Interconnect Architectures for Dynamically Partially Reconfigurable Systems

by

Thanh Thi Thanh Bui

B.Eng. (Electronic and Telecommunication Engineering),
The University of Da Nang, Vietnam, 2011
M.Eng. (Electronic and Computer Engineering),
RMIT University, Australia, 2014

Thesis submitted for the degree of

Doctor of Philosophy

in

Electrical and Electronic Engineering,
Faculty of Engineering, Computer and Mathematical Sciences
The University of Adelaide, Australia

December, 2017

Supervisors:

Dr Braden Phillips, School of Electrical & Electronic Engineering

Prof Michael Liebelt, School of Electrical & Electronic Engineering

THE UNIVERSITY of ADELAIDE

Contents

Abstract

Dynamically partially reconfigurable FPGAs (Field-Programmable Gate Arrays) allow hardware modules to be placed and removed at runtime while other parts of the system keep working. With their potential benefits, they have been the topic of a great deal of research over the last decade. To exploit the partial reconfiguration capability of FPGAs, there is a need for efficient, dynamically adaptive communication infrastructure that automatically adapts as modules are added to and removed from the system. Many bus and network-on-chip (NoC) architectures have been proposed to exploit this capability on FPGA technology. However, few realizations have been reported in the public literature to demonstrate or compare their performance in real world applications.

While partial reconfiguration can offer many benefits, it is still rarely exploited in practical applications. Few full realizations of partially reconfigurable systems in current FPGA technologies have been published. More application experiments are required to understand the benefits and limitations of implementing partially reconfigurable systems and to guide their further development. The motivation of this thesis is to fill this research gap by providing empirical evidence of the cost and benefits of different interconnect architectures. The results will provide a baseline for future research and will be directly useful for circuit designers who must make a well-reasoned choice between the alternatives.

This thesis contains the results of experiments to compare different NoC and bus interconnect architectures for FPGA-based designs in general and dynamically partially reconfigurable systems. These two interconnect schemes are implemented and evaluated in terms of performance, area and power consumption using FFT (Fast Fourier Transform) and ANN (Artificial Neural Network) systems as benchmarks. Conclusions drawn from these results include recommendations concerning the interconnect approach for different kinds of applications. It is found that a NoC provides much better performance than a single channel bus and similar performance to a multi-channel bus in both parallel and parallel-pipelined FFT systems. This suggests that a NoC is a better

choice for systems with multiple simultaneous communications like the FFT. Bus-based interconnect achieves better performance and consume less area and power than NoC-based scheme for the fully-connected feed-forward NN system. This suggests buses are a better choice for systems that do not require many simultaneous communications or systems with broadcast communications like a fully-connected feed-forward NN. Results from the experiments with dynamic partial reconfiguration demonstrate that buses have the advantages of better resource utilization and smaller reconfiguration time and memory than NoCs. However, NoCs are more flexible and expansible. They have the advantage of placing almost all of the communication infrastructure in the dynamic reconfiguration region. This means that different applications running on the FPGA can use different interconnection strategies without the overhead of fixed bus resources in the static region.

Another objective of the research is to examine the partial reconfiguration process and reconfiguration overhead with current FPGA technologies. Partial reconfiguration allows users to efficiently change the number of running PEs to choose an optimal power-performance operating point at the minimum cost of reconfiguration. However, this brings drawbacks including resource utilization inefficiency, power consumption overhead and decrease in system operating frequency. The experimental results report a 50% of resource utilization inefficiency with a power consumption overhead of less than 5% and a decrease in frequency of up to 32% compared to a static implementation. The results also show that most of the drawbacks of partial reconfiguration implementation come from the restrictions and limitations of partial reconfiguration design flow. If these limitations can be addressed, partial reconfiguration should still be considered with its potential benefits.

Statement of Originality

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signed

Date

Acknowledgments

First and foremost, I am deeply grateful to my principal supervisor, **Dr Braden Phillips**, for his constant support throughout my research time at the University of Adelaide. This thesis would not be completed without his encouragement, constructive feedback and critical comments. To me, he is a wonderful research supervisor and mentor whose advice I always trust.

I am also indebted my co-supervisor, **Prof Michael Liebelt**, for all advice and feedback he has provided throughout my candidature.

I would like to thank staff and postgraduates at the School of Electrical and Electronic Engineering. I have had a great time doing research at the School thanks to their constant support and friendship.

I also thank my Vietnamese friends, Andy, Duong, Dung, My, Phuong, Loc, Huyen, Quynh, Nguyet, Nghia, Cuong. I really enjoyed the weekends hanging out with them.

I extended my thanks to the examiners of this thesis. It is not easy to write a thesis and it is no less easy to review one.

Last but not least, I would like to give my endless love and appreciation to my fabulous family who were the pillars of emotional support and encouragement. Mother, thank you for giving birth to me and encouraging me through those tough years.

Thanh Thi Thanh Bui
December 2017

List of Acronyms

ANN Artificial Neural Network

ASIC Application-Specific Integrated Circuit

BRAM Block RAM

BUFG Global Clock Buffer

CLB Configurable Logic Block

CoNoChi Configurable Network-on-Chip

CuNoC Communication Unit Network-on-Chip

DRNoC Dynamic Reconfigurable Network-on-Chip

DSP Digital Signal Processing

DynaCORE Probability Distribution Function

DyNoC Dynamic Network-on-Chip

EEFROM Electrical Erasable Programmable Read-Only Memory

EFROM Erasable Programmable Read-Only Memory

FIFO First In First Out

FPGA Field-Programmable Gate Array

GSM Global System for Mobile Communication

HDL Hardware Description Language

ICAP Internal Configuration Access Port

NoC Network-on-Chip

FFT Fast Fourier Transform

LUT Look-Up Table

MGT Multi-Gigabit Transceiver

OCEAN On-Chip Efficiently Adaptive Network

PAR Place And Route

PE Processing Element

RMBoC Reconfigurable Multiple Buses on Chip

SRAM Static Random Memory Access

List of Figures

List of Tables

Chapter 1

Introduction

THIS chapter describes the context and motivation of the research presented in this thesis. It briefly highlights the development and advantages of FPGAs and their capacity for dynamic partial reconfiguration. The objectives of the thesis are then stated, followed by an overview of the thesis structure.

1.1 Background

Since their first introduction in 1984, FPGAs (Field-Programmable Gate Arrays) have become common platforms for a wide range of applications (Trimberger, 2015). By providing reconfiguration flexibility, hardware parallelism, ease-of-use and time to market advantages, they are competitive alternatives to ASICs (Application-Specific Integrated Circuits) and microprocessors (Fawcett, 1994). Over three decades of development, FPGA capacity and speed have increased by a factor of 10000 and a factor of 100 respectively, while their cost and power consumption per unit have decreased by more than a factor of 1000 (Trimberger, 2015). Today's high-end FPGA devices have millions of LUTs (look-up tables), megabytes of on-chip memory and thousands of DSP (Digital Signal Processing) blocks. Higher-end FPGAs can contain hard blocks such as high speed multi-gigabit transceivers, processors, PCI/PCI Express and external memory controllers. FPGAs are now widely used in a diverse variety of applications in digital signal processing, computer vision, communications, and other growing applications in military, aerospace, and consumer electronics (Tessier *et al.*, 2015). The global market size of FPGAs was valued at USD 6.36 billion in 2015. It is estimated to reach to USD 14.2 billion by 2024 according to a report by Grand View Research (Grand View Research, 2016).

An advantage of FPGAs is their reconfiguration flexibility. An FPGA can be globally or partially reconfigured for a modified design without the need for re-fabrication. With a global reconfiguration, the entire design implemented by the FPGA is exchanged for a modified design. Partial reconfiguration brings this flexibility to a higher level, allowing hardware modules to be placed and removed at runtime while other parts of the system keep working (Xilinx, 2013). This permits a radical departure from the way application-specific hardware is usually designed. In a static system, there must be a fixed set of processing resources sufficient to meet performance requirements under worst-case load conditions. If the workload changes, processing resources sit idle. A system that moves through modes with distinctly different processing needs should provide different resources for each mode. Dynamic partial reconfiguration can: reduce power consumption by removing resources not currently required; achieve better utilization by changing the mix of processing resources as the requirements of the

system change; and deliver better performance by using heterogeneous processing resources optimized for particular stages in an algorithm, rather than making do with static generic processing resources that must serve all stages.

1.2 Research Motivation

With their potential benefits, dynamically partially reconfigurable FPGAs have been the topic of a great deal of research over the last decade (Zhang *et al.*, 2010; Ahmad *et al.*, 2010; Bonamy *et al.*, 2012; Dennl *et al.*, 2012; Wang *et al.*, 2013; Ahmad *et al.*, 2013). To exploit the partial reconfiguration capability of FPGAs, there is a need for efficient, dynamically adaptive communication infrastructure that automatically adapts as modules are added to and removed from the system. Many bus and network-on-chip (NoC) architectures have been proposed to exploit this capability on FPGA technology. However, few realizations have been reported in the public literature to demonstrate or compare their performance in real world applications.

