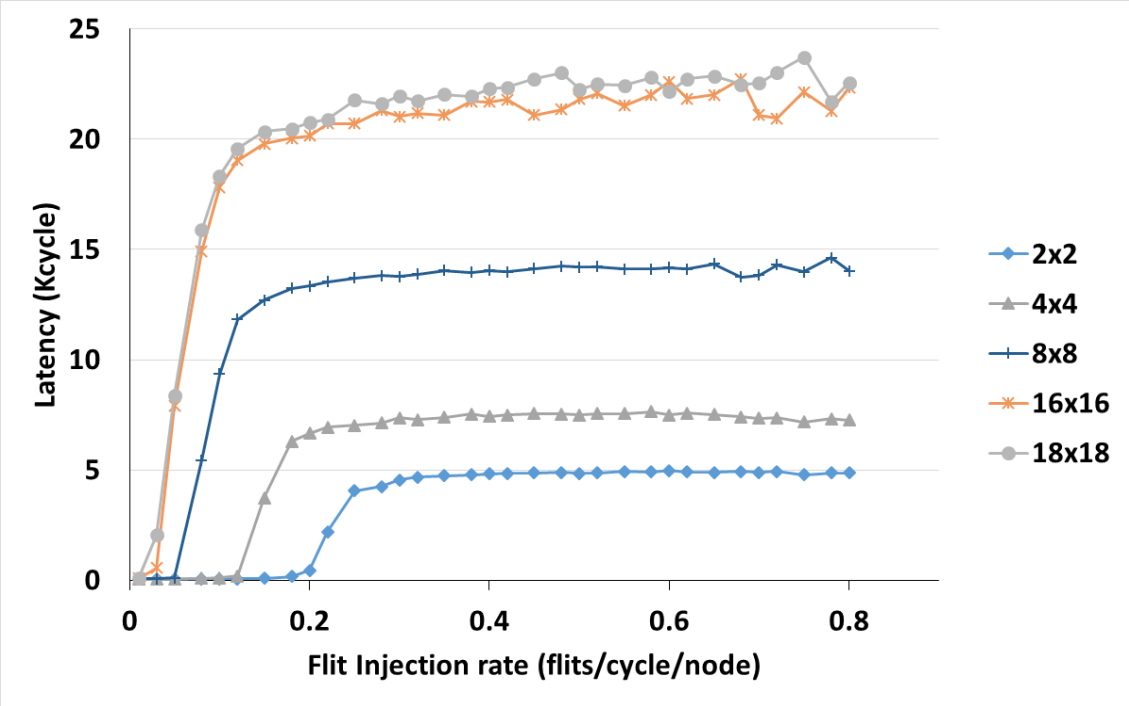


Figure 4-8: Average latency for 2-flits buffer depth for NoC-DPR simulator

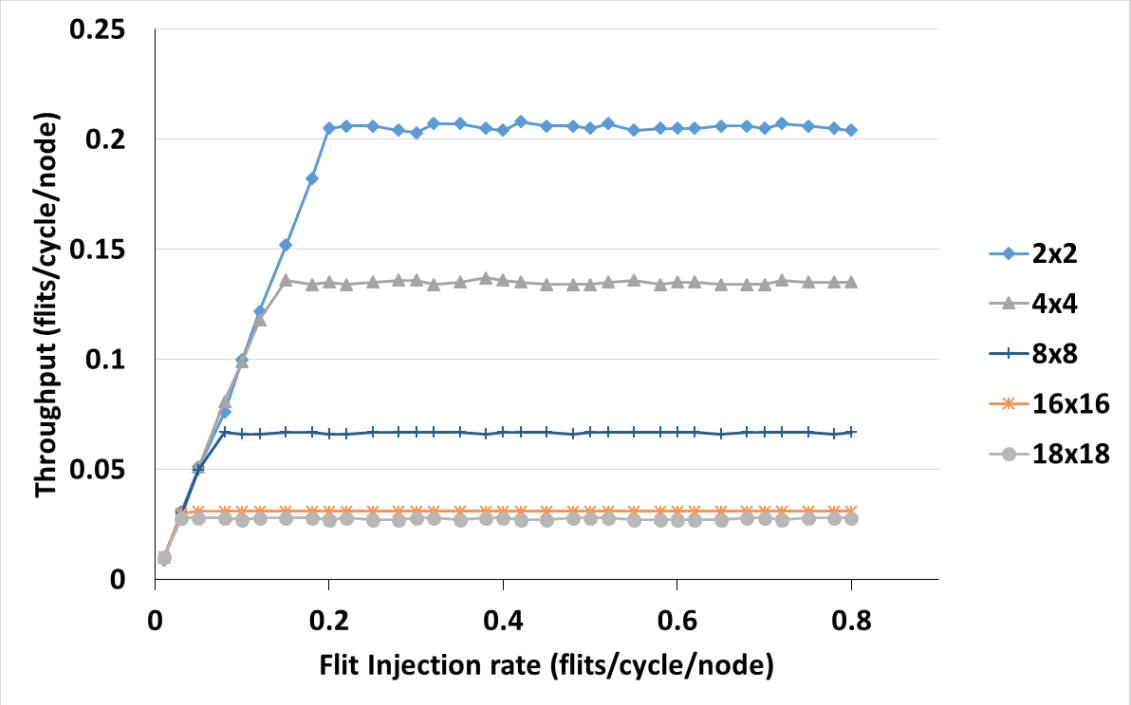


Figure 4-9: Average throughput for 2-flits buffer depth for NoCTweak simulator without PE buffer

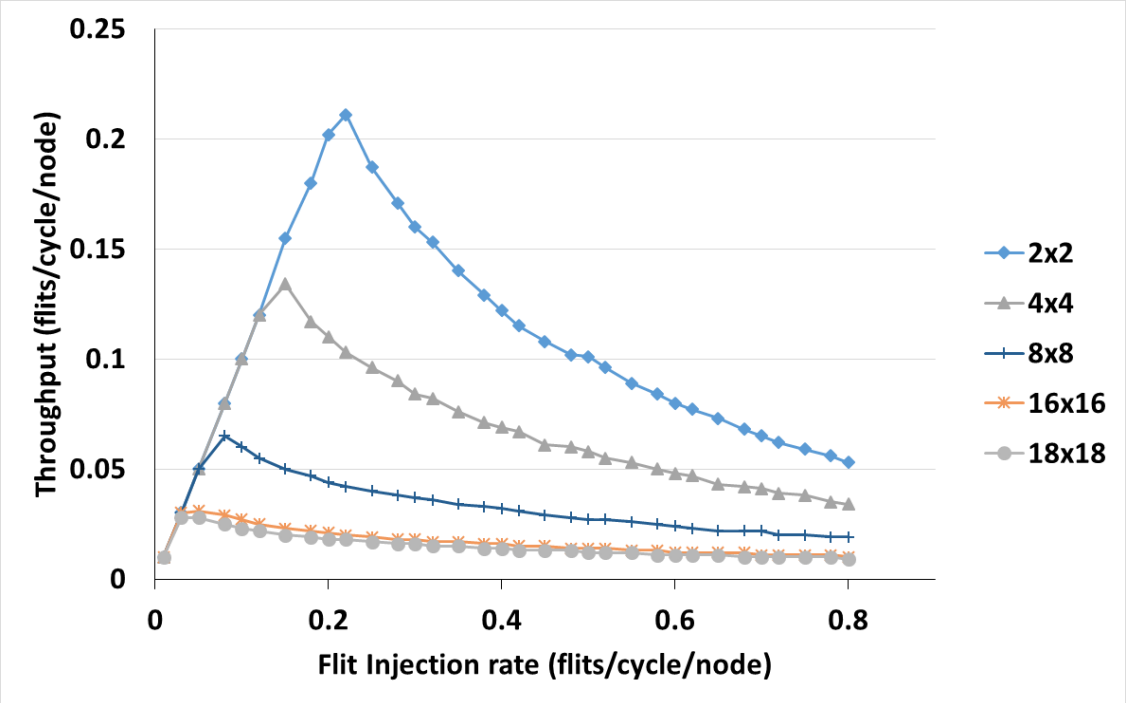


Figure 4-10: Average throughput for 2-flits buffer depth for NoC-DPR simulator

4.6. **Different buffer depths**

The impact of buffer depth on the latency and the throughput is measured using two buffer depths: 4-flits and 8-flits. The network of wormhole routers with a 3-pipeline stage running at 100MHz over synthetic traffic is used with injection flits rates on multiple network sizes, and then the results are compared to 2-flits buffer depth.

The latency is calculated with 4-flits and 8-flits buffer over the random synthetic traffic pattern as shown in Figure 4-11, and Figure 4-12 respectively. Similarly, the throughput of the two buffer sizes is demonstrated in Figure 4-13 and Figure 4-14 respectively. The correlation between the buffer depth and the network's latency and throughput is revealed, as the wormhole router has three pipeline stages; thus, the routers with at least 5-flits per buffer have the same zero-load network latency. Consequently, the router with a buffer depth of 2-flits achieves the worst network performance.

On the other hand, In NoC-DPR simulator, the NI increases the number of buffers in the network, which results in handling more round-trip flow control signalling. Consequently, increasing the buffer depth from 2-flits to 4-flits and 8-flits improves the network latency by a factor of 1.6x and 2.7x as demonstrated in Figure 4-11, and Figure 4-12 respectively. Also, increasing the buffer depth enhances the throughput at higher FIR by 50.9% and 28.8%, as portrayed in Figure 4-13 and Figure 4-14 respectively. The summary of the results of latency and throughput for different network sizes using 2, 4 and 8-filt buffer depth is listed in Table 4-1.

Table 4-1: Latency and Throughput for different network sizes using 2 and 8-filt buffer depth

	2-flit buffer size		**4-flit buffer size**		**8-flit buffer size**	
Net size	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**
4x4	7.5	0.13	4	0.25	2	0.45
8x8	14	0.06	7.9	0.14	4	0.25
16x16	22	0.03	13.5	0.07	8	0.12
18x18	23	0.02	14.7	0.06	8.7	0.11