While partial reconfiguration can offer many benefits, it is still rarely exploited in practical applications. Few full realizations of partially reconfigurable systems in current FPGA technologies have been published. More application experiments are required to understand the benefits and limitations of implementing partially reconfigurable systems and to guide their further development.

The motivation of this thesis is to fill this research gap by providing empirical evidence of the cost and benefits of different interconnect architectures. The results will provide a baseline for future research and will be directly useful for circuit designers who must make a well-reasoned choice between the alternatives.

1.3 Research Objectives

The main objective of the research presented in this thesis is to examine different NoC and bus interconnect architectures for FPGA-based designs in general and dynamically partially reconfigurable systems. These two interconnect schemes are implemented and evaluated in terms of performance, area and power consumption using

FFT (Fast Fourier Transform) and ANN (Artificial Neural Network) systems as benchmarks. Based on that, advantages and disadvantages of each of the communication architectures are clearly pointed out. Conclusions drawn from these results will include recommendations concerning the interconnect approach for different kinds of applications.

Another objective of the research is to examine the partial reconfiguration process and reconfiguration overhead with current FPGA technologies. Reconfiguration overhead is measured in terms of resource overhead, time and power consumption of the reconfiguration process. Based on experimental results, advantages and disadvantages of dynamic partial reconfiguration are stated. Limitations of current FPGA architectures and design flows for partial reconfiguration are discussed and some enhancements are proposed.

1.4 Original Contributions

The examination of different interconnect architectures in fully implemented systems will provide empirical results for designers who work on FPGA-based design in general and dynamically partially reconfigurable systems. Imagine a designer, considering whether to use partial reconfiguration for a commercial design. There is insufficient information in the public literature for them to be able to determine whether the approach is viable without first implementing and testing their own system. This thesis will implement different interconnect approaches in fully realized systems and point out their benefits as well as drawbacks designers need to consider for an efficient design.

The two case-study systems developed to conduct this research are contributions in their own right. To the best of my knowledge, the FFT system is the first in the public literature using different communication approaches to handle the interconnect complexity in a parallel architecture. It is the first reported implementation of an FFT system that can exploit dynamic partial reconfiguration. The ANN system is one of the largest fully realized FPGA implementations of an ANN in the literature. Its architecture is sufficiently general for a wide variety of different kinds of ANN and it is scalable to the kinds of ANNs now popular for deep learning (Schmidhuber, 2015). The source

code for both of these systems have been made open source and publicly available for further research (Bui, 2017).

The results of this thesis have been presented and included in the following papers:

1. A dynamically reconfigurable NoC for double-precision floating-point FFT on FPGAs (Bui *et al.*, 2016). This work has been presented in the Ninth International Conference on Advances in Circuits, Electronics and Micro-electronics (IARIA CENICS 2016).
2. A scalable NoC-based neural network implementation on FPGAs (ready to submit).
3. A comparison of NoC and bus interconnect for dynamically partially reconfigurable FPGAs (in submission).

1.5 Thesis Structure

The thesis is divided into 6 chapters, including the Introduction and Conclusion chapters. The main contributions of each chapter are highlighted as follows.

- Chapter 2 provides a review of the technologies relevant to this thesis. It starts with a brief overview of FPGA architectures and programming technologies. Then, a review of NoC and bus interconnect architectures for on-chip systems is provided. Finally, related works on different NoC and bus architectures for dynamically partially reconfigurable systems are presented. These architectures are clearly classified and organized in sub-categories. Advantages and disadvantages of each architecture are also discussed.
- Chapter 3 presents the implementations of FFT systems using NoC and bus communications on FPGAs. Hardware design of the FFT is discussed in details. Bus and NoC architectures customized for the FFT systems are implemented and evaluated in terms of area, power consumption, and performance. The proposed FFT systems are also compared with other hardware implementations as well as software implementations on CPUs.

- Chapter 4 presents another case study, a feed-forward neural network for handwritten digit recognition. Dedicated NoC and bus interconnects are implemented to best suit the massive communication requirements of the network. Different design configurations are discussed and examined. Experiments are carried out to compare area, power consumption, and performance of the two interconnect architectures for the neural network application. The advantages of the proposed NN implementations compared to traditional implementations are also highlighted. Combining with the results in Chapter 3, this chapter gives recommendations on the applications and communication requirements each interconnect architecture is better used for.

- Chapter 5 presents the dynamically partially reconfigurable implementations for the FFT and the NN systems presented in Chapter 3 and Chapter 4. Experiments are carried out to measure resources overhead, time and power consumption of the reconfiguration process. Comparisons between dynamic and static implementations are made to highlight the advantages and disadvantages of dynamic partial reconfiguration. Based on experiments with some current FPGA devices, limitations of current FPGA architectures and design flows for partial reconfiguration are also discussed. In addition, the benefits and limitations of NoC and bus interconnect architectures for dynamic partial reconfigurable systems are also stated.

- Chapter 6 concludes the thesis with a summary of thesis contribution and significance, and proposes directions for future work and improvements.

Chapter 2

Literature Review

THIS chapter provides a review of the literature of technologies related to this thesis. It starts with a brief overview of FPGA architectures and programming technologies. Then, a review of NoC and bus interconnect architectures for on-chip systems is provided. Finally, related works on different NoC and bus architectures for dynamically partially reconfigurable systems are presented. These architectures are clearly classified and organized in sub-categories. Advantages and disadvantages of each architecture are also discussed.

2.1 Overview of FPGAs

FPGAs are pre-fabricated integrated circuits that can be reconfigured to perform almost any kind of custom digital functions. Commercial FPGAs from different vendors differ in their underlying programming technologies and their architectures.

2.1.1 Programming Technologies

FPGAs programming technologies have a significant impact on their programmable logic architectures. There are a number of programming technologies. Some common approaches are SRAM, anti-fuse and floating gate programming technology.

The SRAM programming technology uses static memory cells as the basic configurable elements. These are distributed throughout the FPGAs to select control lines of multiplexers or to store data in LUTs to implement logic functions (Kuon *et al.*, 2008). Fig. 2.1 shows an illustration of SRAM programmable switches to route signals between logic blocks on an FPGA. SRAM programming technology is widely used in modern FPGAs due to its two main advantages: fast reprogramming; and compatibility with standard CMOS logic process technology. However, it has some major drawbacks. SRAM is volatile, hence SRAM-based FPGAs must be reconfigured every time the chip powers up, which requires an external memory to store the configuration data. Another disadvantage of this programming approach is its large size. A SRAM cell to store one bit of configuration data requires 5 or 6 transistors in addition to 1 transistor for routing. Despite these drawbacks SRAM programming technology is the dominant method in modern commercial FPGAs. SRAM-based FPGAs are also the main platform that is used for experiments and evaluation in this research.

An alternative programming technology uses anti-fuses as the programmable elements. An anti-fuse is a two-terminal device with a fixed state presenting a very high resistance between its terminals and a programmed state presenting a low resistance link. It is possible to switch the device from the fixed state to the programmed state by applying a high voltage from 11 to 20 volts across its terminals (Rose *et al.*, 1993). Unlike SRAM programming technology, this link is permanent. The main advantages of the anti-fuse programming approach are its small size and its low capacitance and

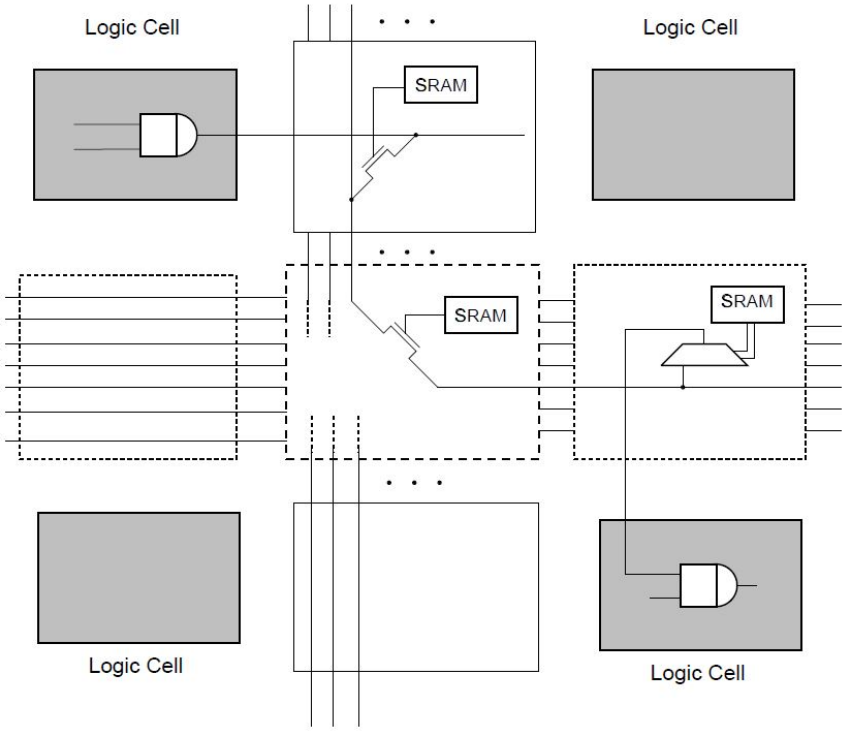


Figure 2.1: SRAM programmable switches (Kuon *et al.*, 2008).

resistance. In addition, the non-volatile characteristic allows a programmed device to work instantly once power is applied. However, its major drawback is that it requires a non-standard CMOS fabrication process. Note that CMOS currently dominates the IC industry, especially for high-density devices. Using a non-standard process adds significantly to the cost of manufacture and hence to the per-unit cost for anti-fuse FPGA devices.

The floating gate programming technology is based on the technology used in ultra-violet EPROM and electrically erasable EEPROM. The programmable switch is a transistor that can be disabled by injecting a charge onto a floating gate terminal. This increases the threshold voltage of the transistor (Brown and Rose, 1996). An illustration of EFROM programmable switches is given in Fig. 2.2. The floating gate programming technology has the advantages of reprogramming ability like the SRAM programming technology and non-volatility like the anti-fuse programming technology. There is no need for external memory storage of the configuration data. Similar the anti-fuse programming technology, it also has the disadvantage of a a non-standard CMOS process.

Another significant drawback is its high resistance and capacitance due to the use of transistor-based switches.

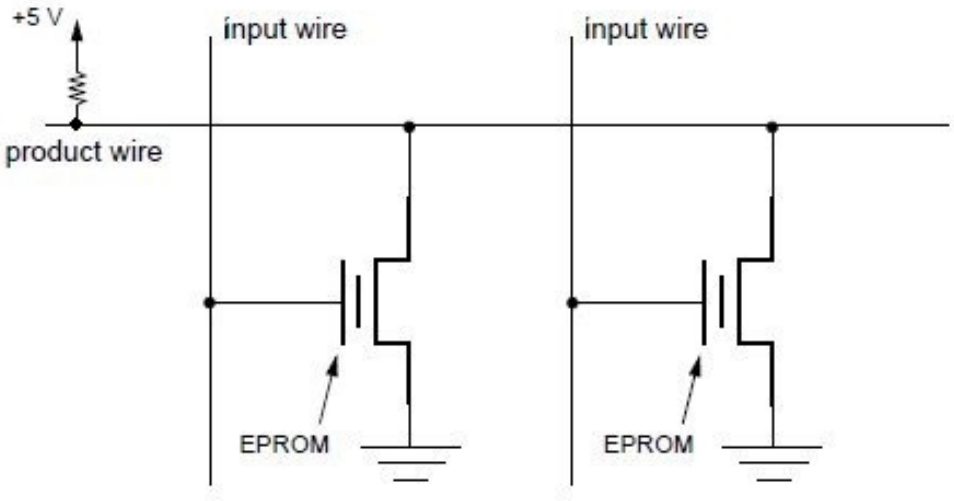


Figure 2.2: EFROM programmable switches (Brown and Rose, 1996).

2.1.2 FPGA Architectures

An FPGA consists of an array of configurable logic blocks, interconnect resources and I/O blocks as illustrated in Fig. 2.3. FPGA architectures can be homogeneous or heterogeneous. Homogeneous FPGAs contain only general logic blocks. Heterogeneous FPGAs contain a mixture of different types of logic blocks such as general logic, memory and DSP blocks. Many modern FPGAs are heterogeneous with dedicated logic blocks in addition to general logic blocks for efficient implementations of specific functions.

Logic blocks can be classified as fine-grain and coarse-grain. Fine-grain logic blocks have only transistors as the basic logic elements. All gates and storage elements are built from transistors. The main advantage of this kind of logic blocks is its efficient utilization of transistor resources within used blocks. However, they require a large amount of routing resources to implement a logic function, which is costly in delay and area. At the other extreme, coarse-grain logic blocks are made of different types of logic elements such as transistors, NAND gates, LUTs and multiplexers.

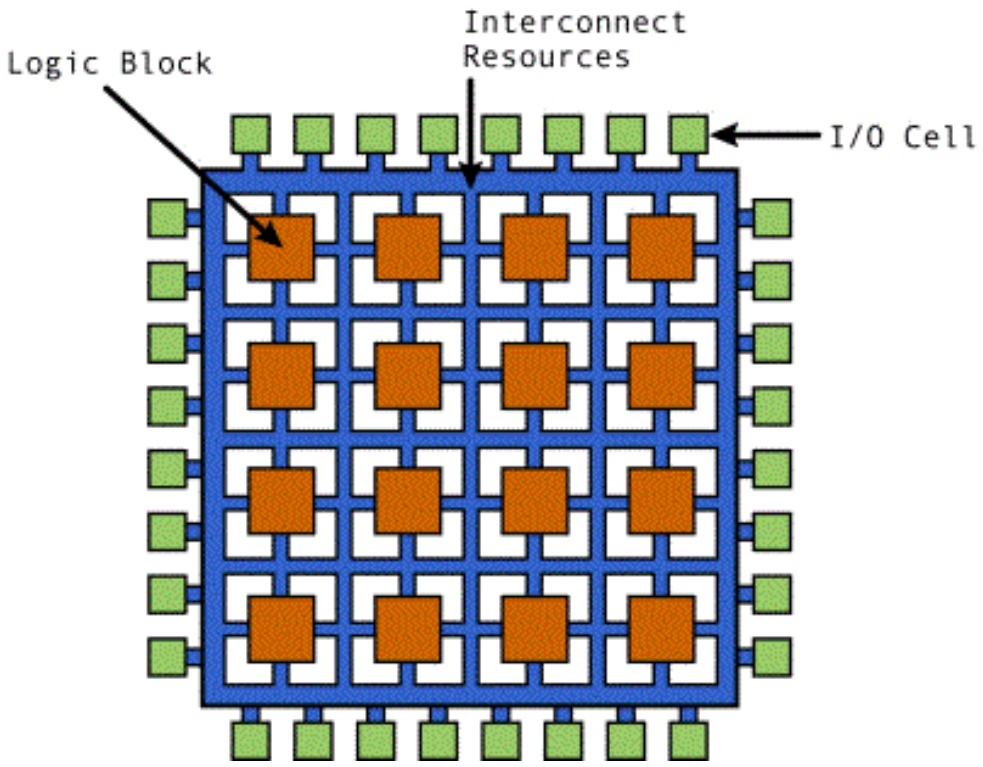


Figure 2.3: FPGA architecture.

The basic elements of logic blocks in most commercial SRAM-based FPGAs are LUTs (Lookup Tables). LUTs are actually small memories that can be used to realize combinational logic functions. A n-input LUT refers to a memory of 2^n elements, which is capable of implementing any logic function of n Boolean variables. In addition to LUTs, a logic block can have other elements such as registers, adders or multiplexers. The logic block architectures are distinct among different FPGA device series and FPGA providers. For example, an Altera Stratix-II logic block consists of an 8-input LUTs, 2 full adders, 2 multiplexers and 2 registers as shown in Fig. 2.4 (Altera, 2006). A Xilinx Virtex-5 logic block contains 2 slices, each has a 6-input LUT, some carry logic, 2 multiplexers and 1 register as illustrated in Fig. 2.5 (Borisov and Kukenska, 2009).