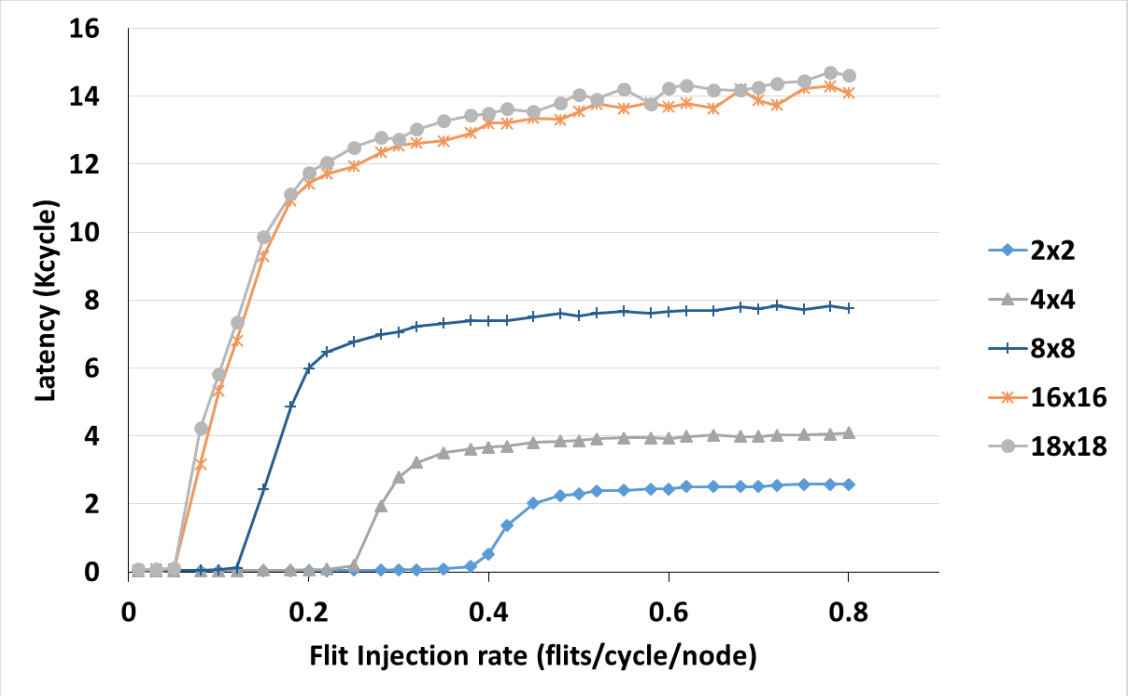


Figure 4-11: Average latency for 4-flits buffer depth

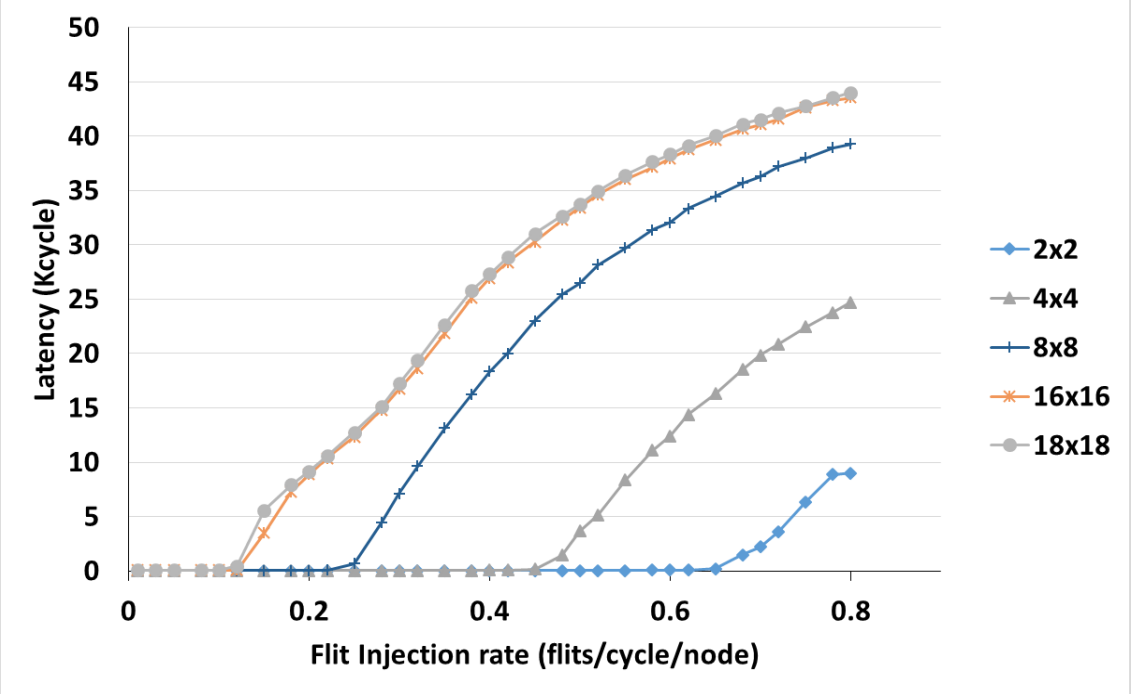


Figure 4-12: Average latency for 8-flits buffer depth

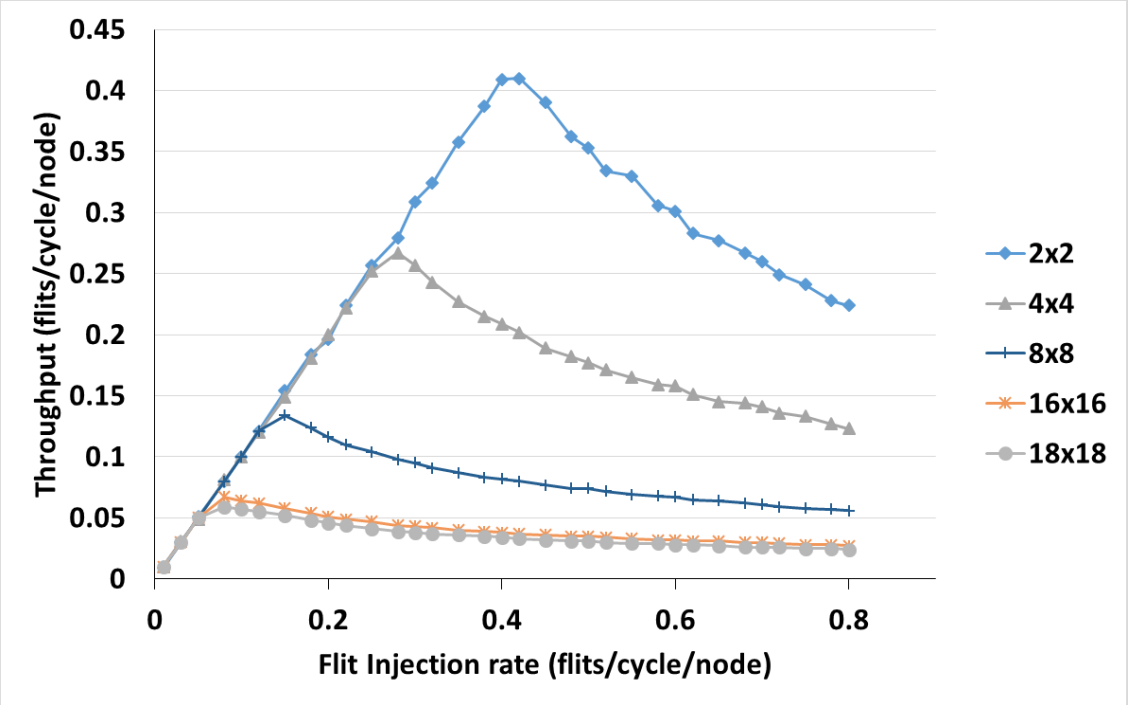


Figure 4-13: Average throughput for 4-flits buffer depth

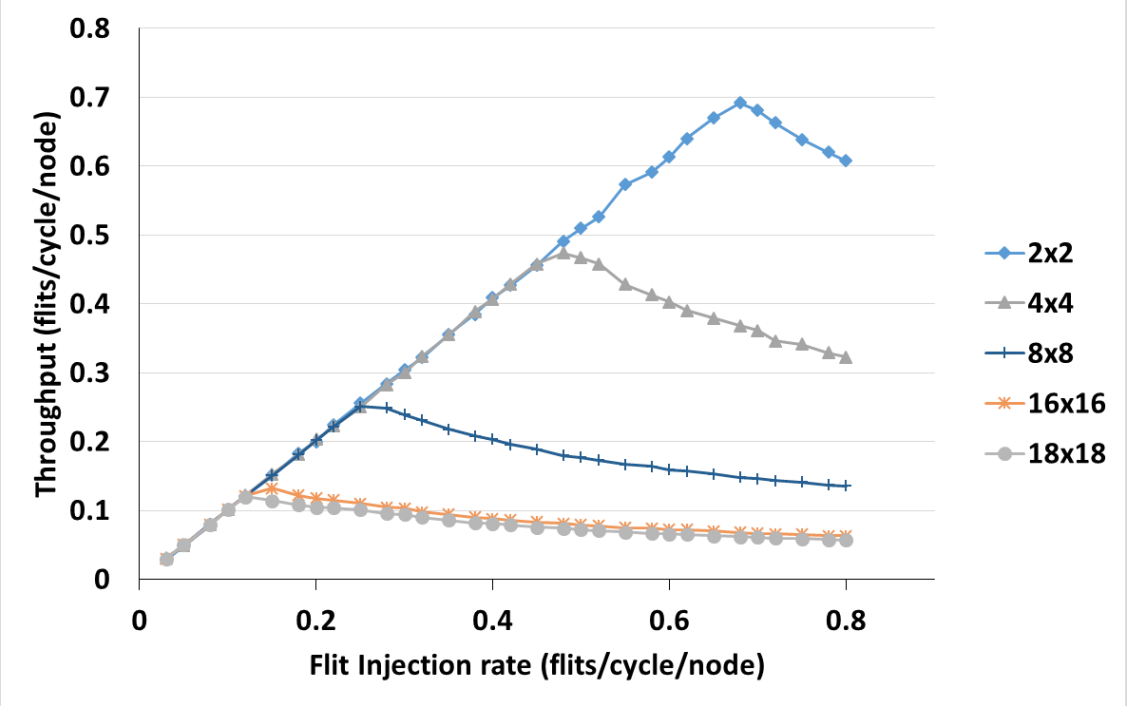


Figure 4-14: Average throughput for 8-flits buffer depth

4.7. **Different router types**

NoC-DPR simulator supports a wormhole router with virtual channels along with the pipeline wormhole router. The simulation using two networks of wormhole routers with 4-virtual channels and 8-virtual channels are considered to study the impact of changing the router type on network's latency and throughput.

As shown in Figures 4-15, 4-16, 4-17 and 4-18, the saturation values of latency are improved by factors of 3.6x and 4x for 4-virtual channels routers and 8-virtual channel routers respectively. Latency saturated at 6,000 cycles and 5,500 cycles as depicted in Figure 4-15 and Figure 4-16. In contrast, the saturated value "22,000 cycles" of the network of wormhole routers without virtual channels for the network size of 18x18 as shown in Figure 4-8.

This improvement is because of the multiplexing of the same physical link across multiple flits from the different packets; thus routers with VCs prevent deadlocks.

Interestingly, latency and throughput improve with factors more significant than buffer depth factors. It is also noticed that the throughput is saturated; which is expected as long as the network is not congested, and the flow control of NI is disabled.

Therefore, PEs inject more flits, and the network is loaded with maximum accepted flits at higher FIR as shown in Figure 4-17 and Figure 4-18. Consequently, the increasing percentage is about 60.1% and 61% for throughput in 4-VCs and 8-VCs over wormhole router with no VCs. Shown in Figure 4-10. The summary of results of latency and throughput for a different network using 4 and 8-VC wormhole router is listed in Table 4-2.

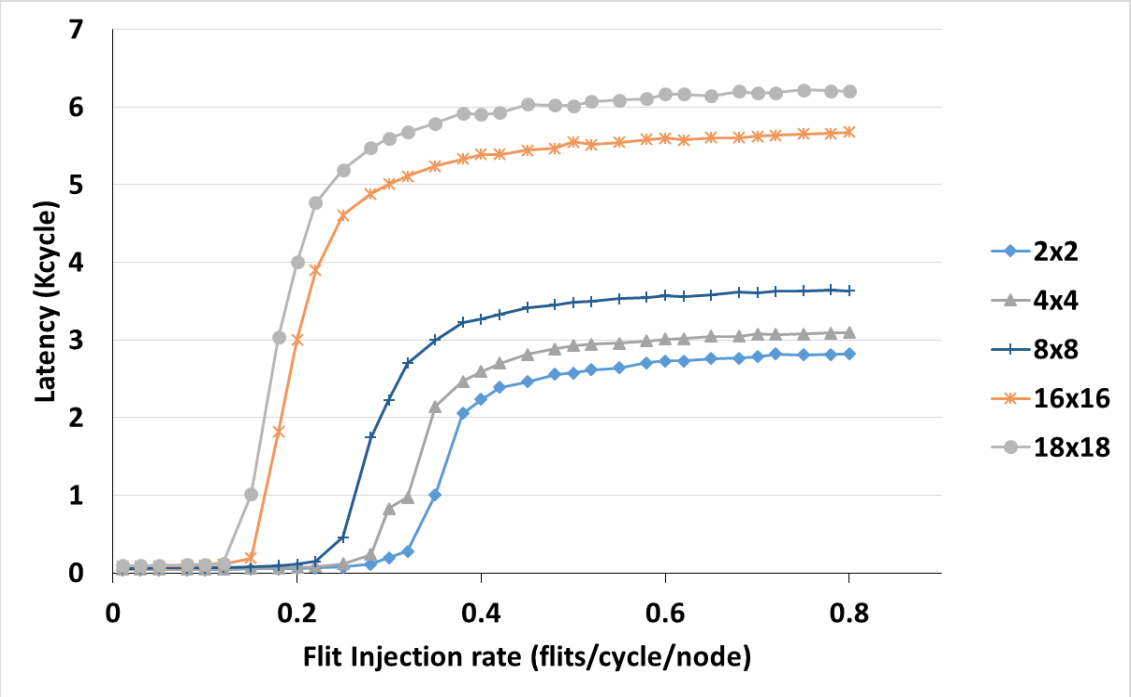


Figure 4-15: 2-flits buffer depth average latency for 4-virtual channel

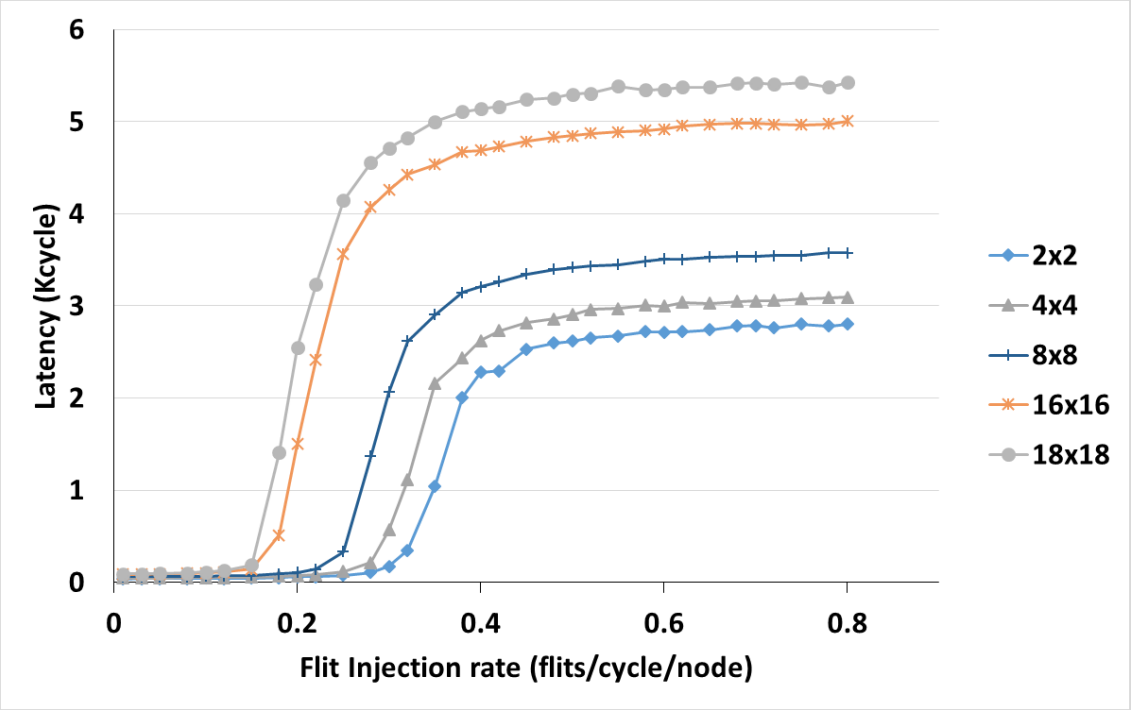


Figure 4-16: 2-flits buffer depth average latency for 8-virtual channel

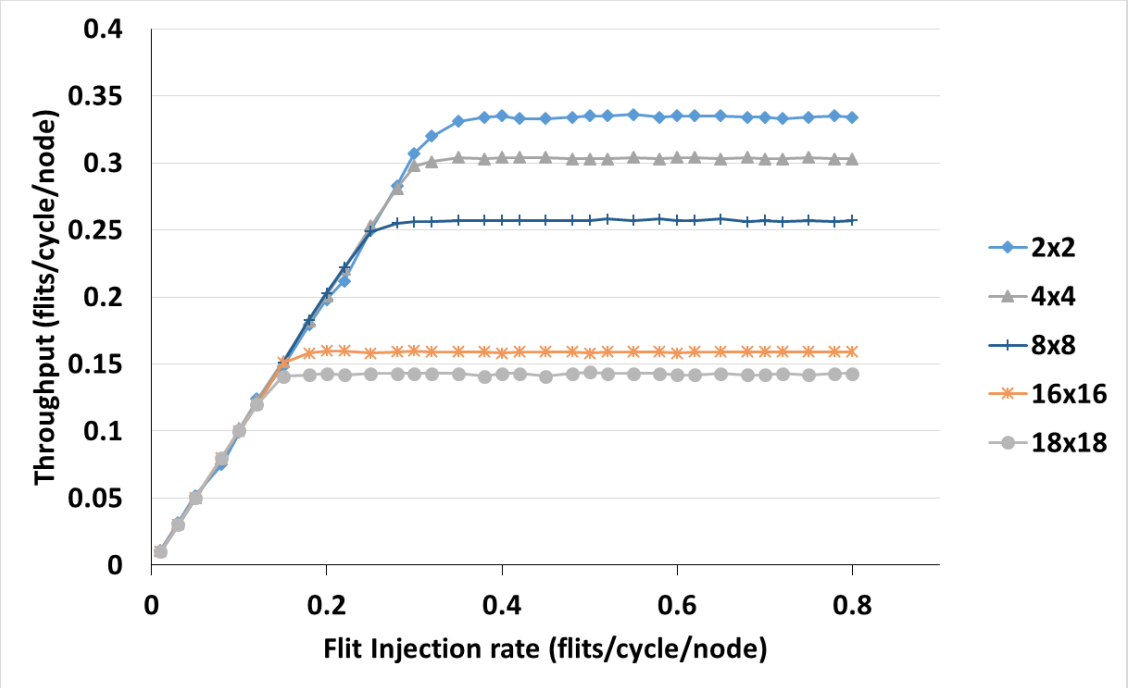


Figure 4-17: 2-flits buffer depth average throughput for a 4-virtual channel

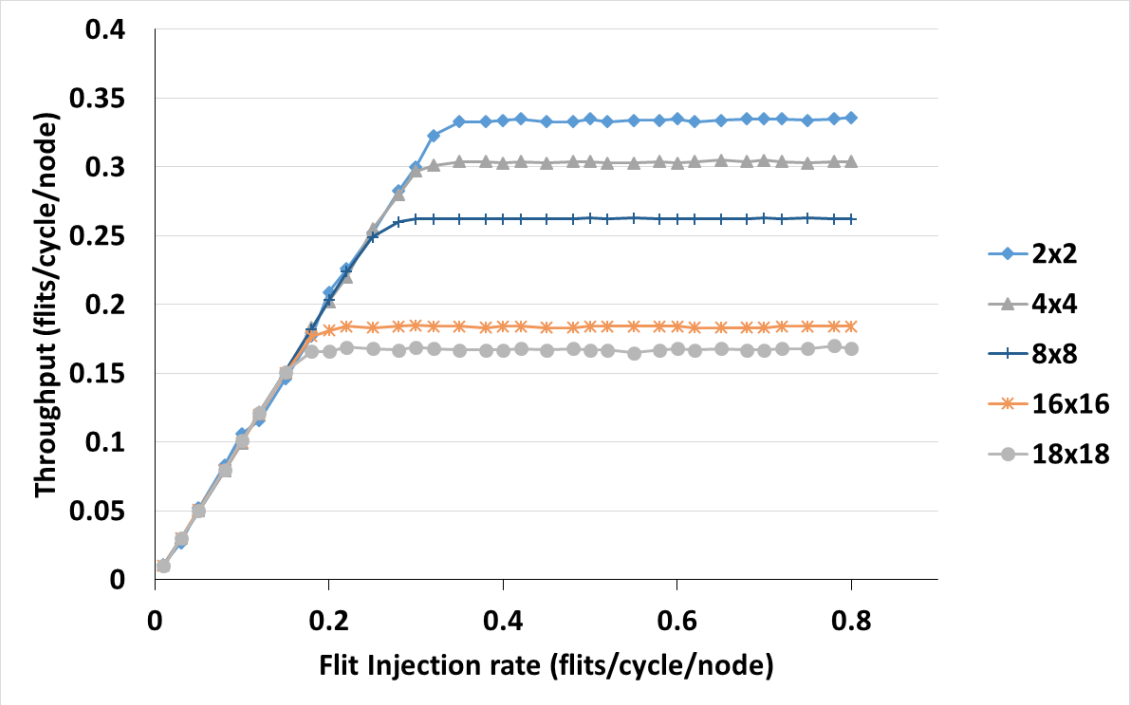


Figure 4-18: 2-flits buffer depth average throughput for an 8-virtual channel

Table 4-2: Latency and Throughput for different network sizes using 4 and 8-VC wormhole router

	2-flit buffer size With 4-VC		2-flit buffer size With 8-VC	
Network size	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**
4x4	3	0.3	3	0.3
8x8	3.5	0.25	3.5	0.26
16x16	5.5	0.15	5	0.18
18x18	6	0.14	5.2	0.17

Table 4-3: Latency and Throughput for different network sizes using 2-flits buffer depth and 8-flits buffer depth with 4-VC wormhole router

	2-flit buffer size		8-flit buffer size with 4-VC	
Network size	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**	**Latency (Kcycles)**	**Throughput (flits/cycle/node)**
4x4	7.5	0.13	1.5	0.58
8x8	14	0.06	2.7	0.36
16x16	22	0.03	4.8	0.2
18x18	23	0.02	5.2	0.18

4.8. Summary

The architecture of the proposed NoC-DPR simulator is discussed in this chapter. Then, the performance of the network regarding throughput and latency is evaluated after modifying the NoCtweak simulator and merging it with the Rechannel Library. That is done by inserting an explicit network interface between the PE and the router. It is shown that the latency is improved due to this change, while the throughput is degraded. This degradation is because of adding NI of NoC-DPR simulator, which is responsible for controlling packet generation from PE according to NoC state. Therefore, NI sends and receives control signals to the PE so that it prevents network congestion and deadlocks.

The variation of buffer depth and router type has an effect also on the network's throughput and latency. It is shown that buffer depth of 4-flits and 8-flits enhances the latency with factor 1.6x and 2.7x respectively. Also, increasing the buffer depth improves the throughput at higher FIR by 50.9% and 28.8% over a network of 2-flits buffer.

While the latency is improved by factors of 3.6x and 4x using the 4-virtual channels routers and 8-virtual channel routers respectively, the increasing percentage is about 60.1% and 61% for throughput in 4-VCs and 8-VCs over wormhole router with no VCs.

Chapter 5 : Simulation Results and Discussion

5.1. Introduction

The test experiment in this chapter aims to simulate the DPR on NoC-based FPGA using of parallel DPRs on plenty of network sizes. Moreover, a comparison is held concerning the Reconfiguration Time (RT). Two separate applications are used to estimate the reconfiguration time: the first one is random (synthetic), and the second is an embedded application.

5.2. Dynamic partial reconfiguration simulation setup

Virtex-5 xc5vfx100t FPGA is used as the primary configuration plane. It is divided into PR regions according to the network size. Consequently, for network size 2x2, as illustrated in Figure 5-1, all configuration plane of Virtex FPGA is divided into 4 reconfiguration regions. Nevertheless, the bitstream sizes of each configuration region are calculated using the Xilinx ISE v14.7 tool. Finally, RT is determined by using partial reconfiguration cost calculator [42].

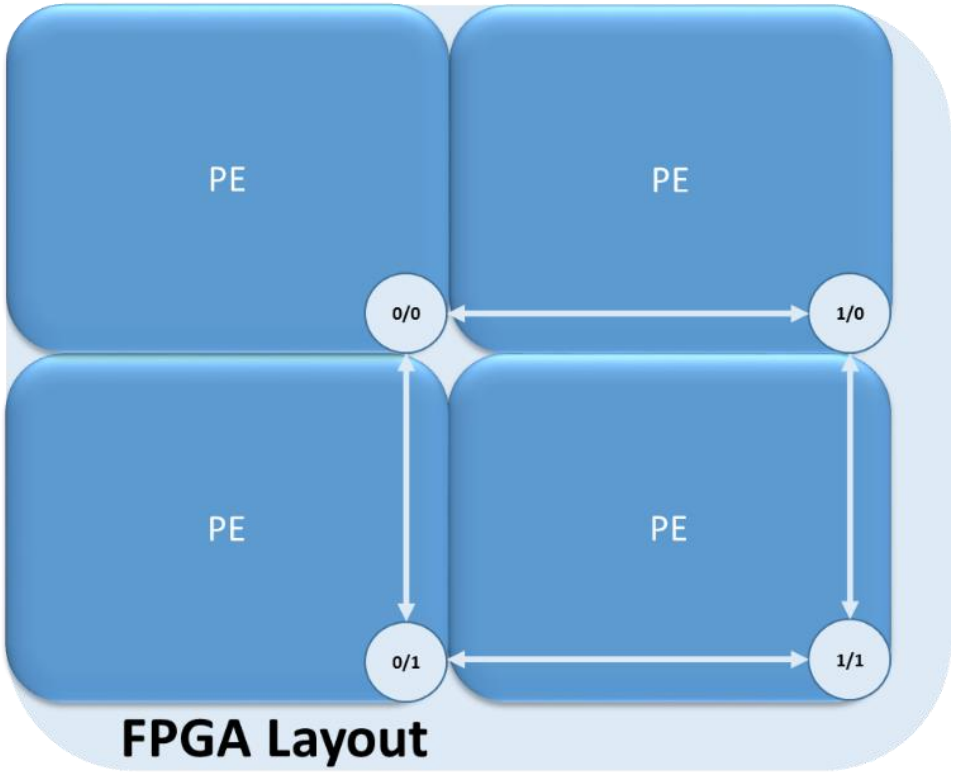


Figure 5-1: Process Elements as reconfiguration region for 2x2 NoC

Each PE of NoC is assumed as PRs, as depicted in Figure 5-1 and Figure 5-2 for network size of 2x2 and 4x4 respectively. Therefore, the network resources such as the

router, NI, and wires are considered hardwired IPs on an FPGA chip layout to simplify the RT estimation. The primary advantage of using NoC-based FPGA instead of the conventional SRAM-based FPGA is that multiple simultaneous DPRs are achieved. Accordingly, each PE has a reconfiguration control unit such as ICAP which is assumed as hardwired IP.

The reconfiguration time is determined by the cost calculator using the partial bitstream size for PR of each PE in the different network size as listed in Table 5-1. However, the reconfiguration time of DPR is not related directly to design resources. Nevertheless, RT is proportional to the PR region selection, which is translated to a significant number of frames [1].

As given in (1) and (2) [47], RT_{SM-PPC} is the time from the storage means (i.e., compact flash or RAM) to the memory of a local processor, $RT_{PPC-ICAP}$ is the time from local processor memory to ICAP memory, $RT_{ICAP-CM}$ is the time from ICAP memory to the configuration memory, f_s is the frame size, and n is the number of frames per bitstream file.

The full derivation from (1) to (2) is performed by analysis each phase of reconfiguration and the corresponding throughput and theoretical bandwidth. RT values are used along with the NoC-DPR simulator for this test experiment.

$$RT = RT_{SM-PPC} + RT_{PPC-ICAP} + RT_{ICAP-CM} \quad (1)$$

$$RT = f_s(n+1) \times 3.66 \times 10^{-3}\, ms \quad (2)$$

Table 5-1: Reconfiguration time for different NoC sizes

Network size	Bit file size (Kbytes)	Reconfiguration time for each node (msec)
1x1	2526	570.34
2x1	1263	285.188
2x2	632.5	142.612
3x3	280	63.402
4x4	158	35.68
5x5	101	22.848
6x6	70	15.878
7x7	51.5	11.675
8x8	39.4	9.069
9x9	31.2	7.08
10x10	25.25	5.818
11x11	20.3	4.814

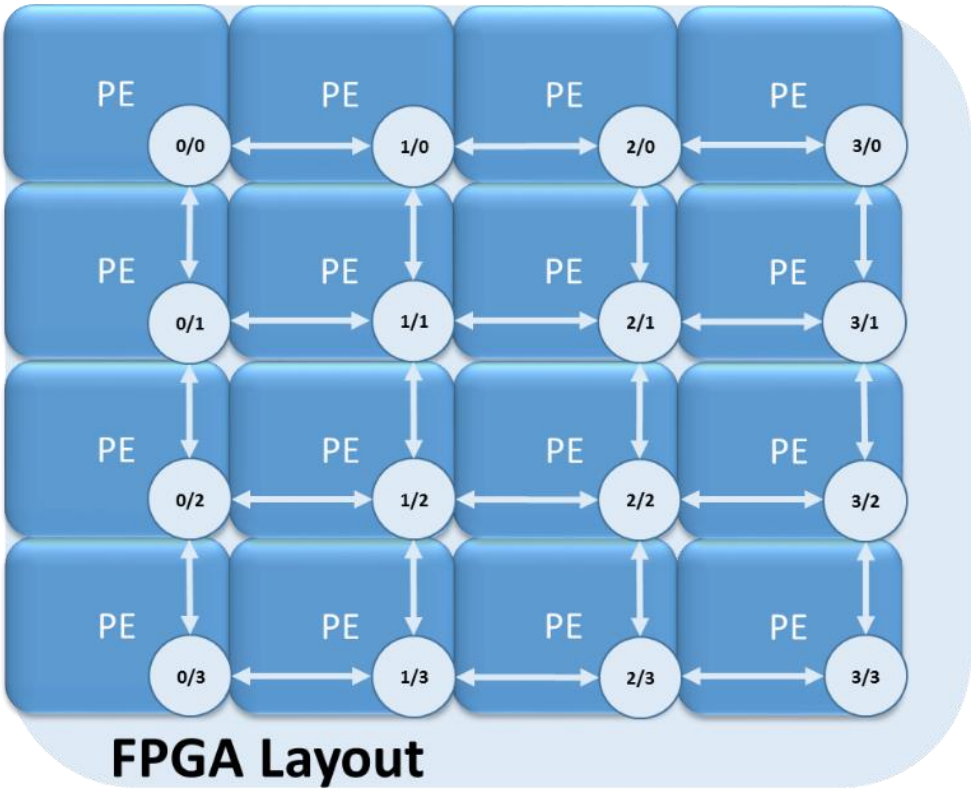


Figure 5-2: Process Elements as reconfiguration region for 4x4 NoC

5.3. Simulation results of the synthetic application

In the following section, the suitable network size and the number of suitable parallel DPRs are investigated for two types of wormhole router (i.e., the wormhole router with no VCs and with VCs). Further tests are performed to study the effect of network performance variations on RT.

Theoretical RT for the 1-DPR process is calculated using (1) and (2) as shown in Table 5-1. Nevertheless, the theoretical RT for simultaneous 3-DPR is evaluated by dividing the RT of 1-DPR by 3. The criteria for selecting (RT / No. of DPR) metric to compare the different number of simultaneous DPRs are adapted to have a fair comparison between conventional SRAM-based and NoC-based FPGAs.

5.3.1. Wormhole router without virtual channels

5.3.1.1. Wormhole router with 2-flits buffer depth

A comparison between the different numbers of simultaneous DPRs on the NoC is performed. It is held based on the difference between the theoretical and the simulated RT for each DPR number, as illustrated in Figure 5-4, Figure 5-5, and Figure 5-6 for one, three, and five simultaneous reconfigurations respectively. This metric is a vital indicator for the performance variation because each network size has different values of throughput and latency that affect the RT. However, a wormhole router with buffer size

2-flits and FIR of 0.1 flits/cycle running at 100MHz over synthetic traffic is used for this test experiment .

In network sizes less than 10x10 and RT above 1 msec, the difference is unnoticeable between theoretical and simulated RT. For instance, in Figure 5-4, a slight increase is noticed in the difference in the network size less than 7x7, the difference does not increase over 30% as illustrated in Figure 5-3, while for the network size larger than 8x8 it reaches over 70%. Alternatively, the simulated RT for five simultaneous reconfigurations, as shown in Figure 5-6, the difference is drifted at network sizes less than 10x10. Then, the gap stroked at larger network sizes that are larger than 10x10, where the difference percentage is higher than 100% and reaches 400% at network size 18x18 as illustrated in Figure 5-3.

Figure 5-3 shows the percentage difference between the theoretical and the simulated RT using one, three and five simultaneous DPRs. It is shown that at small network sizes, from 2x2 to 9x9, the difference is always lower than 40%, 50% and 100% respectively, because RT is relatively more significant than any network overheads, such as latency and throughput overheads.

On the other hand, starting from network size of 10x10, the latency is considered and affects the RT significantly.

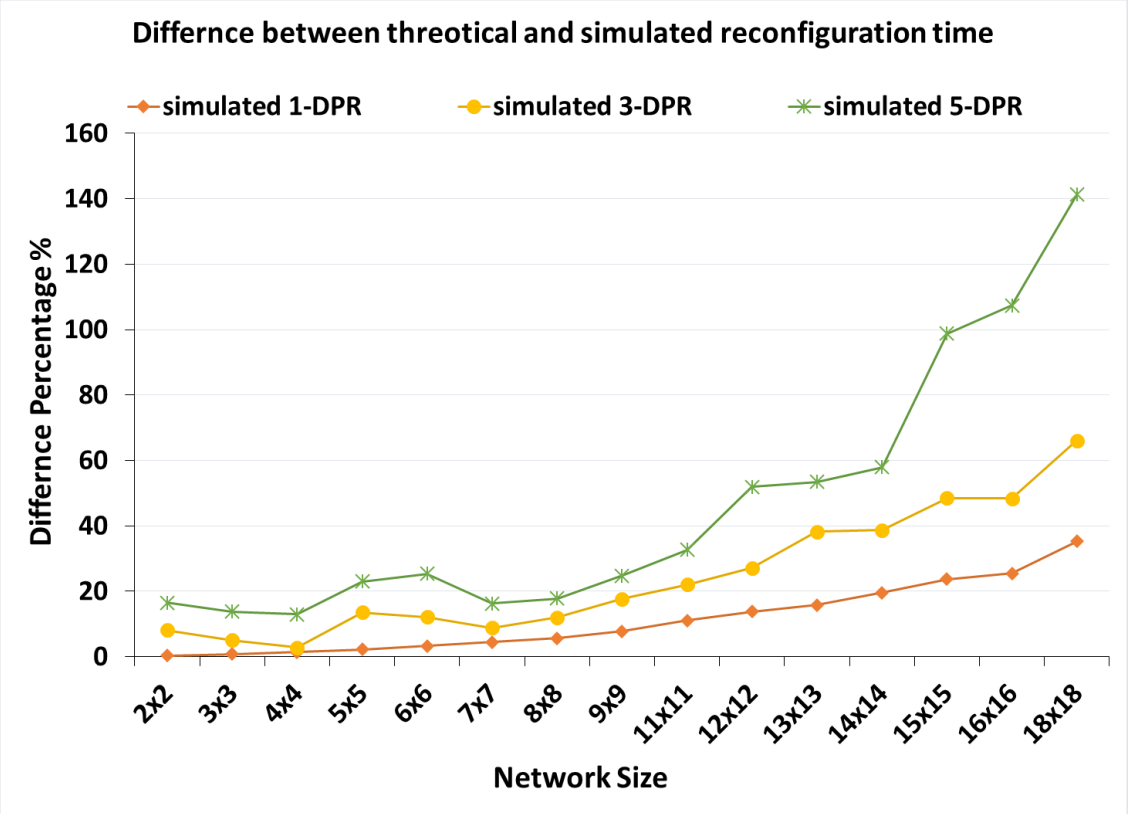


Figure 5-3: Difference percentage between the theoretical and the simulated 1, 3 and 5-DPR using wormhole router with 2-flits buffer depth

As the number of simultaneous DPRs increases, the difference increases markedly. The difference percentage does not exceed 70% for all network sizes using 1-DPR as shown in Figure 5-3. Whereas, in simultaneous 3-DPRs, the difference percentage reaches up to 75% at large network sizes of 18x18 as shown in Figure 5-5. This is due to the complexity of controlling parallel DPRs, and the large network's latency due to small buffer depth 2-filts. Furthermore, other nodes are unable to communicate with the node that is reconfigured dynamically.

The experiment's DPR results using a wormhole router with buffer size 2-flits and FIR of 0.1 flits/cycle running at 100MHz over synthetic traffic are summarized in Table 5-2, where the recommended network size and the number of simultaneously DPRs are listed according to the reconfiguration time.

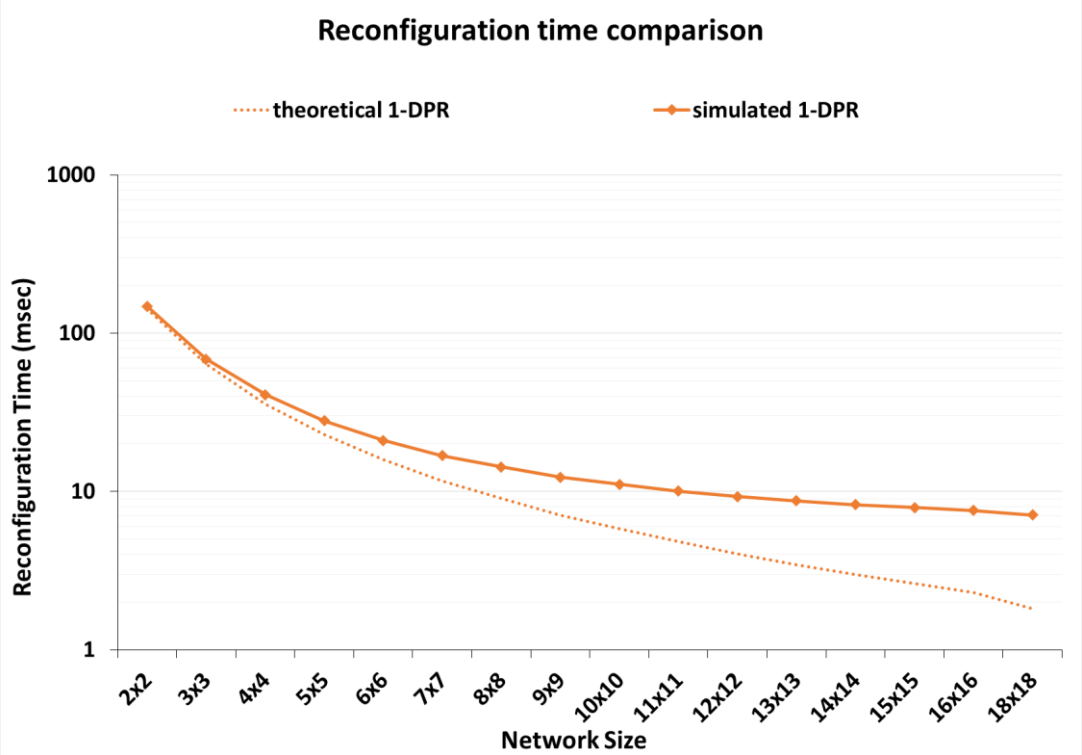


Figure 5-4: RT Comparison between the theoretical and the simulated 1-DPR using wormhole router with 2-flits buffer depth

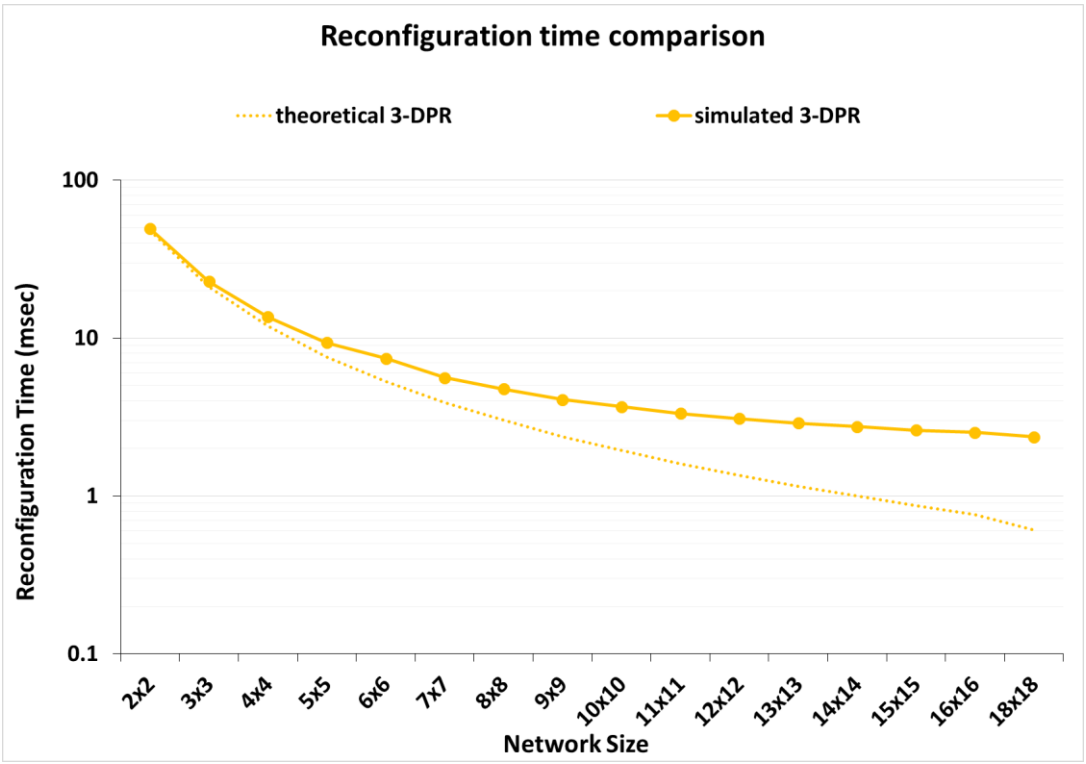


Figure 5-5: RT Comparison between the theoretical and the simulated 3-DPR using wormhole router with 2-flits buffer depth