Interconnections of logic blocks are provided by programmable routing resources that consist of wire segments and programmable switches. FPGA routing architectures can be classified as hierarchical (Aggarwal and Lewis, 1998; Tsu *et al.*, 1999) and island-style architectures (Betz *et al.*, 1999; Baig and Lee, 2012; Ayorinde *et al.*, 2015). In the hierarchical routing architecture, connections between logic blocks are divided into groups

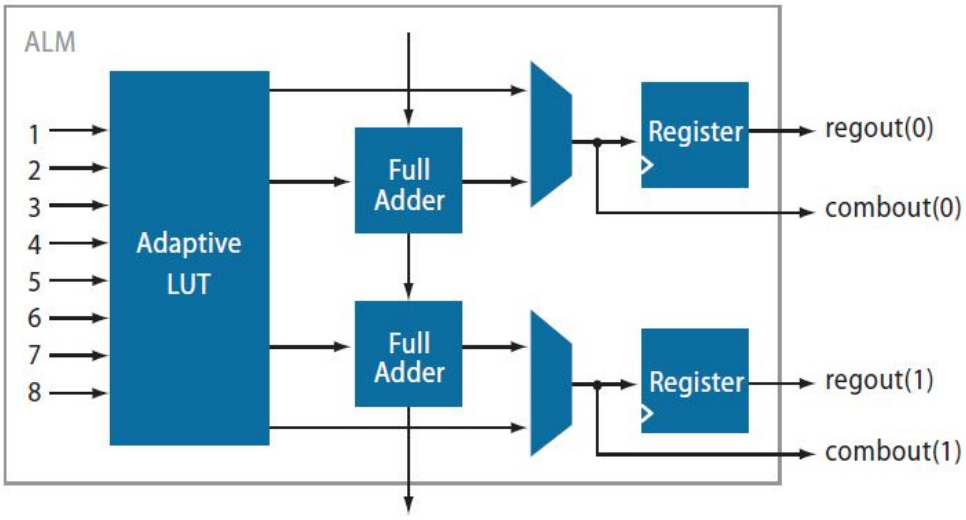


Figure 2.4: Altera Stratix II logic block architecture (Altera, 2006).

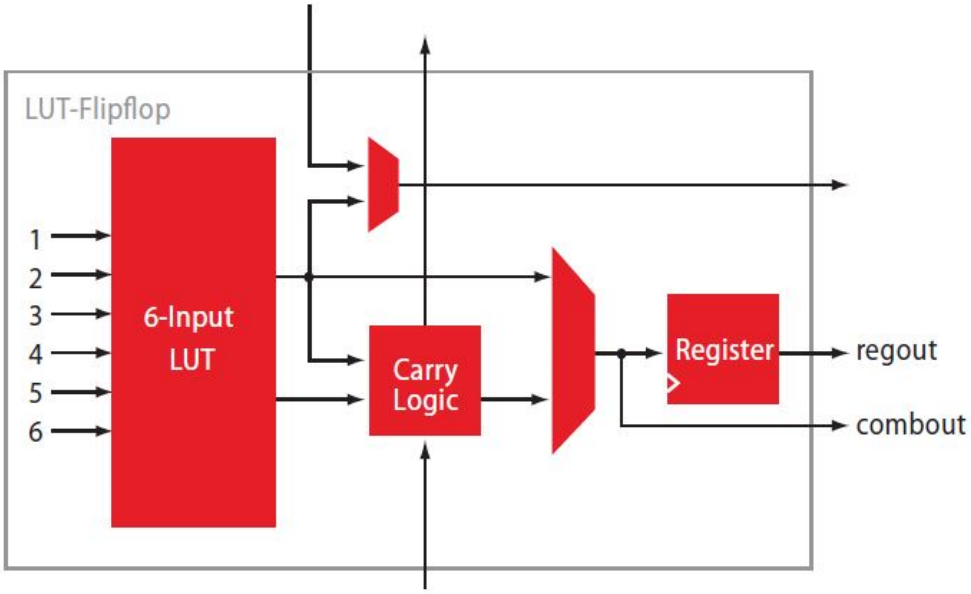


Figure 2.5: Xilinx Virtex 5 logic slice architecture (Borisov and Kukenska, 2009).

as shown in Fig. 2.6. Connections within a group use wire segments. This is the lowest level of the routing hierarchy. Connections between groups are traversed one or several levels.

In the island-style architecture, routing resources are evenly distributed throughout a 2-D mesh of logic blocks as shown in Fig. 2.7. There are vertical and horizontal routing

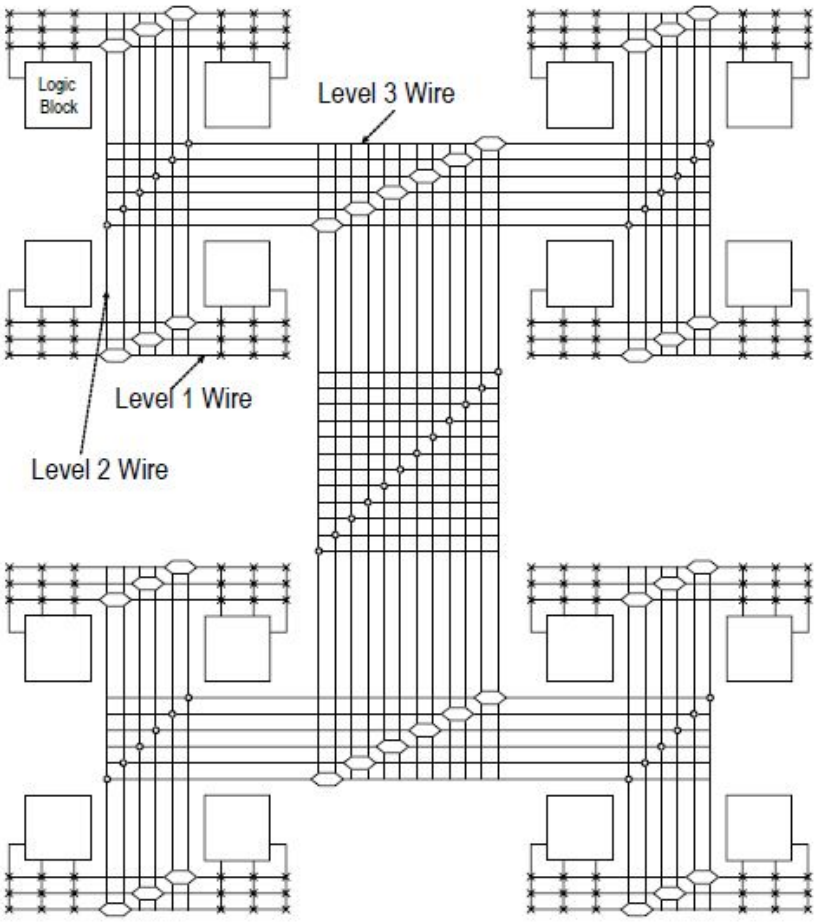


Figure 2.6: Hierarchical routing architecture (Kuon *et al.*, 2008).

channels on four sides of the logic blocks. Connections between the wire segments at an intersections of a horizontal channel and a vertical channel are formed by a switch block. Each channel has a set of wire segments of different lengths to provide appropriate length for different connections. The island-style routing architecture is used in most of commercial SRAM-based FPGA architectures (Kuon *et al.*, 2008).

2.2 FPGA Design Flow

An FPGA design flow is as shown in Fig. 2.8 (Xilinx, 2008). It usually comprises the following stages:

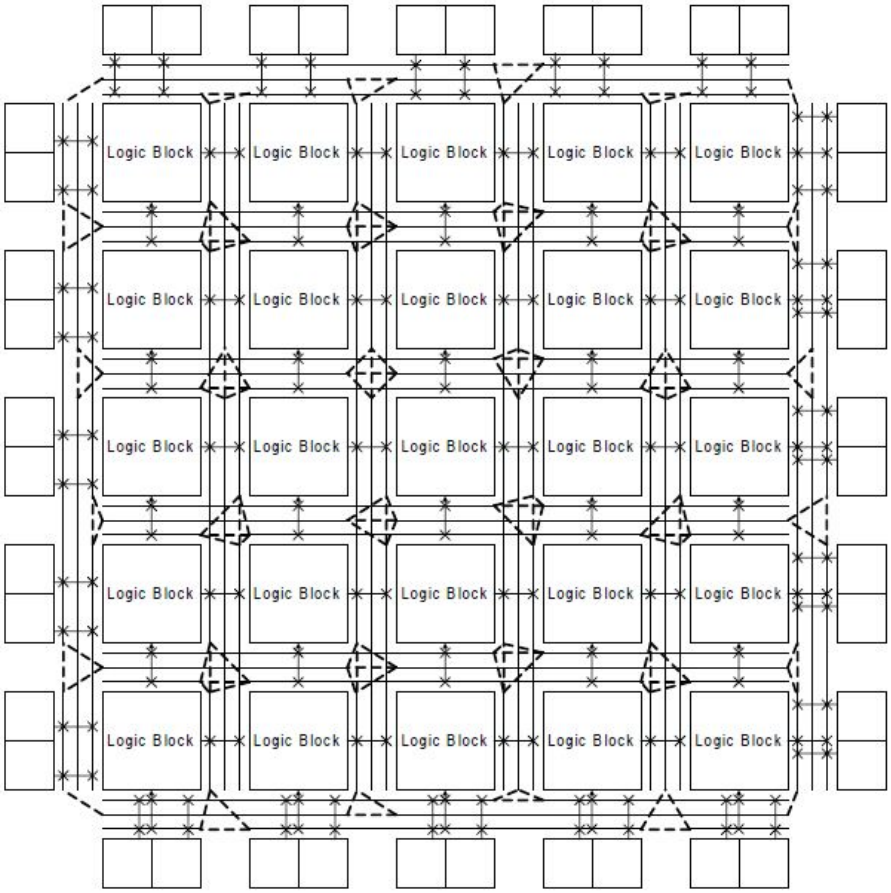


Figure 2.7: Island-style routing architecture(Kuon *et al.*, 2008).