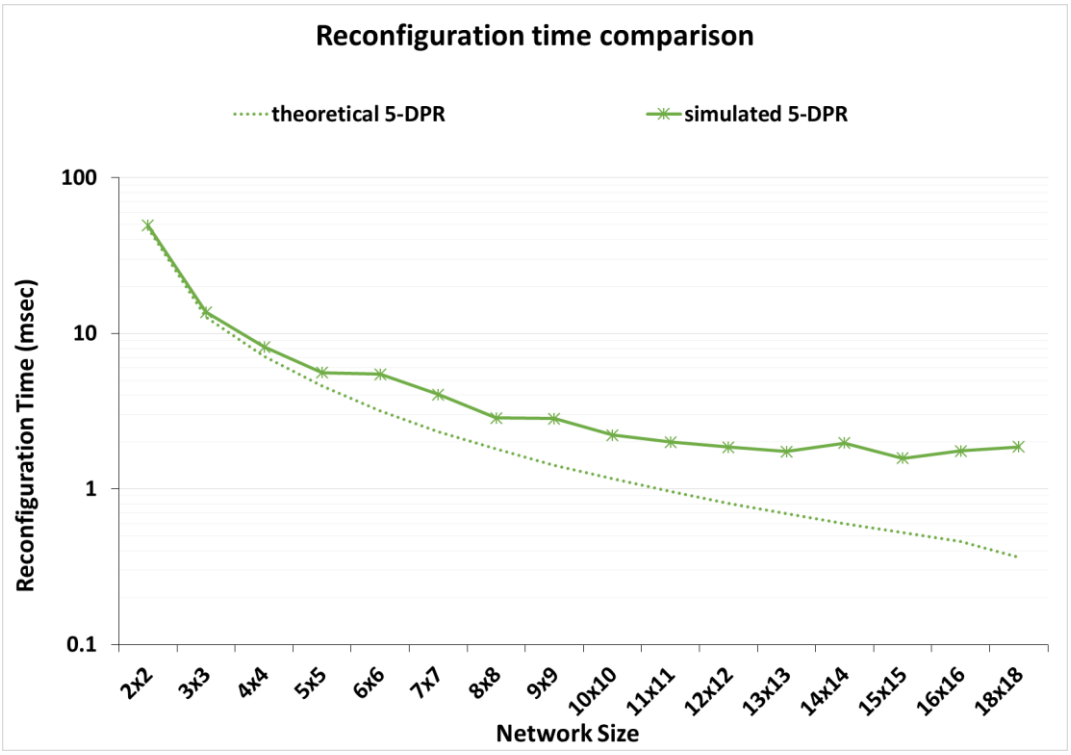


Figure 5-6: RT Comparison between the theoretical and the simulated 5-DPR using wormhole router with 2-flits buffer depth

Table 5-2: Network size recommendation for selected reconfiguration time for wormhole router with 2-flits buffer depth

Reconfiguration Time	No of DPRs can be used	Network size
~10 msec	3 to 5 DPRs	3x3 to 5x5
~5 msec	5 DPRs	5x5 to 6x6
~1 mesc	Not recommended	Not recommended

5.3.1.2. Wormhole router with 8-flits buffer depth

Figure 5-7 shows the percentage difference between the theoretical and the simulated RT using one, three, and five simultaneous DPRs respectively. Using a wormhole router with buffer size 8-flits and FIR of 0.1 flits/cycle running at 100MHz over synthetic traffic. It is shown that at network sizes starting from 2x2 to 9x9, the difference is always lower than 7.5%, 10% and 15% respectively because RT is relatively more significant than latency and throughput overheads. On the other hand, starting from network size of 10x10, the latency is considered and affects the RT.

The difference is unnoticeable between the theoretical and the simulated RT in network sizes that are less significant than 10x10. For instance, in 1-reconfiguration Figure 5-8, a little increase is noticed in the difference in network dimension smaller than 11x11, it does not increase over 10% Figure 5-7, while for network size larger than 12x12 it reaches over 35%. Alternatively, the simulated RT for three simultaneous reconfigurations, as depicted in Figure 5-9, the difference is drifted at network sizes less than 10x10. Then, the gap stroked at larger network sizes that are larger than 10x10, where the difference percentage is higher than 10% and reaches 30% at network size 18x18 as illustrated in Figure 5-7.

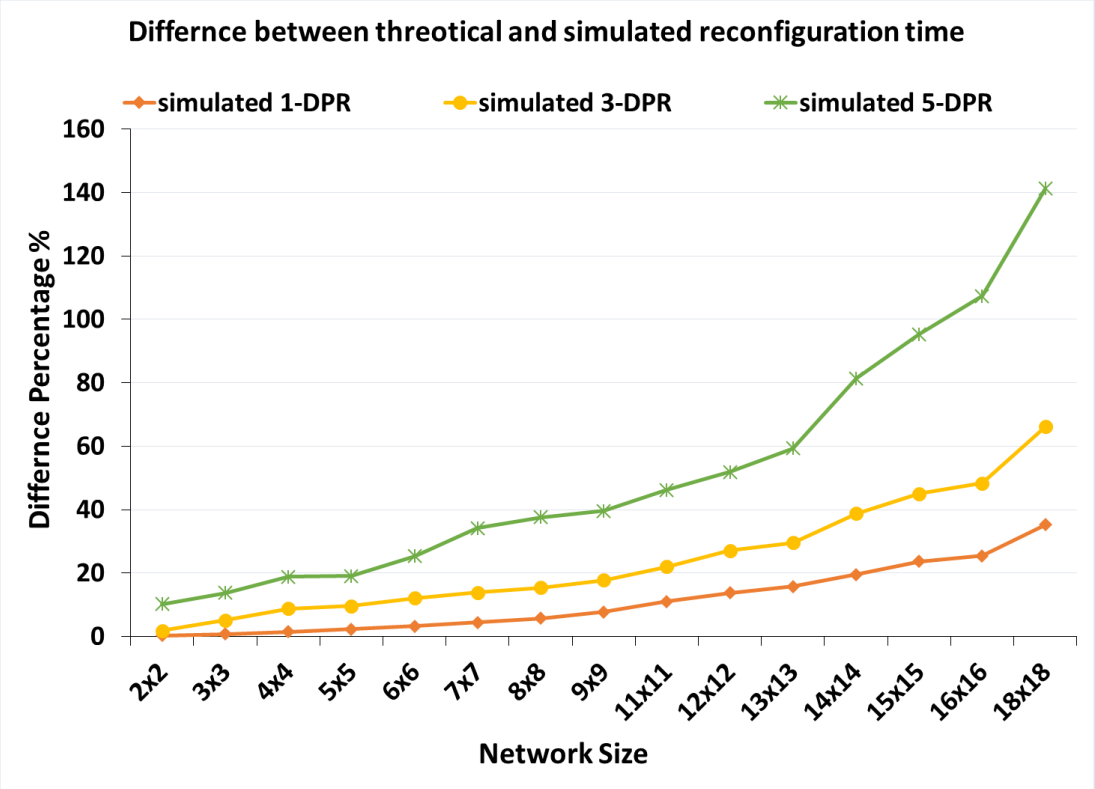


Figure 5-7: Difference percentage between the theoretical and the simulated 1, 3 and 5-DPR using wormhole router with 8-flits buffer depth

The difference percentage between the theoretical and simulated RT increases as the number of simultaneous DPRs increases. The difference percentage does not exceed 35% for all network sizes using 1-DPR as shown in Figure 5-7. Whereas, when using simultaneous 5-DPRs as shown in Figure 5-10, the difference percentage reaches up to 70% at large network sizes of 18x18 as shown in Figure 5-7. That is due to the complexity of controlling parallel DPRs, and the vast network's latency. Furthermore, all nodes are not able to communicate with the node that is under dynamic reconfiguration.

Table 5-3 summarizes the experiment's results of DPR using a wormhole router with buffer size 8-flits and FIR of 0.1 flits/cycle running at 100MHz over synthetic traffic. The recommended network size and the number of simultaneously DPRs are calculated according to the RT. The recommendation is made based on the difference between the theoretical and the simulated results of RT.

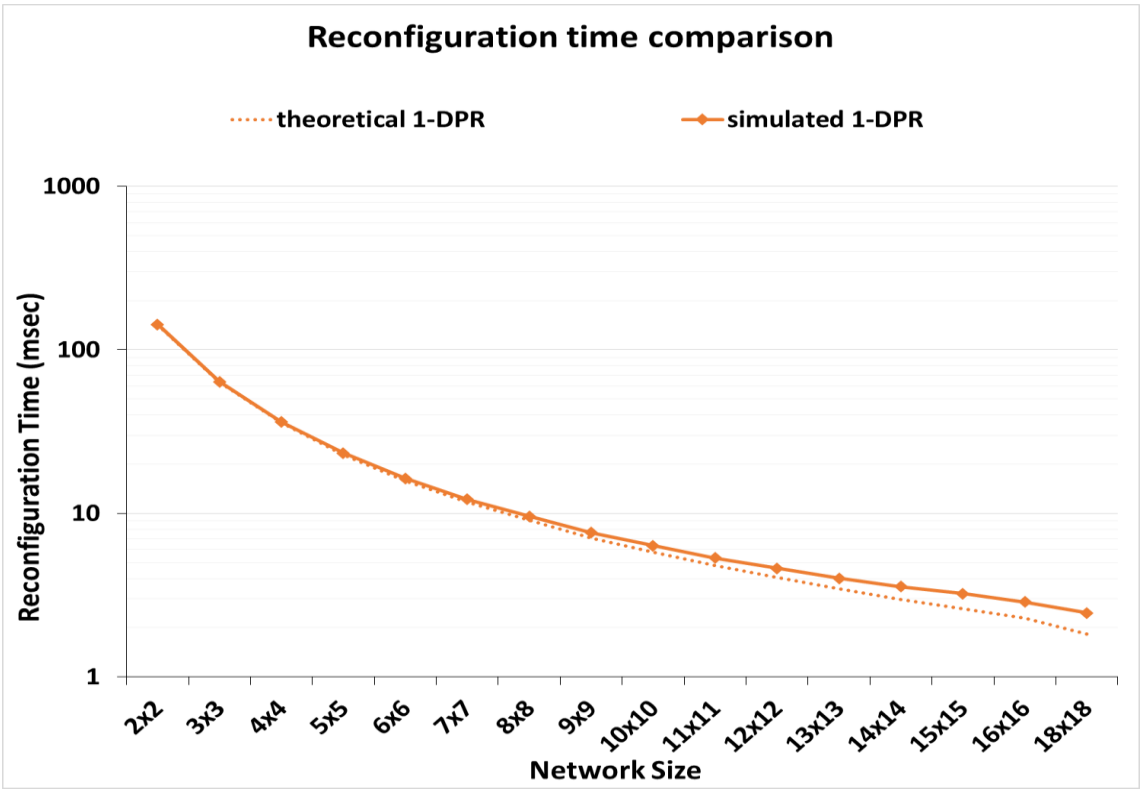


Figure 5-8: RT Comparison between the theoretical and the simulated 1-DPR using wormhole router with 8-flits buffer depth

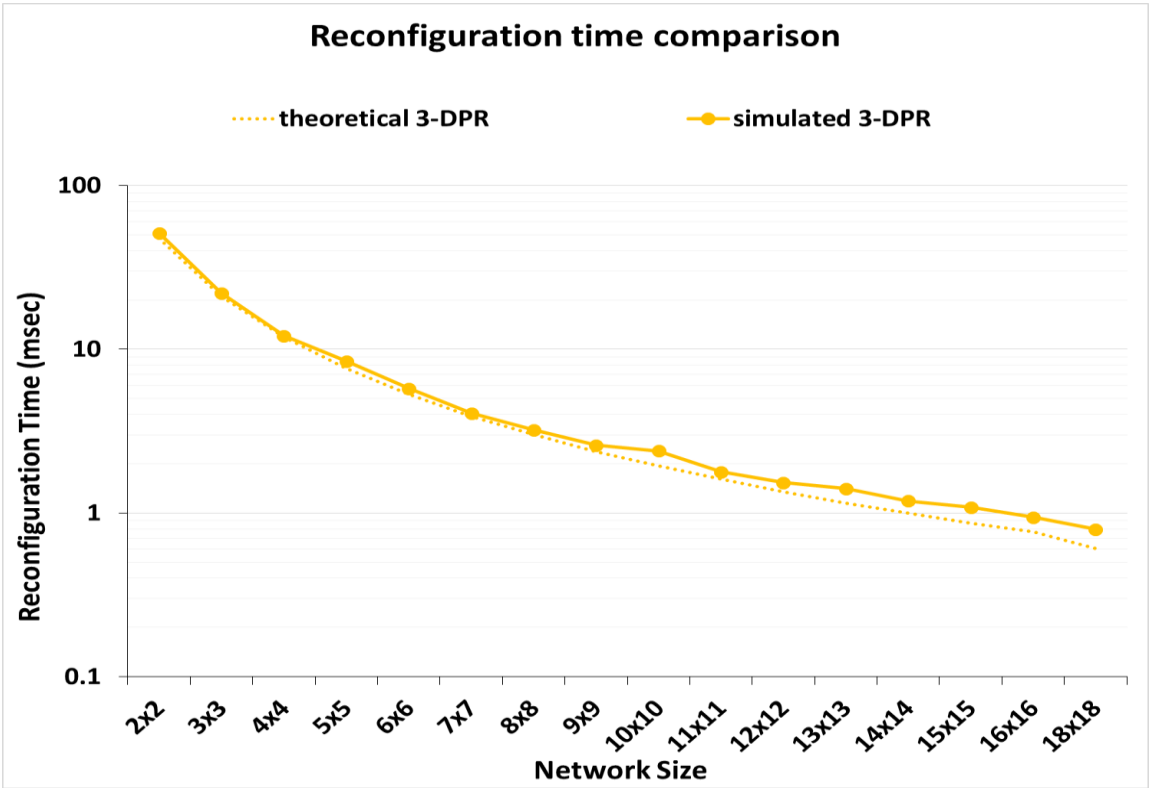


Figure 5-9: RT Comparison between the theoretical and the simulated 3-DPR using wormhole router with 8-flits buffer depth

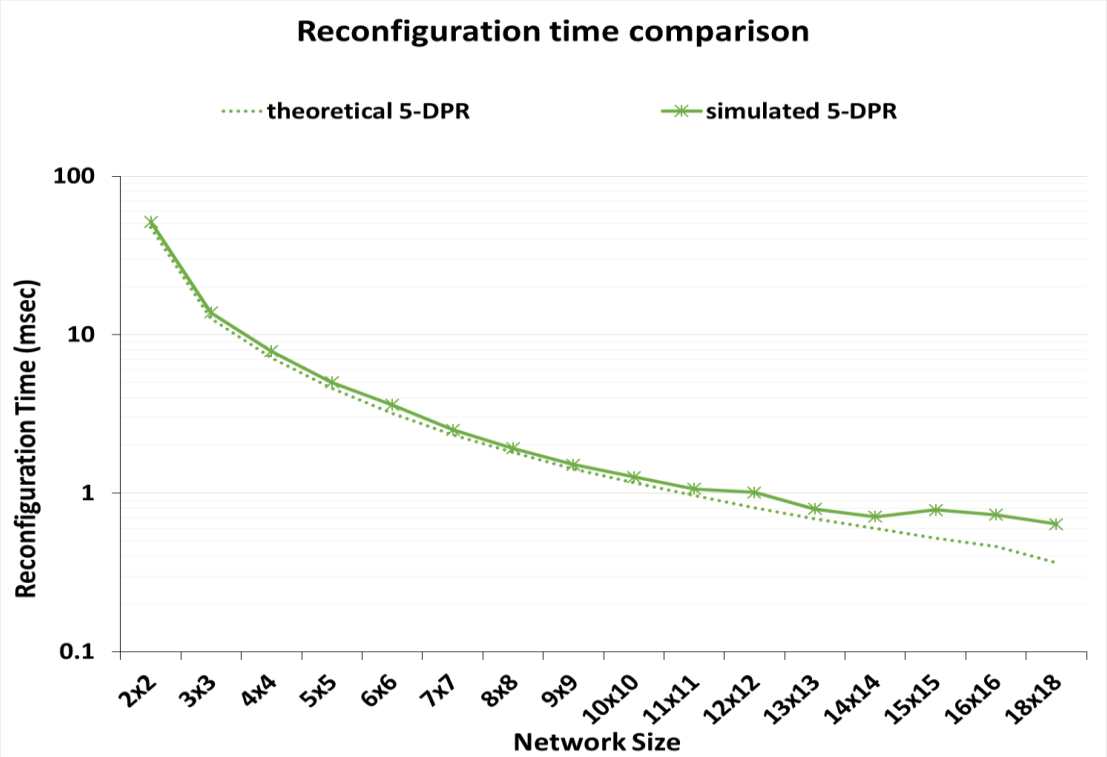


Figure 5-10: RT Comparison between the theoretical and the simulated 5-DPR using wormhole router with 8-flits buffer depth

Table 5-3: Network size recommendation for selected reconfiguration time for wormhole router with 8-flits buffer depth

Reconfiguration Time	No of DPRs can be used	Network size
~10 msec	1 to 5 DPRs	3x3 to 8x8
~5 msec	1 to 5 DPRs	5x5 to 12x12
~1 mesc	3 DPRs	10x10 to 18x18

5.3.2. Wormhole router with virtual channels

The previous experiment is repeated using the network of wormhole router with the 8-virtual channel and buffer size of 8-flits at FIR of 0.1 flits/cycle.