1. Design entry: At this stage, a design is described using a schematic or a hardware description language (HDL). HDL is preferred for most contemporary designs.

2. Behavior simulation: This stage verifies the correctness of the design entry by comparing the outputs of the design model and the behavioral model.

3. Design synthesis: This stage translates the design entry into a so-called netlist. The synthesis process is performed by a software called a synthesizer. It can reveal some errors and problems that cannot be found using the behavior simulation.

4. Design implementation: At this stage, a synthesized netlist and its constraints are firstly combined and translated into a logic design file. Then, the logic defined in this file is mapped into FPGA logic blocks. Finally, the place and route (PAR) process places the sub blocks from the map process into logic blocks on the targeted

FPGA and connects these logic blocks. Verification can be done at different sub-stages of the implementation process. Functional simulation can be performed after the translate process to verify the logic operation of the circuit. Static timing analysis can be done after the map or PAR processes, which provides a comprehensive timing summary of the design. Timing simulation can be done after the PAR process to verify the functionality in conjunction with timing constraints of the design.

5. Device programming: At this stage, a bitstream of the design is generated and loaded on the FPGA. The design functionality can be now verified on the FPGA.

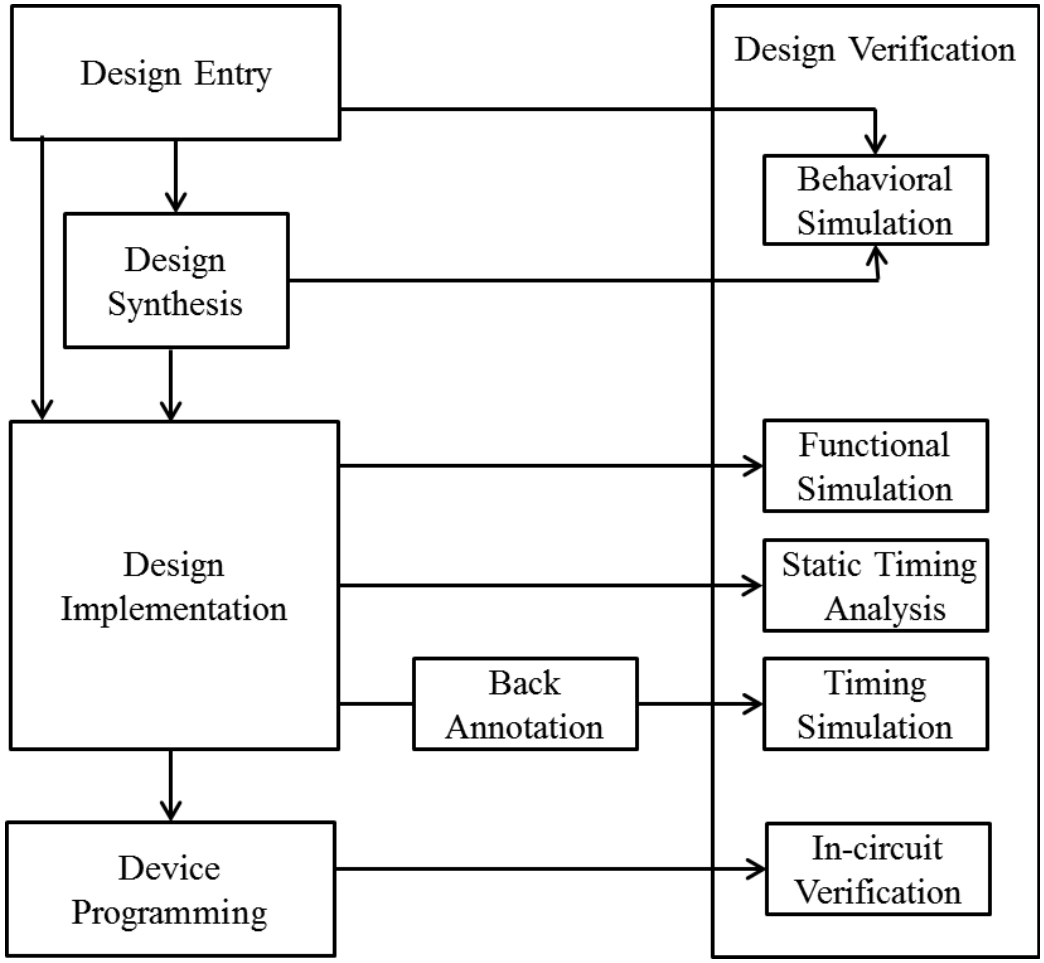


Figure 2.8: FPGA design flow (Xilinx, 2008).

2.3 On-chip Interconnect Architectures

2.3.1 Buses

Buses are commonly used to connect modules in a system-on-chip. A bus architecture is defined by its architectural and physical characteristics that lead to different implementations. The following overview of bus architectures draws on the review published by (Pasricha and Dutt, 2008).

Bus Architectures

A bus-based communication architecture provides one or more shared channels among modules attached to it. Modules that initiate read and write data transfers on the bus are referred to as masters. Modules that simply respond to the requests from the masters are referred to as slaves. Some modules can act as both masters and slaves. The bus channel consists of wires that can be classified into 3 categories: address, data and control signals. Address signals are used to transmit the address of the destination of a data transfer on the bus. Data signals are used for data transferred on the bus. Control signals are used for control and arbitration.

In addition to the bus channel, a bus-based communication architecture consists of some other components such as arbiters, decoders and bridges. An arbiter determines which master can have access to the bus when multiple masters want to access the bus at the same time. Some common arbitration schemes are static priority, round-robin and time division multiple access (TDMA). In the static priority arbitration scheme, priorities of the masters on the bus are fixed. This scheme is simple to implement and can provide high performance. However, it can lead to starvation as masters with lower priorities can never get access to the bus if there are frequent accesses by masters with higher priorities. At another extreme, the round-robin arbitration approach grants access to the bus to all masters in a circular manner. This ensures that every master can have access to the bus, hence there is no starvation in the system. However, this approach suffers a significant drawback in that critical data transfers may have to wait a long time to be proceeded.

A decoder selects the appropriate slave to receive the data by decoding the address signals. It can be implemented in a centralized or distributed manner as shown in Fig. 2.9 (a) and (b) respectively. In the centralized implementation, the decoder sends a select signal to the appropriate slave on the bus according to the decoded address. In the distributed approach, decoding is performed in every slave by its own decoder. The centralized approach has the advantage of easy expansion. When new modules are added to the system, only the centralized decoder needs to be adjusted. In addition, although there are fewer control wires in the distributed approach compared to the centralized approach as there is no need to select signals from the centralized decoder to every slaves, the distributed approach usually consumes more logic and area due to the decoding hardware duplications in every slave.

Bus Topologies

Bus-based interconnect architectures can have different topologies. Common bus topologies are shared buses, hierarchical buses and rings. A shared bus as shown in Fig. 2.10 (a) is the simplest scheme. All modules in the system are connected to a single shared bus. The main advantages of shared buses are their simplicity and low area cost. However, they suffer from lower bandwidth when the system size increases. A hierarchical bus as shown in Fig. 2.10 (b) is a more efficient topology. It consists of several shared buses connected by bridges. The commercial AMBA bus (ARM, 2017) is an example of a hierarchical bus. Another common bus topology is a ring topology as shown in Fig. 2.10 (c). In this scheme, modules are connected using a ring interface. Data can be transferred in a clockwise or anti-clockwise direction. Ring buses can be found in many applications such as networks processor and ATM switches (Mitić and Stojčev, 2006).

Physical Implementations

A physical shared bus can be implemented using tri-state buffers, multiplexers and AND-OR logic as shown in Fig. 2.11 (a), (b), (c) respectively. Tri-state buffer implementations of buses consume fewer wires and have smaller area; however they have major

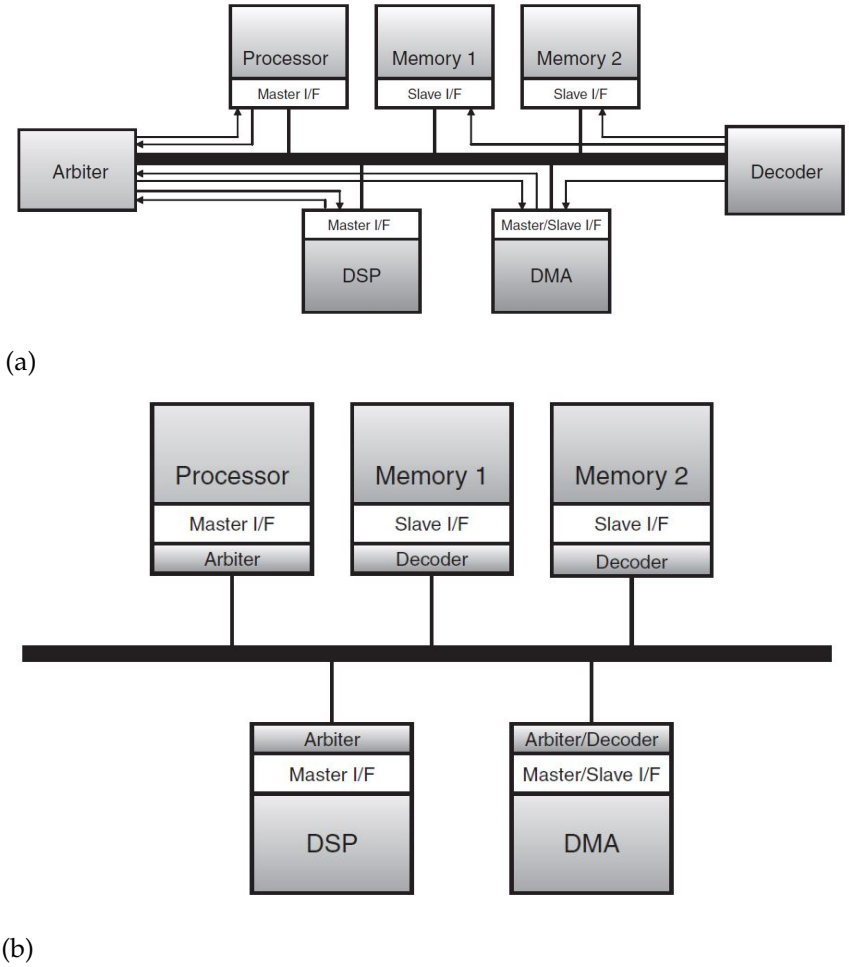


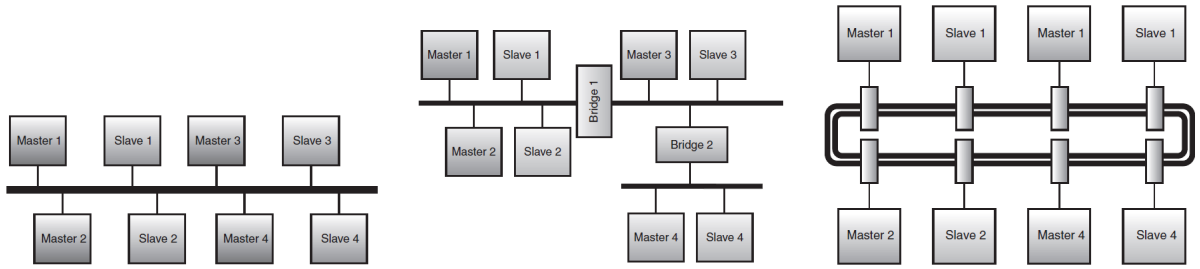
Figure 2.9: Decoder implementation strategies (Pasricha and Dutt, 2008). (a) Centralized decoding. (b) Distributed decoding.

drawbacks of high power consumption and delay, and debugging problems. Therefore, they are not commonly used in modern bus-based communication architectures compared to the other two alternatives.

Another important characteristic of buses is their clocking type. Based on the clock signal, buses can be divided into 2 categories: synchronous buses that have a clock signal as a control signal, and asynchronous buses without a clock signal in the control signals of the buses.

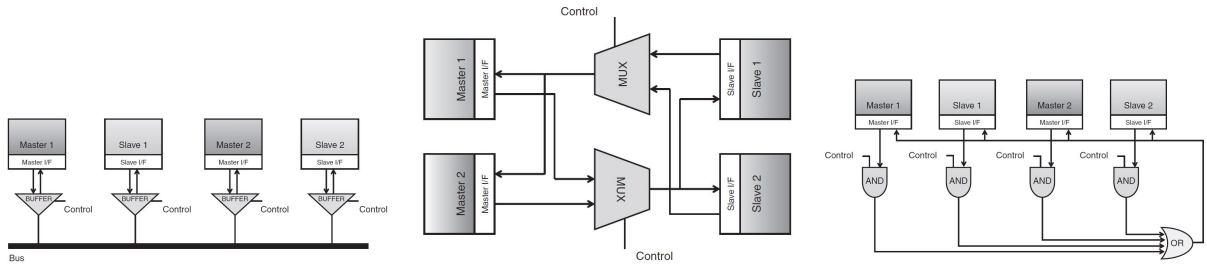
2.3.2 Networks on Chip

The advances and developments in semiconductor technologies have made it possible to integrate hundreds or thousands of cores onto a single chip that contains a very high level of integration complexity and functionality. These complex systems require a highly efficient interconnection infrastructure. Traditional bus-based communication may be insufficient to meet these requirements due to limitations in throughput and speed that are dependent on physical factors such as the length of bus and the number of devices. Networks-on-chip (NoCs) have been suggested as a promising alternative to the bus-based interconnection system. NoCs have has crystallized into a significant research domain due to their advantages of scalability, modularity and reusability (Benini and Micheli, 2002). The following introduction to NoCs draws upon reviews by (Benini and Micheli, 2002) and (Enright and Peh, 2009). A NoC can be characterized



(a) (b) (c)

Figure 2.10: Common bus topologies (Pasricha and Dutt, 2008). (a) A shared bus. (b) A hierarchical bus. (c) A ring bus.



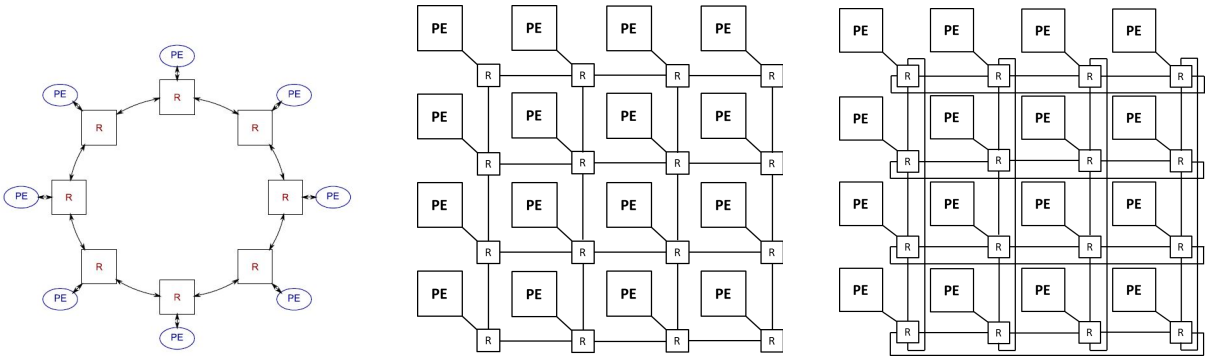
(a) (b) (c)

Figure 2.11: Physical implementation approaches of buses (Pasricha and Dutt, 2008). (a) Tri-state buffer. (b) Multiplexers. (c) AND-OR.

by its topology, routing algorithm, flow control and router microarchitecture. The basic elements in a NoC are routers, which are responsible for transferring messages in the network.

Topology

A NoC topology is the physical layout of nodes in the network. It determines connections among the nodes and the number of hops a message must traverse. Therefore, it has a significant influence on the network latency. NoC topologies can be classified as direct and indirect. In a direct topology, all routers are attached to their local processing elements (PEs). In an indirect topology, in addition to routers attached to PEs, there are intermediate routers which are not associated with any PE. These routers simply switch traffic in the network. Rings, meshes and tori are common direct NoC topologies. These are shown in Fig. 2.12 (a), (b) and (c) respectively. Butterflies and fat trees, as illustrated in Fig. 2.13 (a) and (b), are examples of indirect topologies. There are also irregular NoC topologies, which are customized for heterogeneous systems. For these systems, a customized topology often provides better performance and power efficiency. An example of a customized topology is given in Fig. 2.14.



(a) (b) (c)

Figure 2.12: Common direct NoC topologies. (a) A ring. (b) A mesh. (c) A torus.