As illustrated in Figure 5-11, at network dimension less than 10x0, which has a significant RT (more than 5 msec), using a wormhole router with VC has a minor effect on RT. Interestingly, for higher network sizes, over 10x10, routers with VCs improve the RT and the difference between the theoretical and the simulated result is reduced by factor 60% than that a wormhole router without VC is used. These reconfiguration values correlate favourably with the network performance metrics, which also are enhanced by using wormhole router with VCs. Figure 5-12 and Figure 5-13 state the result of 1-DPR and 3-DPR at a time, where a constant difference is noticed as the network size varies from 2x2 to 18x18. In contrast, the simulated RT for 5-DPR simultaneously increases at network sizes more than 10x10 as shown in Figure 5-14.

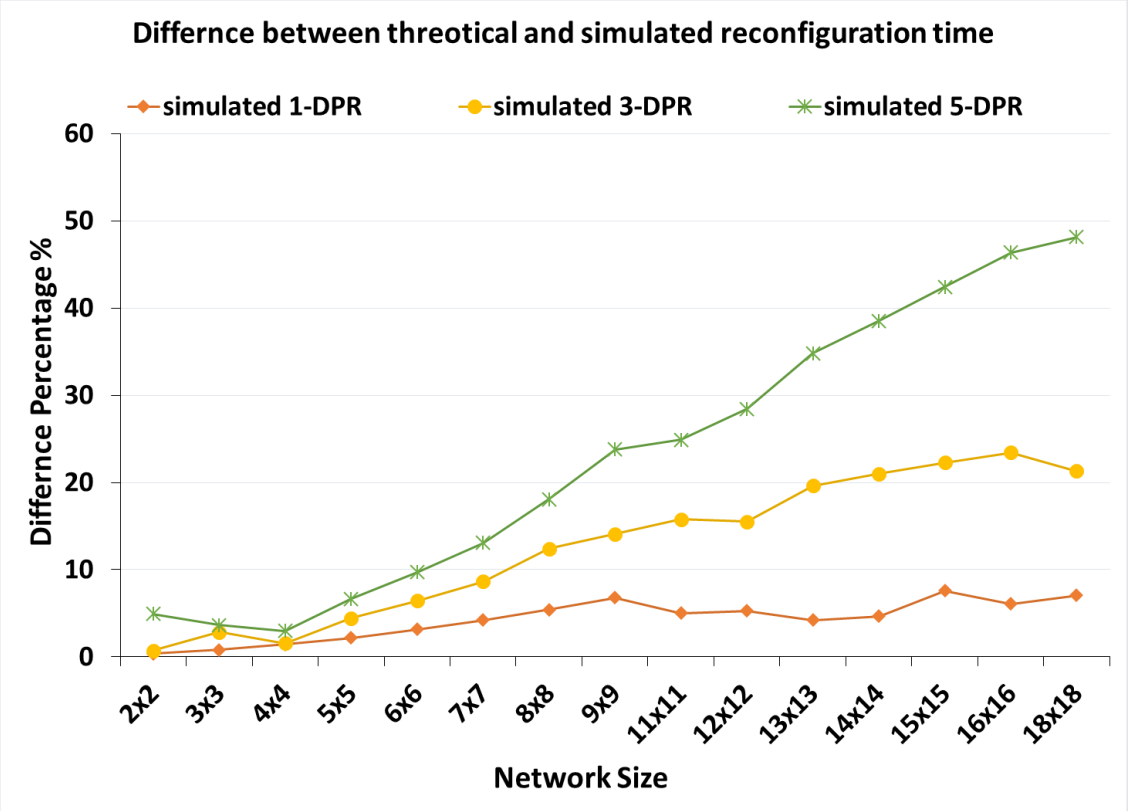


Figure 5-11: Difference percentage between the theoretical and the simulated 1, 3 and 5-DPR using wormhole router with 8-flits buffer depth and 8-VC

Figure 5-11 shows that at small network sizes (i.e., 2x2 to 9x9), the difference percentage is lower than 7%. On the other hand, starting from the network dimension of 10x10, the difference reaches up to 12%, as the RT becomes smaller and is affected by the network's latency.

At network sizes 2x2 to 9x9, the difference is lower than 10%. Starting from network size of 10x10, the difference reaches up to 30%, as the RT becomes smaller and is affected by the network's latency as shown in Figure 5-11.

In contradiction with the earlier findings using wormhole router without VCs, as the simultaneous number of DPR increases and the network size increases, the percentage difference between the theoretical and the simulated RT values does not increase significantly. The percentage does not exceed 8% all over network sizes using 1-DPR as denoted in Figure 5-12, and by using 3-DPR simultaneously, the percentage reaches up to 20% at large network size of 18x18 in Figure 5-13, which substantiates previous findings in chapter 4.

Table 5-4 summarizes the experiment's results of DPR according to reconfiguration time, network size and the number of simultaneous DPRs. The recommendation is made based on the difference between the theoretical and the simulated result of RT for both networks of wormhole router without VC and with 8-virtual channels using a synthetic application.

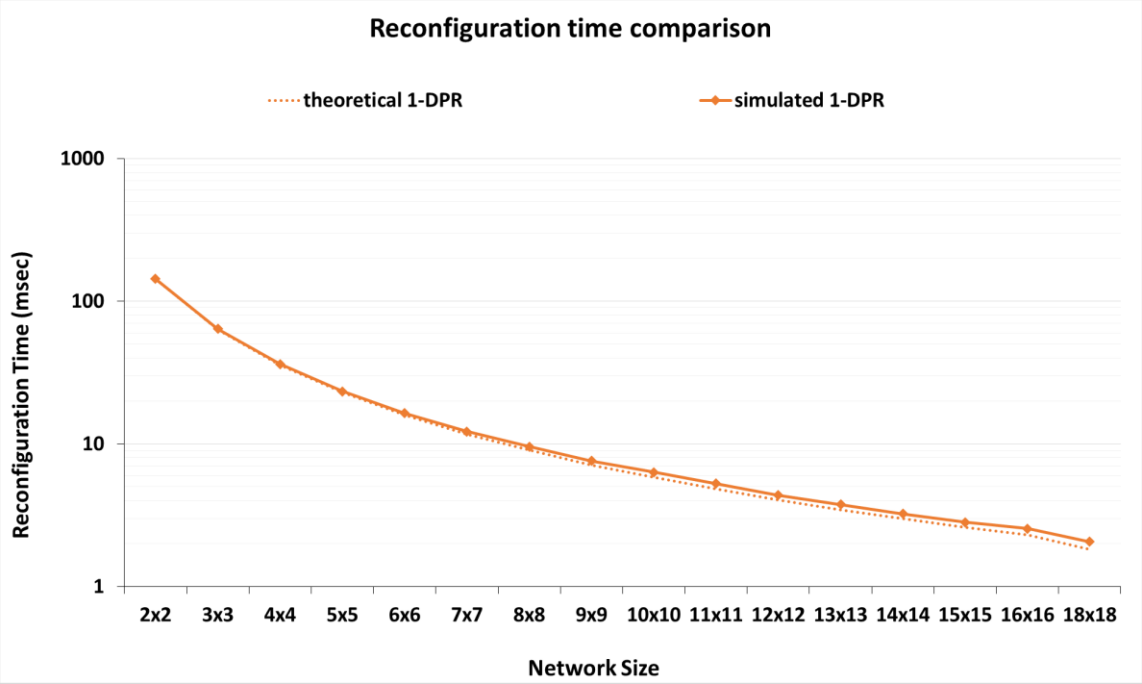


Figure 5-12: RT Comparison between the theoretical and the simulated 1-DPR using wormhole router with 8-flits buffer depth and 8-VC

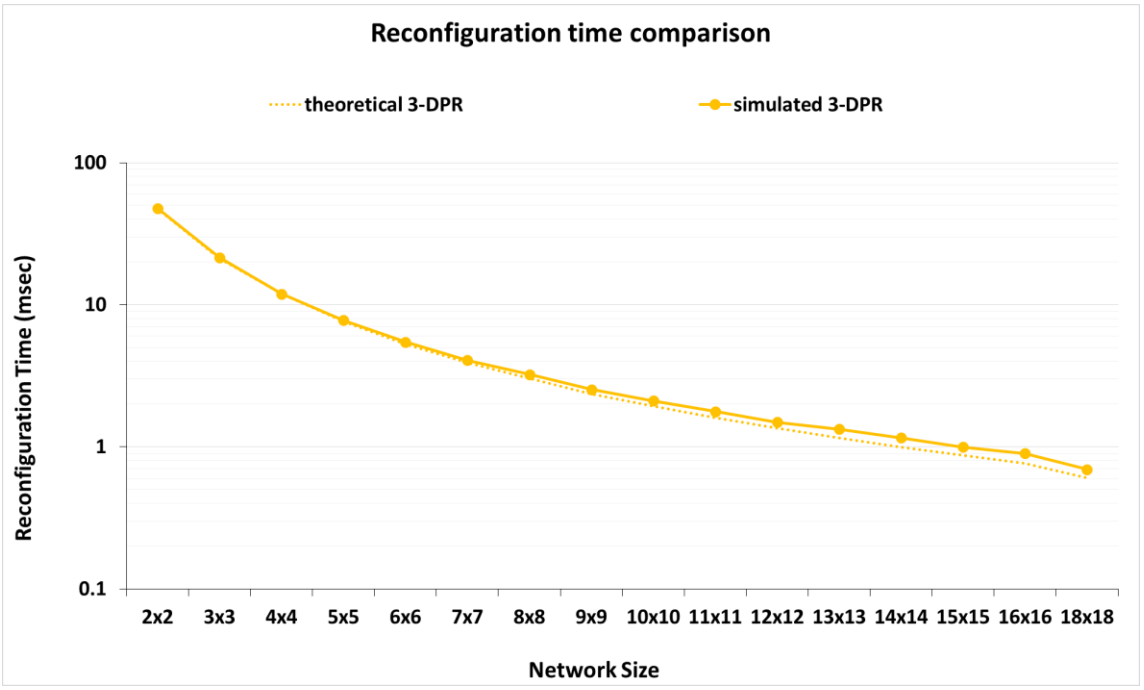


Figure 5-13: RT Comparison between the theoretical and the simulated 3-DPR using wormhole router with 8-flits buffer depth and 8-VC

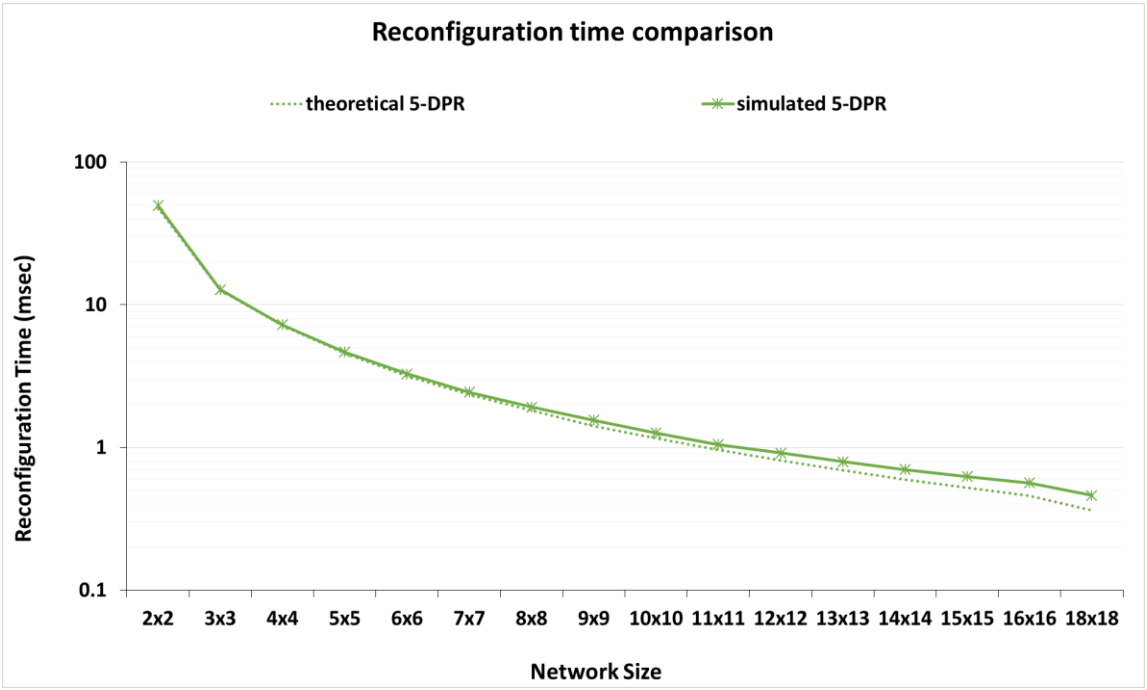


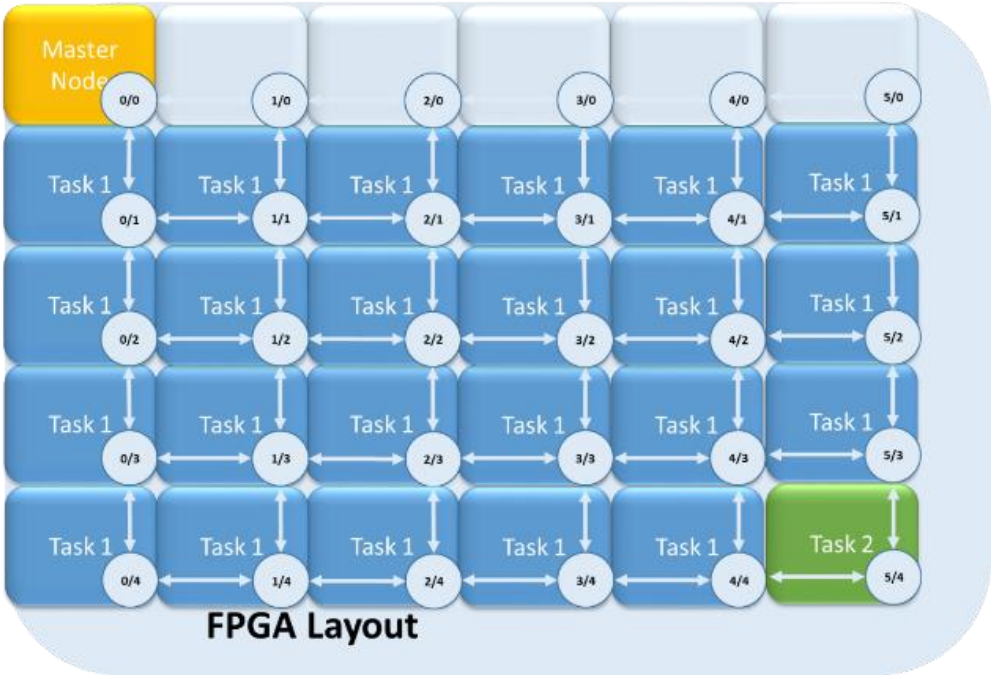
Figure 5-14: RT Comparison between the theoretical and the simulated 5-DPR using wormhole router with 8-flits buffer depth and 8-VC

Table 5-4: Network size recommendation for selected reconfiguration time for wormhole router with 8-flits buffer depth and 8-VC

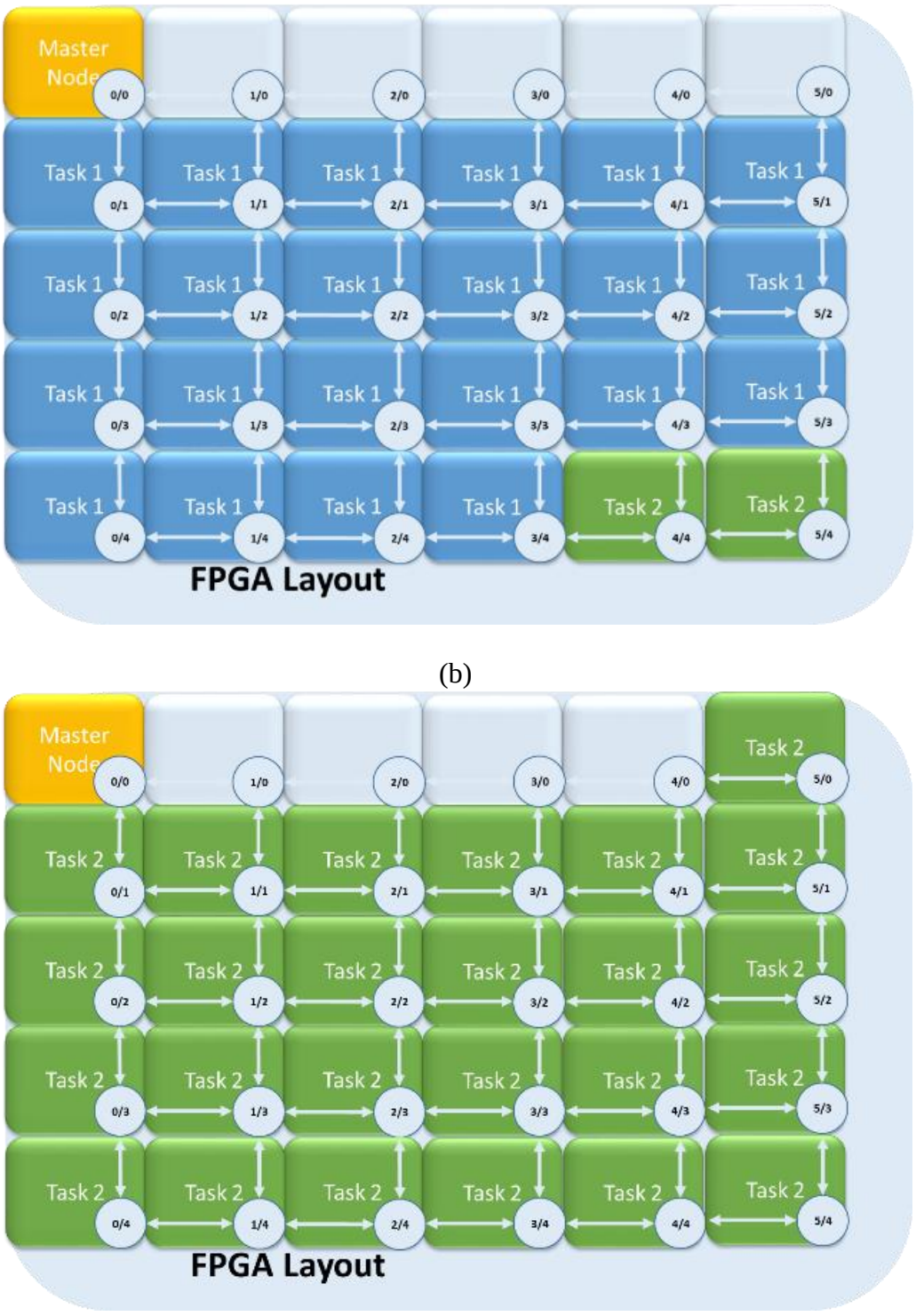
Reconfiguration Time	No of DPRs can be used	Network size
~10 msec	1 to 5 DPRs	3x3 to 8x8
~5 msec	1 to 5 DPRs	5x5 to 12x12
~1 mesc	3 to 5 DPRs	10x10 to 18x18

5.4. Simulation results of the embedded application

In this experiment, the DPR of NoC-based FPGA is studied and evaluated with an embedded application. The impact of changing the simultaneous dynamic reconfiguration, representing the switching between two applications, is assessed and compared to the theoretical results.



(a)



(b)

(c)

Figure 5-15: Network 5x6 for Wifi receiver application switching to Multi-Media application

5.4.1. Embedded Application

Many embedded applications such as 802.11a WiFi receiver [47], Video Object Plane Decoder, and multimedia system [48] are examined using NoC-DPR simulator to have early access to the application performance before the design stage .

Each application is composed of a different number of tasks with different FIR. All tasks are mapped onto the network using either random mapping or n-map mapping algorithm [49]. Furthermore, each task communicates with one or multiple destinations. The specifications of the embedded application are parsed to the NoC-DPR through an application file that includes the number of tasks, the destination, and the FIR of each task.

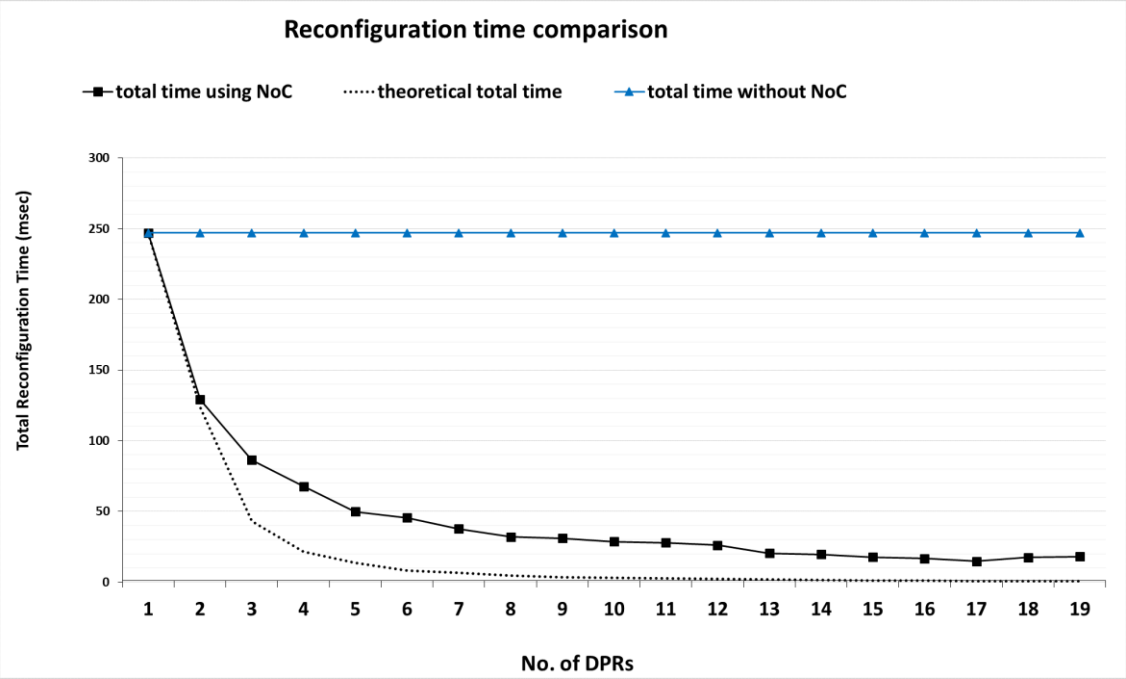


Figure 5-16: Comparison between the theoretical and the simulated multiple DPR for Wifi receiver application using 4-filts buffer depth

The multiplexing between many embedded applications using DPR is a prime benchmark for NoC-DPR simulator, as this experiment emphasizes the advantages of NoC-based FPGAs over the conventional SRAM-based FPGAs, where the main advantage is the ability to perform multiple DPRs simultaneously.

The network of wormhole routers with a buffer size of 4, 8, and 16-flits running at 100MHz are considered for this experiment. The study selects 802.11a WiFi receiver and multimedia system application. The 802.11a WiFi receiver has 24 tasks, and the multimedia system has 25 tasks as depicted in Figure 5-15a to Figure 5-15c. The reconfiguration is started by the farthest node from the master node (0, 0), then the next node as shown in Figure 5-15b, so on till all the 25 tasks are configured as illustrated in Figure 5-15c.

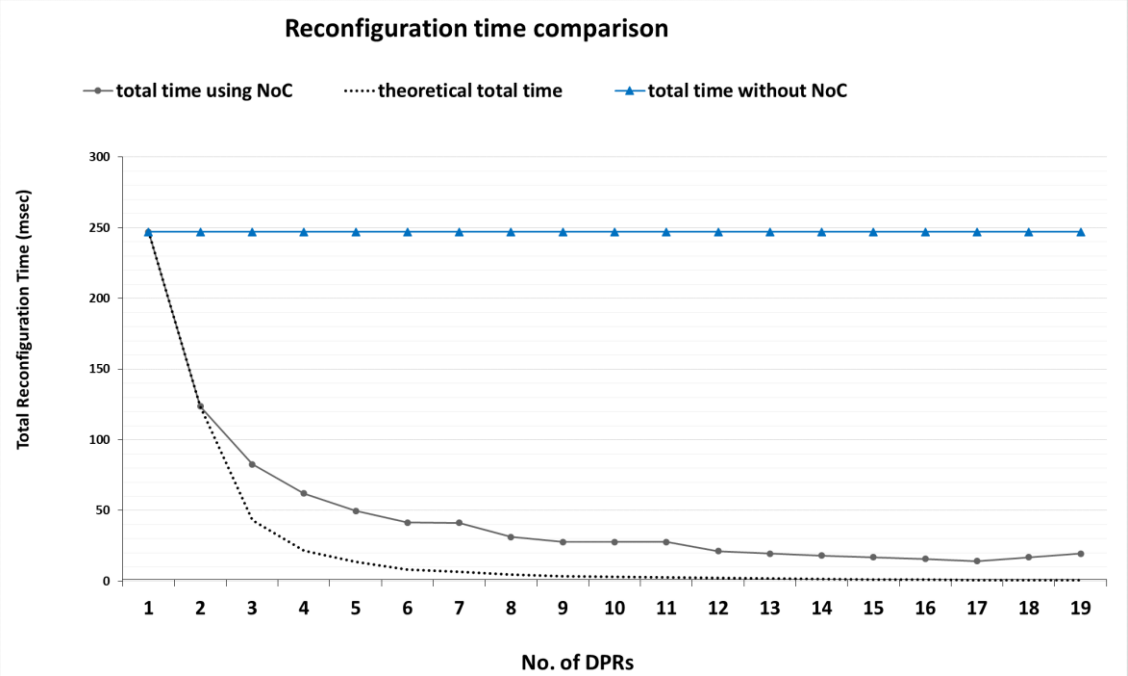


Figure 5-17: Comparison between the theoretical and the simulated multiple DPR for Wifi receiver application using 8-filts buffer depth

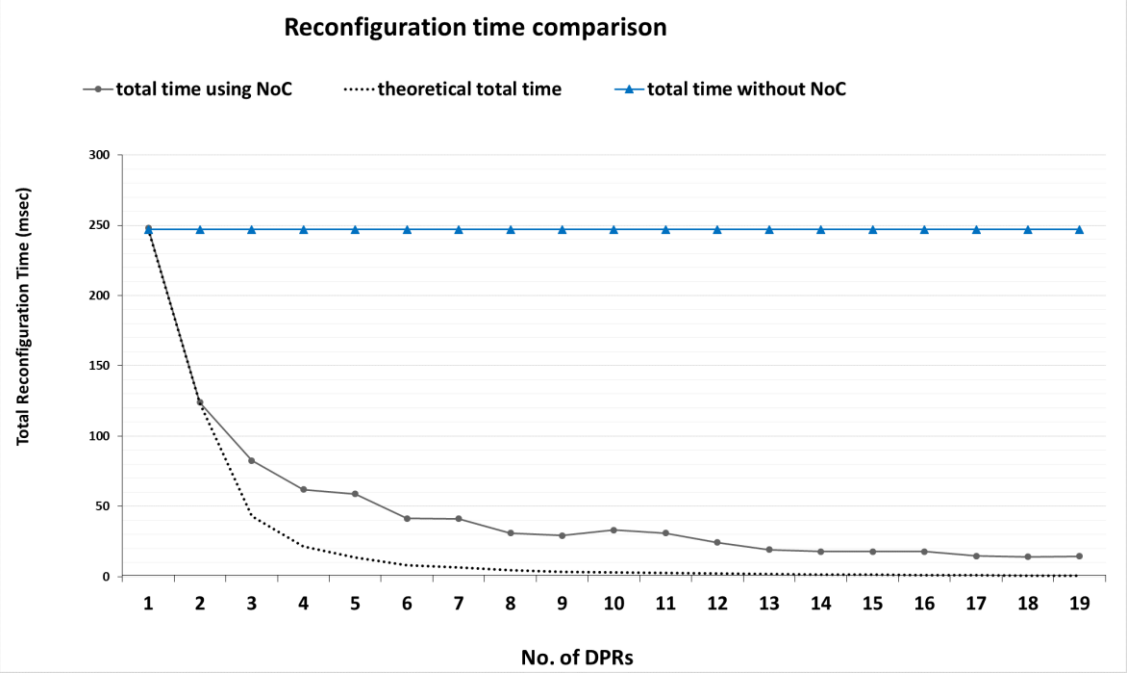


Figure 5-18: Comparison between the theoretical and the simulated multiple DPR for Wifi receiver application using 16-filts buffer depth

The impact of changing the number of simultaneous DPRs to switch between embedded applications is evaluated and compared to the theoretical results as depicted in Figure 5-16 for a network of wormhole routers with buffer depth 4-flits.

The total time of reconfiguring all tasks from multimedia application to or from WiFi application is measured by varying the number of DPRs simultaneously from 1-DPR to 24-DPR as illustrated in Figure 5-15 and compared to the conventional FPGA that allows only 1-DPR, where RT saturates at 245 msec.

Significantly, the difference between the theoretical and the simulated RT is slightly drifting as portrayed in Figure 5-16 using 4-flits buffer depth, as well as in Figure 5-17 and Figure 5-18 for 8 and 16 buffer depths receptively.

These findings are not in a complete agreement with the previous results using the wormhole router without VCs. Nevertheless, further tests are performed to resolve this contradiction; the comparison between the results of using wormhole router with eight flits buffer depth and wormhole router with 4-VCs is depicted in Figure 5-19. These results revealed the cause of the latter contradiction, which is the order of sending the reconfiguration packets; whereas, in this case study, the master node sends the reconfiguration packets in order, from the farthest to the nearest nodes. Nevertheless, the RT to switch between the WiFi and multimedia system is reduced by a factor of 12.25x using the NoC-based FPGAs.

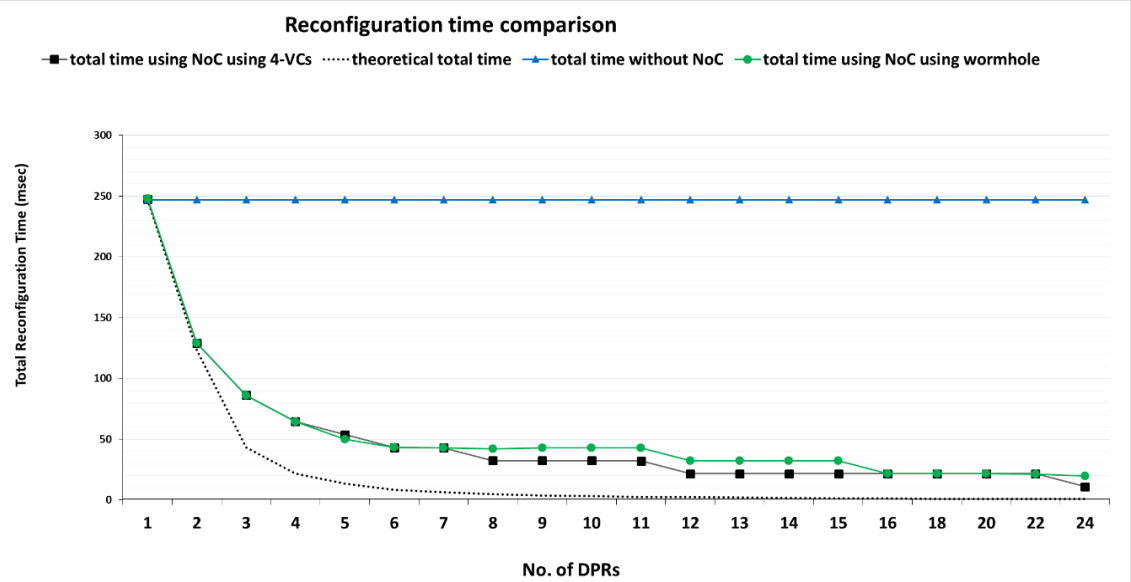


Figure 5-19: Comparison between the theoretical and the simulated multiple DPR for Wifi receiver application using 8-filts buffer depth of both wormhole router and wormhole router with 4-VCs

5.5. Design recommendations

Based on the previous experiments; some design insights and recommendations should be taken into consideration during the design of DPR on NoC-based FPGAs [51] [52]:

•A convential NoC platform cannot be used to implement DPR application directly. For instance, when one PE is performing DPR, the network should prevent other PEs from sending or receiving data to/from this PE until DPR is finished.

•Selecting a master node to control DPR's process: in the proposed NoC-DPR simulator, it is assumed that PE (0, 0) is the master of DPR process that is responsible for sending of reconfiguration packets as depicted in Figure 5-15.

•When the target PE receives the reconfiguration packet, it starts to perform reconfiguration. After the DPR is finished, the destination sends back to master node acknowledge packet to broadcast the availability.

•The clear advantage of using NoC-based FPGAs is the ability to perform multiple DPRs simultaneously. Thus numerous reconfiguration controllers should be distributed along with each PE.

•The recommended network size and number of simultaneous DPRs are estimated according to the desired RT. As portrayed in Figure 5-3 to Figure 5-14, the RT is the primary parameter over the other network parameters (i.e., latency and throughput), which affects the deviation between the theoretical and the simulated results directly. For instance, if the RT is 100 msec, up to 5-DPR simultaneously are executed, at any suitable network of wormhole routers with size from 2x2 to 9x9. In contrast, if the limit of RT is 1 msec, the optimal choice is 5-DPR simultaneously and wormhole network with 8-VC with a size larger than 10x10.

•Using virtual channel wormhole router enhances network latency and throughput, and reduces the RT.

5.6. Summary

In this chapter, two experiments are discussed. It is shown that NoC-based FPGA enhances the reconfiguration performance due to the ability to perform various DPRs at a time. The first experiment is executed using a synthetic application, where PEs of the network is theoretically mapped on Virtex-5 FPGAs, then the results of RT difference are calculated, the time difference between the theoretically and the simulated in both network of wormhole routers with and without virtual channels are estimated. The recommended network size and number of simultaneous DPRs are estimated according to the desired RT. It is found that the difference is always lower than 50% and 30% at small size networks for a network of wormhole routers with and without virtual channels respectively.

Finally, the RT is enhanced by a factor of 12.25x when switching between the WiFi and multimedia system.

Chapter 6 : Conclusions and Future work

Merging NoC with FPGA becomes an essential step for enhancing data communication, and adding DPR improves bandwidth utilization and increases designs efficiency and scalability.

The thesis's contribution is examined in four phases. In Chapter 2, an overview of FPGA's building blocks, an introduction for networking principles, and a survey of several NoC simulators are provided.

In Chapter 3, a comprehensive survey on the different techniques of DPR on Xilinx FPGA is presented. The four configuration methods used with DPR in Xilinx FPGAs are reviewed, and the results stated that JTAG's performance is better than the others with small designs where the area overhead is very noticeable. Despite that, the performance is not good with large designs where the space cost is not reasonable compared to the design area.

In Chapter 4, a comparison of the NoC-DPR performance to NoCTweak simulator performance is discussed, then the latency and throughput are estimated for 2-flits, 4-flits, and 8-flits buffer sizes and wormhole router and wormhole router with virtual channels.