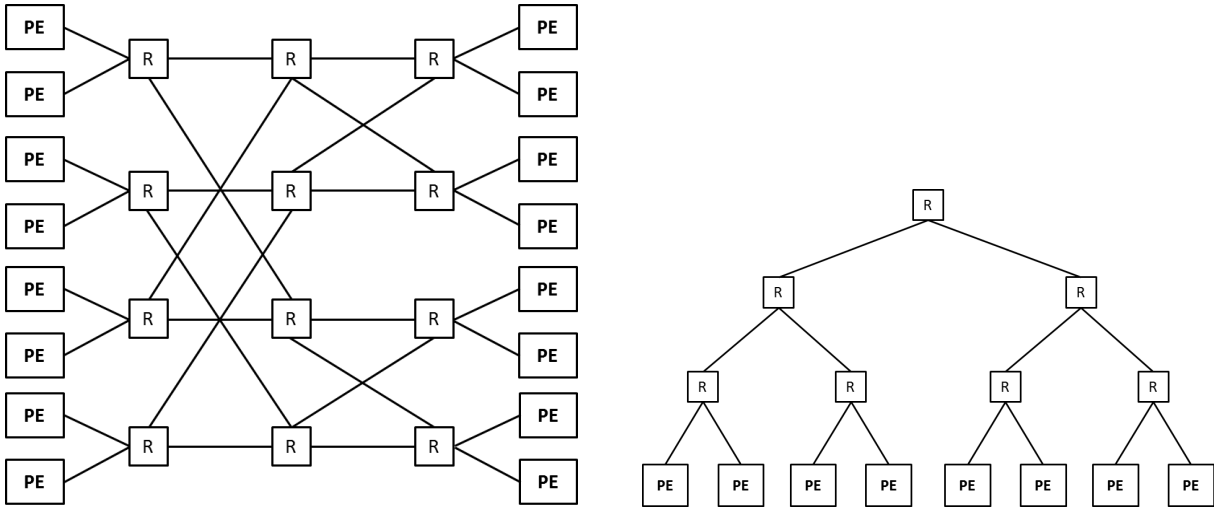


Figure 2.13: Common indirect NoC topologies. (a) A butterfly. (b) A fat tree.

Routing Algorithm

A NoC determines the path a message will traverse to reach its destination according to its routing algorithm. Routing algorithms can be divided into deterministic and adaptive. With deterministic algorithms, paths between sources and destinations are predefined without considering the current network traffic. The dimension-ordered routing algorithm is a common example of a deterministic routing algorithm. With this algorithm, messages traverse in the network dimension by dimension. They reach an ordinate matching their destinations before switching to the other dimension. For example, with the X-Y routing algorithm, messages travel horizontally through the network (along the x dimension) before traveling vertically (in the y dimension). Dimension-ordered routing is simple and deadlock-free. However, with this algorithm, there is always only one path between every source and destination pair. This eliminates path diversity in the network, which can lead to congestion, and thus lowers network throughput.

An adaptive algorithm is a more complex algorithm that takes into consideration the network congestion. When there is a congestion in the network, messages will be redirected to other paths to avoid congested links. Adaptive routing can reduce congestion

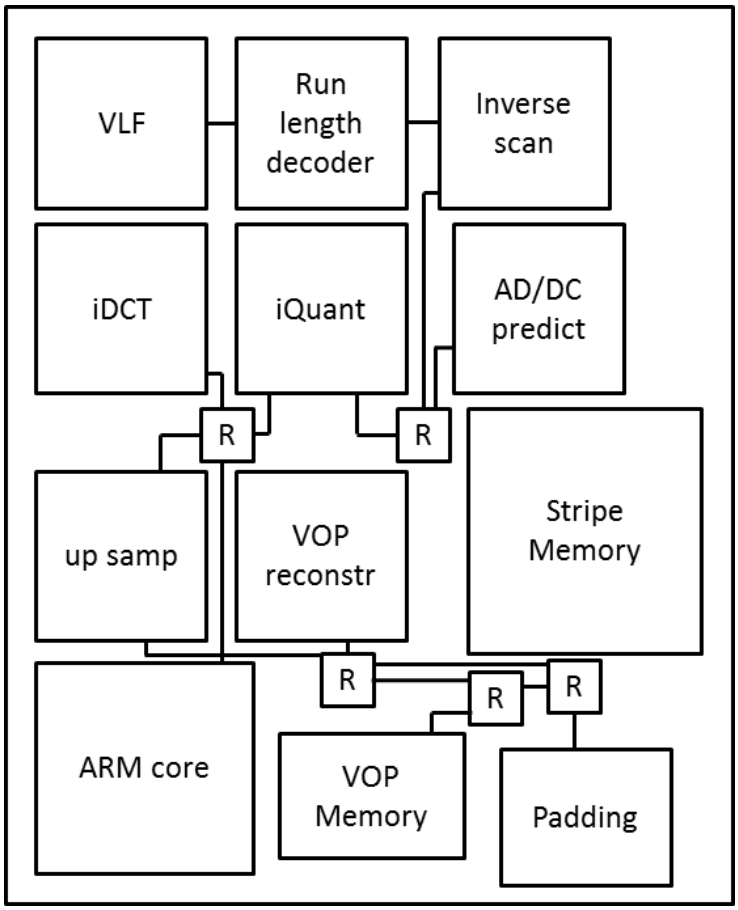


Figure 2.14: A customized topology for a video object plane decoder (Enright and Peh, 2009)

latency in the network and thus achieve higher bandwidth. However, the main challenges of deploying an adaptive routing algorithm are ensuring freedom from deadlock and preserving inter-message orders. In addition, additional circuits are required for congestion control, and this can increase routing latency and router area.

Flow Control

Flow control determines the rate messages are injected into the network. It allocates network buffers and links to the messages. Messages transferred in the network are divided into packets which are then segmented into fixed-length flow-control units, or flits. Based on the granularity of flow control units, flow control methods can classified as message-based, packet-based and flit-based flow controls.

Circuit-switching is an example of message-based flow control. In this flow control technique, a link between source and destination is set up before a message is transferred and it is maintained throughout the transfer. Circuit switching does not require buffers to hold packets at each node and can reduce latency. However, it can have a detrimental affect on bandwidth utilization as the links are idle during setup.

Packet-based flow control allocates resources to packets. Packets of a messages are interleaved on the links, thus link utilization is improved. This flow control technique requires buffering packets at each node. Store-and-forward and virtual cut-through are the two common packet-based flow control techniques. In store-and-forward flow control, each node receives an entire packet before forwarding any part of the packet to the next node. This requires large buffers and incurs long delay each each hop. Virtual cut-through can reduce delay at each hop by allowing a packet to proceed to the next node before the entire packet is received. However, it still requires buffering for an entire packet at each node.

Flit-based flow control can reduce the buffering requirements of packet-based technique as it operates at the flit level. Wormhole flow control is a common flit-based technique. This allows a flit to move on to the next node as soon as a downstream buffer is available for it. Therefore, it can reduce packet latency as well as buffering requirements.

Router Microarchitecture

A basic router microarchitecture consists of input/output ports, buffers, a route calculator, an arbiter and a crossbar. The number of input/output ports and buffer requirements at each port depend on the network topology and flow control. The route calculator determines the path for the current packet in each input port. The arbiter determines which input is selected to proceed to the next stage. Then, a crossbar matches the successful input with the desired output port. State-of-the-art routers are implemented with virtual channels. A virtual channel is basically a separate queue in the router. Multiple virtual channels share the same physical link. The implementation of virtual channels in each input port can reduce head-of-line blocking and deadlocks. However, virtual channel routers require additional resources for buffers and virtual

channel arbitration logic. Fig. 2.15 is an illustration of a virtual channel router microarchitecture with 5 input/output ports and 4 virtual channels at each input port.

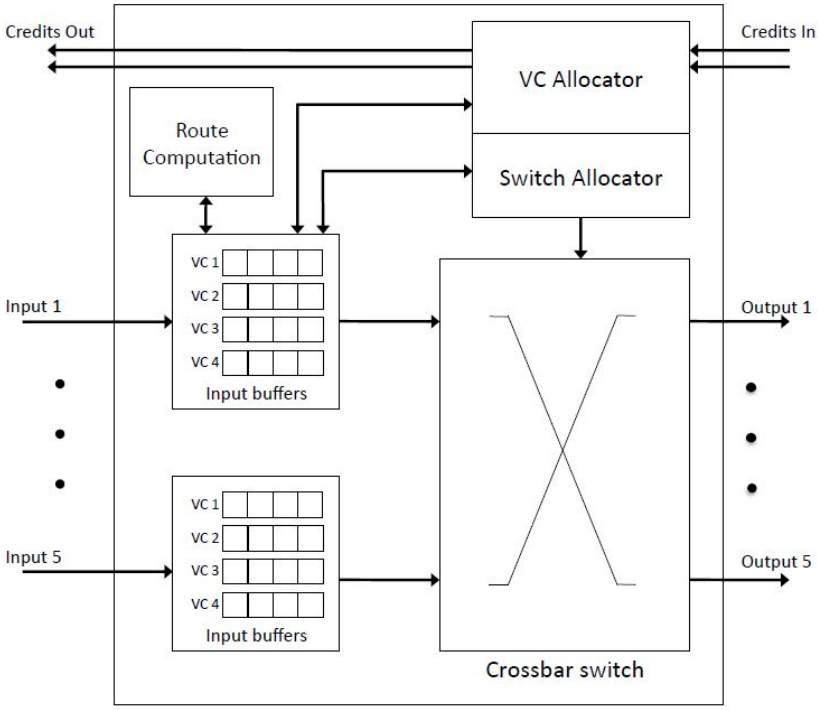


Figure 2.15: A virtual channel router microarchitecture (Enright and Peh, 2009)

2.4 On-chip Interconnect Architectures for Dynamically Partially Reconfigurable Systems

2.4.1 Buses

Bus architectures that have been proposed to support partial reconfiguration can be classified as global shared buses (Hagemeyer *et al.*, 2006; Koh and Diessel, 2006; Silva and Ferreira, 2008; Koch *et al.*, 2008b; Koester *et al.*, 2011) or segmented buses (Ahmadinia *et al.*, 2005).