Finally, Chapter 5 presents the work done to study the RT overhead, which resulted in an increasing number of simultaneous configurations on FPGA fabric using a synthetic and an embedded application. It is evident that NoC-based FPGA enhances reconfiguration performance because multiple configurations are performed simultaneously. However, supporting multiple DPRs needs to add more resources such as controlling unit and decoupling buffers. Accordingly, the reconfiguration time of DPR with NoC is better than the RT of DPR at conventional FPGA. Despite that, the number of simultaneous DPRs cannot exceed the signficant limit for specific network sizes, as no reduction in RT is gained, moreover, more resources are added.

In this work, a state-of-art NoC-DPR simulator is proposed, and some recommendations are extracted for the implementation of DPR on NoC-based FPGA to get the optimal size of the network concerning PE's logic resources.

It is shown that NoC-based FPGA enhances reconfiguration performance, and performs simultaneous DPRs. The first experiment is executed using a synthetic application, then the results of reconfiguration time difference are calculated, the time difference between the theoretically and the simulated in both network of wormhole routers with and without virtual channels. The recommended network size and number of simultaneous DPRs are estimated according to the desired reconfiguration time. It is found that the difference is always lower than 50% and 30% at small size networks for a network of wormhole routers with and without virtual channels respectively.

The DPR of NoC-based FPGA is studied and evaluated using an embedded application that switches from the multimedia system to the WiFi receiver. Furthermore, the reconfiguration time is estimated, which provides further evidence of enhancement

with factor 12.25x over the conventional SRAM-based FPGAs. Plenty of the experiments, results, and their conclusions in this thesis are published in [1, 51 and 52].

This research has three limitations: The first is the comparison of NoC-DPR simulation results with an existing simulators of similar architecture to validate the modifications on NoCtweak simulator. Second limitations is the overhead area estimation for the simulated hardwired module. The second is the power estimation. However, area and power overhead are estimated and added to the proposed simulator NoC-DPR in the future, as each module and sub-module on FPGA, where area and power are calculated by NoC-DPR using pre-calculated sub-modules. Finally, asynchronous support might be added, as the network interface in NoC-DPR simulator isolates PEs and the network. The future work will try to overcome these limitations, and hopefully, there will be a second release of the proposed NoC-DPR simulator.

References

1. A. Hassan, R. Ahmed, H. Mostafa, H. A. H. Fahmy and A. Hussien, "Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on FPGA," IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, pp. 183-186, 2015.

2. C. Hilton and B. Nelson. "PNoC: A Flexible Circuit-Switched NoC for FPGA-based Systems," IEE Proceedings Computers and Digital Techniques, Vol. 153, pp. 181-188, 2006.

3. D. Koch and al., "Partial reconfiguration on FPGAs in practice — Tools and applications," ARCS 2012, Munich, pp. 1-12, 2012.

4. D. P. Schultz, S. P. Young, and L. C. Hung, "Method and structure for reading, modifying and writing selected configuration memory cells of an FPGA," Xilinx Inc., Patent US 6255848, 1999.

5. Xilinx Inc. Development System Reference Guide.

6. Xilinx Inc. Virtex-4 Configuration Guide.

7. Xilinx Inc. Virtex and Virtex-E FPGA Data Sheets, 2000.

8. Xilinx Inc. Virtex-2 and Virtex-2 Pro FPGA Data Sheets, 2002.

9. Xilinx Inc. Processor Local Bus (PLB) v3.4, 2003.

10. Xilinx Inc. Virtex-4 Data Sheets, 2005.

11. Xilinx Inc. Virtex-4 User Guide, 2005.

12. Xilinx Inc. Virtex-5 User Guide, 2007.

13. Xilinx Inc. Virtex-5 Data Sheets, 2008.

14. Xilinx Inc. Virtex-6 Data Sheets, 2009.

15. Xilinx Inc. Virtex-7 Data Sheets, 2012.

16. W. S. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo and S. L. Sze, "A User Programmable Reconfigurable Logic Array," IEEE 1986 Custom Integrated Circuits Conference, pp. 233–235, 1986.

17. K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," ACM Computing Surveys, pp. 171–210, 2002.

18. A. Rohe and S. Teig, "Method and apparatus for identifying connections between configurable nodes in a configurable integrated circuit," Tabula Inc., Patent US 7284222, 2007.

19. Xilinx Inc. Partial Reconfiguration User Guide (UG702).

20. P. Manet, "An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications," EURASIP Journal on Embedded Systems, 2008.

21. P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537), Paris, pp. 250-256, 2000.

22. T. Bjerregaard and S. Mahadevan, “A survey of research and practices of Network-on-chip,” ACM Computer, New York, 2006.

23. M. Moadeli, “Quarc: An Architecture for Efficient On-Chip Communication,” PhD Thesis, University of Glasgow, 2010.

24. E. Salminen and al., “On network-on-chip comparison,” 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp. 503-510, 2007.

25. G. D. Micheli and al., “Networks on Chips: from Research to Products,” in Design Automation Conference, pp. 300-305, 2010.

26. A. Mello, L. Tedesco, N. Calazans and F. Moraes, “Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC,” 18th Symposium on Integrated Circuits and Systems Design, Florianopolis, pp. 178-183, 2005.

27. M. Selva, A. Gamatié, D. Novo and G. Sassatelli, “Speed and accuracy dilemma in NoC simulation: What about memory impact?” 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Tallinn, pp. 1-7, 2016.

28. V. Catania, A. Mineo, S. Monteleone, M. Palesi and D. Patti, “Noxim: An open, extensible and cycle-accurate network on chip simulator,” IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Toronto, pp. 162-163, 2015.

29. N. Jiang, G. Michelogiannakis, D. Becker and B. Towles, W. J. Dally, “Booksim 2.0 User's Guide,” Stanford University, 2010.

30. A. T. Tran and B. M. Baas. “NoCTweak: a Highly Parameterizable Simulator for Early Exploration of Performance and Energy of Networks On-Chip,” Dept. Electr. Comput. Eng., Univ. California, 2012.

31. S. Khan, S. Anjum, U. A. Gulzari and F. S. Torres, “Comparative analysis of network-on-chip simulation tools,” IET Computers & Digital Techniques, pp. 30-38, 2018.

32. L. Jain, B. Al-Hashimi and M. Gaur, “NIRGAM: a simulator for NoC interconnect routing and application modeling,” Workshop on Diagnostic Services in Network-on-Chips, DATE, pp. 16–20, 2007.

33. H. Hossain, M. Ahmed, A. Al-Nayeem, T. Z. Islam and M. M. Akbar, “Gpnocsim - A General Purpose Simulator for Network-On-Chip,” International Conference on Information and Communication Technology, Dhaka, pp. 254-257, 2007.

34. Y. Ben-Itzhak, E. Zahavi, I. Cidon and A. Kolodny, “HNOCS: Modular open-source simulator for Heterogeneous NoCs,” International Conference on Embedded Computer Systems (SAMOS), Samos, pp. 51-57, 2012.

35. A. Ehliar and D. Liu, “An FPGA based open source network-on-chip architecture,” 17th International Conference on Field Programmable Logic and Applications, FPL, IEEE Amsterdam, Holland, pp. 800-803, 2007.

36. M. Stensgaard and J. Spars, "ReNoC: A network-on-chip architecture with reconfigurable topology," Proc. Int. Symp. Networks-on-Chip (NoCS), pp. 55-64, 2008.

37. Xilinx Inc. Virtex-7 FPGAs Configuration User Guide (UG470). 2016.

38. Xilinx Inc. Virtex-5 FPGAs Configuration User Guide (UG191). 2012.

39. M. Liu, W. Kuehn, Z. Lu and A. Jantsch, "Run-time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration," Proceedings of FPL, Prague, 2009.

40. H. Tan, R. F. DeMara, A. J. Thakkar, A. Ejnioui and J. Sattler, "Complexity and Performance Evaluation of Two Partial Reconfiguration Interfaces on FPGAs: A Case Study," in Proceedings of ERSA'06, Las Vegas, pp. 253-256, 2006.

41. A. Raabe, S. Hochgurtel, G. Zachmann and J. K. Anlauf, "ReChannel: Describing and Simulating Reconfigurable Hardware in SystemC," ACM Transactions on Design Automation of Electronic Systems (TODAES), p.1-18, 2008.

42. Adriatic Consortium., "Advanced methodology for designing reconfigurable SoC and application-targeted IP-entities in wireless communications," 2002.

43. I. Benkhermi, A. Benkhelifa, D. Chillet, S. Pillement, J.-C. Prévotet and F. Verdier, "System-Level modelling for reconfigurable SoCs," the 20th Conference on Design of Circuits and Integrated Systems (DCIS), Lisbon, 2005.

44. A. V. De Briton, E. U. K. Melcher and W. Rosas, "An open-source tool for simulation of partially reconfigurable systems using SystemC," Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, p. 434, 2006.

45. A. Schallenberg, F. Oppenheimer and W. Nebel, "Designing for dynamic partially reconfigurable FPGAs with SystemC and OSSS," the Forum on Specification and Design Languages, Lille, 2004.

46. K. Papadimitriou, A. Anyfantis and A. Dollas, "Methodology and Experimental Setup for the Determination of System-Level Dynamic Reconfiguration Overhead," Proc of Field-Programmable Custom Computing Machines, pp. 335-336, 2007.

47. K. Papadimitriou, A. Dollas and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," ACM Transactions on Reconfigurable Technology and Systems (TRETS), p.1-24, 2011.

48. C. R. Berger, V. Arbatov, Y. Voronenko, F. Franchetti and M. Püschel, "Real-time software implementation of an IEEE 802.11a baseband receiver on Intel multicore," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Prague, pp. 1693-1696, 2011.

49. A. Kostrzewa, R. Ernst and S. Saidi, "Multi-path scheduling for multimedia traffic in safety critical on-chip network," 14th ACM/IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), Pittsburgh, pp. 1-10, 2016.

50. E. Carvalho, N. Calazans and F. Moraes, "Investigating runtime task mapping for NoC-based multiprocessor SoCs," 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC), Florianopolis, pp. 71-76, 2009.

51. A. Hassan, H. Mostafa, H. A. H. Fahmy and Y. Ismail, “Exploiting the Dynamic Partial Reconfiguration on NoC-Based FPGA,” New Generation of CAS (NGCAS), Genova, pp. 277-280, 2017.

52. A. Hassan, H. Mostafa and H. A. H. Fahmy, “NoC-DPR: A new simulation tool exploiting the Dynamic Partial Reconfiguration (DPR) on Network-on-Chip (NoC) based FPGA,” VLSI Integration, 2018.

Appendix A: NoC-DPR simulator Options

```
------------------- Platform Options -----------------------------
-platform [option]:       application traffic simulated on this platform.
  option = synthetic:    a synthetic traffic pattern (default)
  option = embedded:     an embedded application trace
  option = reconfig:     perform reconfig application
-seed [value]:    random seed for the simulation.
     (the same random seed will drive the same output results for
     the same network configuration. It's used for easier debugging.
      Default value = system time.)
-log [filename]:  log file for simulation outputs
-vcd [filename]:  VCD file for signal waveform traces
-simmode [option]:        simulation mode (packet or cycle)
-simtime [value]: simulation running time
  value = N. Default = 100,000.
  if simmode option = packet:    stop simulation after transferring N packets
  if simmode option = cycle:     stop simulation after running N clock cycles
-warmtime [value]:        warmup time for the network to become stable
  value = M (M < N). Default = 10,000.
  if simmode option = packet:    do not consider the first M received packets
  if simmode option = cycle:     warmup time is M clock cycles

------------------- Reconfigurable Options -----------------------------
-reconfig_time1 [value]: reconfiguration time for proc 1
-reconfig_time2 [value]: reconfiguration time for proc 2
-fir_rc [value]:  flit injection rate of reconfig (number of flits injected by each core
per cycle)
  0 < fir <= 1. Default = 0.1
-length_rc [value]:       the number of flits per rconfig packet.
  (only for the fixed packet length option. Default = 5.)
-rc_num [value]:  the number of the core to be reconfigured at the time.
  (Default = 1.)
------------------- Synthetic Options -----------------------------
-dimx [value]:    X dimension length of the 2-D mesh network. Default value = 8.
-dimy [value]:    Y dimension length of the 2-D mesh network. Default value = 8.
-traffic [option]:        synthetic traffic patterns used for the simulation.
  option = random:              uniform random (default)
  option = transpose:    transpose
  option = bitc:         bit-complement
  option = bitr:         bit-reverse
  option = tornado:             tornado
  option = shuffle:             bit-shuffle
  option = rotate:              bit-rotate
  option = neighbor:     nearest neighbor traffic
  option = regional:     communication distance <= 3
  option = hotspot:             central or corner hot spots
    -nhs [value]:               the number of hot spots. Default = 4.
```

-hstype [option]: hot-spot type
option = central: hot spots at the central cores
option = corner: hot spots at the corners (default)
-percent [value]: percentage of traffics going to neighboring or regional or hotspot cores

---------- Embedded Application Traces ----------
-appfile [option]: application task communication graph used in the simulation.
option = vopd.app: video object plan decoder with 16 tasks (default)
option = mms.app: multimedia system with 25 tasks
option = mwd.app: multi-window display with 12 tasks
option = wifirx.app: WiFi baseband receiver with 25 tasks
option = cavlc.app: H.24 CAVLC encoder with 16 tasks
option = mpeg4.app: MPEG4 decoder with 12 tasks
option = vce.app: video conference encoder with 25 tasks
option = autoindust.app: E3S auto-indust benchmark with 24 tasks
option = consumer.app: E3S consumer benchmark with 12 tasks
option = telecom.app: E3S telecom benchmark with 30 tasks
-appfile1 [option]: application task communication graph used in the simulation.
-mapping [option]: mapping algorithm used to map the task graph to the processor array
option = random: random mapping
option = nmap: near-optimal mapping using the NMAP algorithm

---------- Traffic Options ----------
-fir [value]: flit injection rate (number of flits injected by each core per cycle)
0 < fir <= 1. Default = 0.1
-dist [option]: probability distribution of the period between two injected packets
option = exponential: exponential distribution (default)
option = identical: identical distribution
-plength [option]: packet length is fixed or variable
option = fixed: fixed packet length (default)
option = variable: variable packet length
-length [value]: the number of flits per packet.
(only for the fixed packet length option. Default = 5.)
-lengthmin [value]: the minimum number of flits per packet
(only for the variable packet length option. Default = 2.)
-lengthmax [value]: the maximum number of flits per packet
(only for the variable packet length option. Default = 10.)

---------- Router Settings ----------
-router [option] the simulated router
option = wh: wormhole router (default)
option = vc: virtual-channel router
option = roshaq: RoShaQ share-queues router
option = bufferless: bufferless router
option = cs: circuit-switched router
-pptype [value]: pipeline type and the number of pipeline stages. Default = 3 stages
-bsize [value]: buffer depth (2, 4, 8, 16, 32 flits). Default = 4 flits.

-sbsize [value]: shared-buffer queue depth (2, 4, 8, 16, 32 flits). Default = 4 flits.
-nvc [value]: the number of virtual-channel buffers per input port. Default = 2 queues.
-nsb [value]: the number of shared-buffer queues in RoShaQ routers. Default = 5 queues.
-routing [option]: routing algorithm
option = xy: XY dimension-ordered routing (default)
option = nfminimal: Negative-First minimal adaptive routing
option = wfminimal: West-First minimal adaptive routing
option = nlminimal: North-Last minimal adaptive routing
option = oeminimal: Odd-Even minimal adaptive routing
option = table: lookup table based routing
-outsel [option]: choose an output port among multiple ones returned by an adaptive routing
option = xyordered: the X dimension first (default)
option = nearestdim: the dimension nearest to the destination first
option = farthestdim: the dimension farthest to the destination first
option = roundrobin: round-robin among output ports
option = credit: the output port having the highest credit first
-sa [option]: switch arbitration policy
option = rr: round-robin (default)
option = oldest: oldest first
option = takeall: winner takes all (only for virtual-channel routers)
option = islip: iSLIP based algorithm (only for virtual-channel routers)
-vca [option]: virtual-channel allocation policy (only for virtual-channel routers)
option = rr: round-robin (default)
option = oldest: oldest first
option = islip: iSLIP based algorithm
-llength [value]: inter-router link length (in um). Default = 1000 um.

---------- Environment Settings ----------
-technode [value]: CMOS technology process (90, 65, 45, 32, 22 nm). Default = 65 nm.
-freqmode [option]: clock frequency setting
option = fixed: fixed clock frequency (in MHz)
option = max: the maximum clock frequency supported by the router
-freq [value]: for fixed clock frequency (in MHz). Default = 1000 MHz.
-volt [value]: supply voltage (in V). Default = 1.0 V.

Figure A-1:NoC-DPR tool

Appendix B: Source Code of introduced modules

Header file Network Interface:

```
#ifndef PROC_NOC_INTER_H_
#define PROC_NOC_INTER_H_

#include <queue>
#include <systemc.h>
#include "../definition.h"
#include "../common_functions.h"
#include "../router/router_parameters.h"

#include "proc_parameters.h"

#include "../global_variables.h"

class proc_noc_interface: public sc_module{
  public:
    // clk and reset
    sc_in <bool> clk;
    sc_in <bool> reset;

    // Signal from the router saying that he received a flit for this interface
    sc_in <bool> r_valid_in;

    // Signal from the router with the flit to be received by this interface
    sc_in <Flit> r_flit_in;

    // Signal from virtual channels of the router saying that one buffer entry is available
    sc_in <bool> r_out_vc_buffer_rd[MAX_N_VCS];

    // Signal to router saying that this interface want to send a flit
    sc_out <bool> r_valid_out;

    // Signal to router with the flit to be sent by this interface
    sc_out <Flit> r_flit_out;

    // Signal to router saying saying that this interface has read an input flit on the given VC (the router can free it)
    sc_out <bool> r_in_vc_buffer_rd[MAX_N_VCS];

    sc_out <bool> r_interface_buffer_rd[MAX_N_VCS];

    // Signal from the processor saying that he received a flit for this interface
    sc_in <bool> p_valid_in;
```

```
// Signal from the processor with the flit to be received by this interface
sc_in <Flit> p_flit_in;

// Signal from virtual channels of the processor saying that one buffer entry is available
sc_in <bool> p_out_vc_buffer_rd;

// Signal to processor saying that this interface want to send a flit
sc_out <bool> p_valid_out;

// Signal to processor with the flit to be sent by this interface
sc_out <Flit> p_flit_out;

// Signal to processor saying saying that this interface has read an input flit on the given VC (the router can free it)
sc_out <bool> p_buff_out_full;
// Signal to processor saying saying that this interface has read an input flit on the given VC (the router can free it)
sc_out <bool> p_vaild_out_buff;

sc_signal <int> out_vc_remain[MAX_N_VCS];          // keep trace of number of idle entries of each output VC
sc_signal <int> out_vc_remain_reg[MAX_N_VCS];

sc_signal <int> in_vc_remain[MAX_N_VCS];           // keep trace of number of idle entries of each output VC
sc_signal <int> in_vc_remain_reg[MAX_N_VCS];

sc_signal <bool> out_buff_empty;
sc_signal <int> in_buff_empty;
sc_signal <int> in_buff_empty_reg;
sc_signal <bool> in_buf_out_buffer_rd;
sc_signal <bool> out_buf_out_buffer_rd;

sc_signal <bool> out_buffer_full;
sc_signal <bool> in_buffer_full;

int local_x;
int local_y;

queue <Flit> out_buffer;
queue <Flit> in_buffer;

// initialize all constants inside the processor (x,y)
void initialize(int x, int y);

void out_vc_remain_process();

void count_plus_process();  // pipelined out_vc_remain
void out_vc_remain_reg_process();// pipelined out_vc_remain
```

```
    void in_vc_remain_process();
    void in_vc_remain_reg_process();  // pipelined out_vc_remain

    void tx_process_out_buff();
    void rx_process_in_buff();

    void out_buffer_process();
    void in_buffer_process();

    // constructor
    SC_HAS_PROCESS(proc_noc_interface);
proc_noc_interface (sc_module_name name): sc_module(name){
            string in_buffer_name;
            in_buffer_name = "in_buffer_interface";

    buffer_size = RouterParameter::buffer_size;

            //in buff full
            SC_METHOD (tx_process_out_buff);
            sensitive << clk.pos() << reset.pos() ;

            SC_METHOD (rx_process_in_buff);
            sensitive << clk.pos() << reset.pos();

            SC_METHOD (in_buffer_process);
            sensitive << clk.pos() << reset.pos();

            SC_METHOD (out_buffer_process);
            sensitive << clk.pos() << reset.pos();

            // update out_vc_remain
            SC_METHOD (out_vc_remain_process);
            for (int vo=0; vo<RouterParameter::n_VCs; vo++){
                    sensitive << out_vc_remain_reg[vo];
                    sensitive << count_plus[vo];
                    sensitive << count_minus[vo];
            }

            // pipelined out_vc_remain
            SC_METHOD (out_vc_remain_reg_process);
            sensitive << clk.pos() << reset.pos();

            // count_plus = out_vc_buffer
            SC_METHOD (count_plus_process);
            for (int vo=0; vo<RouterParameter::n_VCs; vo++){
                    sensitive << r_out_vc_buffer_rd[vo];
            }
            SC_METHOD (in_vc_remain_process);
            for (int vo=0; vo<RouterParameter::n_VCs; vo++){
```

```
                sensitive << in_vc_remain_reg[vo];
                sensitive << in_count_plus[vo];
                sensitive << in_count_minus[vo];
            }

            // pipelined out_vc_remain
            SC_METHOD (in_vc_remain_reg_process);
            sensitive << clk.pos() << reset.pos();

    }
private:

    unsigned int buffer_size;       // number of flits

    sc_signal <bool> in_count_plus[MAX_N_VCS];     // = out_vc_buffer_rd
    sc_signal <bool> in_count_minus[MAX_N_VCS];
    sc_signal <bool> count_plus[MAX_N_VCS];        // = out_vc_buffer_rd
    sc_signal <bool> count_minus[MAX_N_VCS];
};

#endif /* PROC_NOC_INTER_H_ */
```

Header file of Process Element interface with Rechannel Library:

```
#ifndef PROC_RC_H_
#define PROC_RC_H_

#include <systemc.h>
#include "ReChannel.h"
#include "../definition.h"
#include "proc_evaluation.h"

#include "synthetic/without_ACK/synthetic_proc_rc.h"
#include "embedded/without_ACK/embedded_proc_rc.h"

#define NUM_RC_PROC 4

class procRCIf: public VirtualProc,
public rc_reconfigurable{
 public:
   // clk and reset
   rc_in_portal<bool> p_clk;
   //rc_in_portal<bool> p_reset;

   // Input interface
   rc_in_portal<bool> p_valid_in;
   rc_in_portal<Flit> p_flit_in;
   rc_out_portal<bool> p_in_vc_buffer_rd;

   // output interface
   rc_out_portal<bool> p_valid_out;
   rc_out_portal<Flit> p_flit_out;
   rc_in_portal<bool> p_out_vc_buffer_rd;     // "full" signals from virtual
channels of the local router port
   rc_in_portal <bool> p_vaild_in;
   //rc_in_portal<bool> p_reconf_done;

   //---------- functions
   // initialize all constants inside the processor (x,y)
  void initialize(int x, int y, EmbeddedAppHashTable* app_info=NULL);

   ProcEvaluationFactors *evaluation();
   void change_module();
   void reconfig_signal_process();
   //void reconfig_done_signal_process();
   EmbeddedProc_rc*          procEm;
   EmbeddedProc_rc*          procEm1;
   //SyntheticWithACKProc_rc procSynAck;
   SyntheticProc_rc*      procSyn;
   SyntheticProc_rc*      procSyn1;
   VirtualProc *currentProc;
```

```
VirtualProc *rcProc[NUM_RC_PROC];

rc_control ctrl;
sc_signal <bool> reset_int;
sc_signal <bool> recof_done_s;
// constructor
SC_HAS_PROCESS(procRCIf);
procRCIf (sc_module_name name): VirtualProc(name),
                ctrl("control")
{
        //procEm = new EmbeddedProc_rc("EmbeddedProc_rc");
        procSyn = new SyntheticProc_rc("SyntheticProc_rc", this);
        procSyn1 = new SyntheticProc_rc("SyntheticProc_rc1", this);
        procEm = new EmbeddedProc_rc("EmbeddedProc_rc", this);
        procEm1 = new EmbeddedProc_rc("EmbeddedProc_rc1", this);

        rcProc[0] = procSyn;
        rcProc[1] = procSyn1;
        rcProc[2] = procEm;
        rcProc[3] = procEm1;

        p_clk.static_port(clk);
        procSyn->reset(reset);
        procSyn1->reset(reset);
        procEm->reset(reset);
        procEm1->reset(reset);


        p_valid_in.static_port(valid_in);
        p_flit_in.static_port(flit_in);

        p_valid_out.static_port(valid_out);
        p_flit_out.static_port(flit_out);
        p_in_vc_buffer_rd.static_port(in_vc_buffer_rd);
        p_out_vc_buffer_rd.static_port(out_vc_buffer_rd);
        p_vaild_in.static_port(out_buf_vaild_in);
        for(int i=0; i<NUM_RC_PROC; i++){
                p_clk.dynamic_port(rcProc[i]->clk);

                p_valid_in.dynamic_port(rcProc[i]->valid_in);
                p_flit_in.dynamic_port(rcProc[i]->flit_in);

                p_valid_out.dynamic_port(rcProc[i]->valid_out);
                p_flit_out.dynamic_port(rcProc[i]->flit_out);

                p_in_vc_buffer_rd.dynamic_port(rcProc[i]->in_vc_buffer_rd);
                p_out_vc_buffer_rd.dynamic_port(rcProc[i]->out_vc_buffer_rd);
                p_vaild_in.dynamic_port(rcProc[i]->out_buf_vaild_in);
        }
```

```
                procSyn->rc_set_delay(RC_LOAD,
sc_time(ProcessorParameters::proc_reconfig_time_1, SC_NS));
                procSyn1->rc_set_delay(RC_LOAD,
sc_time(ProcessorParameters::proc_reconfig_time_1, SC_NS));
                procEm->rc_set_delay(RC_LOAD,
sc_time(ProcessorParameters::proc_reconfig_time_1, SC_NS));
                procEm1->rc_set_delay(RC_LOAD,
sc_time(ProcessorParameters::proc_reconfig_time_1, SC_NS));
                ctrl.add (*procSyn + *procSyn1 + *procEm + *procEm1);
                if(CommonParameter::platform_type                    ==
PLATFORM_RECONFIG_EM){
                        ctrl.activate(*procEm);
                   currentProc=procEm;
                   active_module = EM;
                }
                else{
                ctrl.activate(*procSyn);
          currentProc=procSyn;
          active_module = SYN;
                }
                //active_module = &procEm;
          do_activate_syn = false ;
          do_activate_syn1 = false ;

          do_activate_em = false ;
          do_activate_em1 = false ;

                SC_THREAD(reconfig_signal_process);
           sensitive << reset.pos() << clk.pos() << procSyn->do_activate_em <<
procSyn1->do_activate_em
                        << procEm->do_activate_em << procEm1->do_activate_em;

        }
        ~procRCIf(){

                delete procSyn;
                delete procSyn1;
                delete procEm;
                delete procEm1;

        }
      private:
        enum modules{SYN,SYN1, EM, EM1} active_module;
        sc_signal<bool>  do_activate_syn1;
        sc_signal<bool>  do_activate_em1;

    };

    #endif /* PROC_RC_H_ */
```

ملخص الرسالة

نتيجه الزيادة المستمرة في تصغير اجهزه أشباه الموصلات اتاح للمصنعين زياده عدد التطبيقات على نفس الدائره المصنعه. ولهذا اصبح مفهوم وجود نظم كامله بداخل الرقائق إلكترونية ضرورى, حيث هو نظام يتكون من وحدات معالجه و وحدات تخزينيه متصله بواسطه نظام اتصالات معقد. فى خلال السنين الماضيه الاتصال بين الوحدات معالجه و الوحدات تخزينيه اصبح عاملا حيويا في تصميم الأنظمة واسعة النطاق. بينما التركيز على زيادة عدد الوحدات معالجه علي التوازي من أجل تعظيم القدرة على التصاميمات الحديثة، زادت الطاقة المستهلكة والتطبيقات ذات البيانات الكثيفة ظهرت.

ونتيجة لذلك، فإن العديد من التحديات في التواصل بين هذه الوحدات المعالجه ، عند تكوينها على الدوائر المصفوفات القابلة للبرمج، أصبحت كبيرة وتتطلب حلول مبتكرة. ولذلك تم تكييف مفهوم بارز للاتصال المعروفة باسم شبكة لتوصيل المعلومات على الرقائق الاكترونيه، لدوائر المصفوفات القابلة للبرمجة للتعامل مع هذه التحديات.

ومن ناحية أخرى، اصبح إعادة التشكيل الجزئي الديناميكي لدوائر المصفوفات القابلة للبرمجة التي تعتمد على ذاكرة الوصول العشوائي الثابتة سمة مطلوبه من قبل العديد من التطبيقات لإمكانية إضافة المزيد من المرونة على مرحلة التشغيل.

في الآونة الأخيرة، تصاميم التطبيقات التى تستخدم إعادة التشكيل الجزئي الديناميكي اصبح أسهل من ذي قبل. ومع ذلك، والتقنيات التي تستخدمها الدوائر المصفوفات القابلة للبرمجة لأداء إعادة التشكيل الجزئي الديناميكي (مثل منفذ الوصول التكوين الداخلي ومجموعة عمل الاختبار المشترك) تواجه مشكله فى الأداء, لانه يسمح لعمل إعادة التشكيل الجزئي الديناميكي واحد فى الوقت.

في تلك الرسالة نقترح محاكى لدوائر المصفوفات القابلة للبرمجة القائمه على وجود شبكة لتوصيل المعلومات بداخلها والذى يدعم محاكاة إعادة التشكيل الجزئي ديناميكي ايضا. وتم احتساب حدود التصميم وتابين الأداء عند استعمال إعادة التشكيل الجزئي ديناميكي لدوائر المصفوفات القابلة للبرمجة القائمه على وجود شبكة لتوصيل المعلومات باستخدام المحاكي المقترح.

تجرى التجارب باستخدام محاكي المقترح لقياس الوقت الزائد فى إعادة تشكيل والذي ينتج عن زمن استجابة الشبكة توصيل المعلومات وعدد إعادة التشكيل الجزئي ديناميكي في وقت واحد على نسيج المصفوفات القابلة للبرمجة القائمه. ويتضح أن الوقت الزائد لاإعادة التشكيل يزيد أضعافا مضاعفة مع زيادة عدد إعادة التشكيل الجزئي ديناميكي في وقت واحد. ومع ذلك، تظهر مزيد من التحقيقات أن شبكة توصيل المعلومات مكونه من أجهزة التوجيه الدوديه ذات القنوات افتراضية يحسن وقت إعادة تشكيل الى عامل قد يصل اربعه اضعاف من شبكة توصيل المعلومات مكونه من أجهزة التوجيه الدوديه دون قنوات افتراضية. وأخيرا يتم تقديم دراسة لتوضيح الفجوة بين أداء إعادة التشكيل الجزئي ديناميكي بين دوائر المصفوفات القابلة للبرمجة القائمه على وجود شبكة لتوصيل المعلومات و دوائر المصفوفات القابلة للبرمجة التقليدية.

مهنـدس: **عمرو حسن على بدار**
تاريخ الميلاد: ١٩٨٩\١١\٢٤
الجنسية: مصرى
تاريخ التسجيل: ٢٠١٢\١٠\١
تاريخ المنح: \ \٢٠١٩
القسم: **هندسة الإلكترونيات و الإتصالات الكهربية**
الدرجة: **ماجستير العلوم**
المشرفون: **أ.د. حسام على حسن فهمى**
د. حسن مصطفى حسن
الممتحنون:

أ.د حسام على حسن فهمى **(المشرف الرئيسي)**
أستاذ الإلكترونيات بكلية الهندسة – جامعة القاهرة

أ.د أحمد حسين خليل **(الممتحن الداخلى)**
أستاذ الإلكترونيات بكلية الهندسة – جامعة القاهرة

أ.د .محمد عبد الغنى سالم **(الممتحن الخارجى)**
أستاذ الإلكترونيات بكلية الهندسة – جامعة الالمانيه بالقاهره

عنوان الرسالة:
استكشاف محاكاة إعادة التشكيل الجزئي الديناميكي لتطبيق شبكات توصيل المعلومات على نظم المصفوفات القابله للبرمجه

الكلمات الدالة:
دوائر مصفوفات البوابات المنطقيه, إعادة التشكيل الجزئي الديناميكي, شبكات توصيل المعلومات على الرقائق الاكترونيه

ملخص الرسالة:
في هذه الرساله، يتم عرض نبذه عن التقنيات المستخدمه لعمل إعادة التشكيل الجزئي الديناميكي في دوائر مصفوفات البوابات المنطقيه التقليدية، ثم يتم تقديم استعراض مقارن بين هذه التقنيات فيما يتعلق وقت إعادة تشكيل والمساحه المستخدمه. وقد تم تحليل معلومات مختلفة للشبكه المتاحه فى دوائر المصفوفات القابلة للبرمجة القائمه على وجود شبكة لتوصيل المعلومات لتقدير التأثير على أداء إعادة التشكيل الجزئي الديناميكي باستخدام المحاكى المقترح, وتم اقتراح تصاميم و توصيات للمصممين للتحقق من الحجم مناسب لشبكة لتوصيل المعلومات على الرقائق الاكترونيه باستخدام زمن إعادة تشكيل.

استكشاف محاكاة إعادة التشكيل الجزئي الديناميكي لتطبيق شبكات توصيل المعلومات على نظم المصفوفات القابله للبرمجه

اعداد

عمرو حسن على بدار

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الإلكترونيات و الإتصالات الكهربية

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: حسام على حسن فهمى **المشرف الرئيسى**
أستاذ الإلكترونيات بكلية الهندسة – جامعة القاهرة

الاستاذ الدكتور: احمد حسين خليل **الممتحن الداخلى**
أستاذ الإلكترونيات بكلية الهندسة – جامعة القاهرة

الاستاذ الدكتور: محمد عبد الغنى سالم **الممتحن الخارجى**
أستاذ الإلكترونيات بكلية الهندسة – الجامعة الالمانيه بالقاهره

كليــة الهندســة - جامعــة القاهــرة

الجيـزة - جمهوريـة مصـر العربيــة

٢٠١٩

استكشاف محاكاة إعادة التشكيل الجزئي الديناميكي لتطبيق شبكات توصيل المعلومات على نظم المصفوفات القابله للبرمجه

اعداد

عمرو حسن على بدار

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الإلكترونيات و الإتصالات الكهربية

تحت اشراف

أ.د. حسام على حسن فهمى

أستاذ

قسم هندسة الإلكترونيات و الإتصالات الكهربية

كلية الهندسة - جامعة القاهرة

د. حسن مصطفى حسن مصطفى

مدرس

قسم هندسة الإلكترونيات و الإتصالات الكهربية

كلية الهندسة - جامعة القاهرة

كليــة الهندســة - جامعــة القاهـرة

الجيـزة - جمهوريـة مصـر العربيــة

٢٠١٩

استكشاف محاكاة إعادة التشكيل الجزئي الديناميكي لتطبيق شبكات توصيل المعلومات على نظم المصفوفات القابله للبرمجه

اعداد

عمرو حسن على بدار

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

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة
٢٠١٩