Fig. 2.16 illustrates a global shared bus system. These consist of shared signals and dedicated signals. Shared signals are often used for data and address channels while dedicated signals are used for control and arbitration. In the context of dynamically

partially reconfigurable systems the shared and dedicated signals are fixedly implemented. Much of the published literature on dynamically reconfigurable global buses focuses on different approaches of implementing connection points between reconfigurable modules and the bus logic. Four different methods for distributed connection points are introduced and evaluated by (Hagemeyer *et al.*, 2006). These approaches are based on tristate wires and targeted at Xilinx Virtex-II FPGAs which allowed modules to span over the horizontal width of the devices.

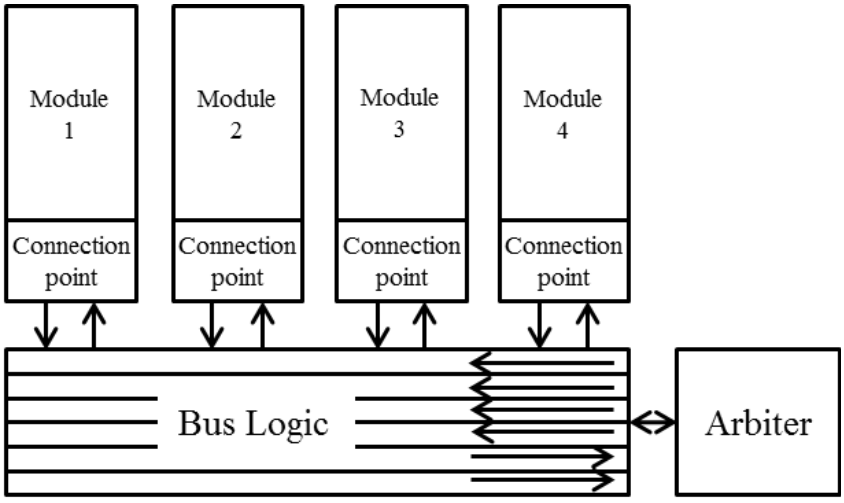


Figure 2.16: Dynamically reconfigurable global shared bus architecture.

Modern FPGAs no longer support tristate buffers and do not require modules to span the full device height. Partial modules can be placed in both horizontal and vertical directions; therefore, more flexible methods for routing connection signals are required. Linking partial modules using logic resources is an alternative to the tristate driver approach. In (Koh and Diessel, 2006) partial modules are reconfigured in vertical alignment using LUT-based bus macros. This approach allows variable-size modules to be placed in different slots but does not allow direct communications between partial modules. A reconfigurable system with partial modules arranged in two dimensions is proposed by (Silva and Ferreira, 2008). In this system, communications are performed from the left side to the right side via LUT-based bus macros. Partial reconfiguration modules are interconnected in a tiled region using embedded communication macros (Koester *et al.*, 2011). These macros are embedded in the tiles, which provides better resource utilization than other approaches.

A complete global shared bus for dynamically reconfigurable systems is proposed by (Koch *et al.*, 2008b). In this approach, logic and routing resources to implement the communication infrastructure are located in a homogeneous manner. Experiments show this technique provides high bus bandwidth and flexible module placement with only low resource overhead. A tool chain to build this bus, called Recobus-Builder, has been introduced by (Koch *et al.*, 2008a). However, this tool chain supports Xillinx Virtex II and Spartan-3 FPGAs only. Recent FPGA series are not supported.

Segmented buses are an alternative to global shared buses. They consist of multiple bus segments that build a complete bus system connecting all modules. RMBoC (Reconfigurable Multiple Buses on Chip) (Ahmadinia *et al.*, 2005) as shown in Fig. 2.17 is a segmented bus. RMBoC consists of cross-points that interface with the PEs and are linked to each other by bus segments. The RMBoC routing concept is similar to circuit switching in a NoC. When a PE wants to communicate with another, it sends a request message to its local cross-point. Cross-points determine the direction of the request and forward the request to suitable neighbor cross-points. If the destination is reached and the request is accepted, a reply message is sent to establish the channel. Otherwise, a cancel message is sent back to the source PE. When communication finishes, a destroy message is sent to release the channel. The concept and realization of a RMBoC cross-point are similar to those of a NoC router. This approach blurs the lines between NoCs and buses.

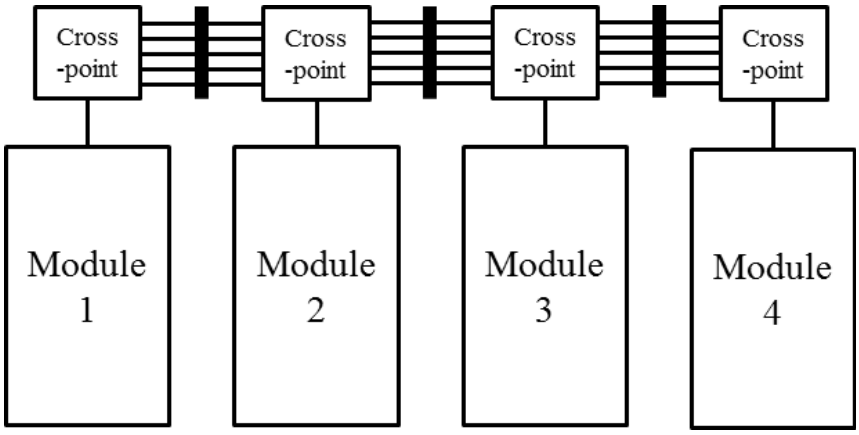


Figure 2.17: RMBoC architecture.

The body of literature on bus architectures for partially reconfigurable systems has some shortcomings. Most of these architectures are reported on synthesized results of resources costs and operating frequency on FPGA devices. Only the works by (Koch *et al.*, 2008a) and (Koester *et al.*, 2011) are mapped on example systems with some test modules. None of the papers reviewed here presents results from a realistic, fully implemented case study. This presents a challenge for the current work as there are no established benchmarks for comparison.

2.4.2 Networks on Chip

Various NoC architectures that exploit dynamic partial reconfiguration of FPGAs have been proposed (Bobda *et al.*, 2005; Jovanovic *et al.*, 2007; Pionteck *et al.*, 2008; Krasteva *et al.*, 2010; Devaux and Pillement, 2014).

DyNoC proposed by (Bobda *et al.*, 2005) is one of the first NoC architectures that supports dynamic reconfiguration. DyNoC has a mesh topology and uses the wormhole routing method. Modules consisting of several processing elements (PEs) and routers can be dynamically placed or removed on the chip at runtime. These modules communicate in the network through the routers attached to their upper right side. The routers inside modules can be reconfigured as additional resources for the modules. For routing in the network, an adaptive S-XY (Surrounding-XY) routing algorithm is used. The major drawback of DyNoC is that a module needs a ring of routers surrounding it to ensure its reachability; thus, the area ratio of PEs/routers is low.

CuNoC (Jovanovic *et al.*, 2007) is another approach for FPGA-based reconfigurable devices. When a new module is added to the network, one or several routers with the total area equal to or greater than the module are overwritten. CuNoC uses store-and-forward switching and arbitration based on the priority-to-the-right rule. It uses an adaptive routing algorithm that takes into account the current network condition. The advantage of CuNoC is its light-weight router architecture. However, the adaptive routing algorithm sometimes results in longer paths than S-XY used in DyNoC.

For the CoNoChi system (Pionteck *et al.*, 2008), as shown in Fig. 2.19, when PEs are inserted or removed, the number of routers and their positions are adapted. To do that, the architecture is divided into homogeneous partial dynamic reconfigurable logic

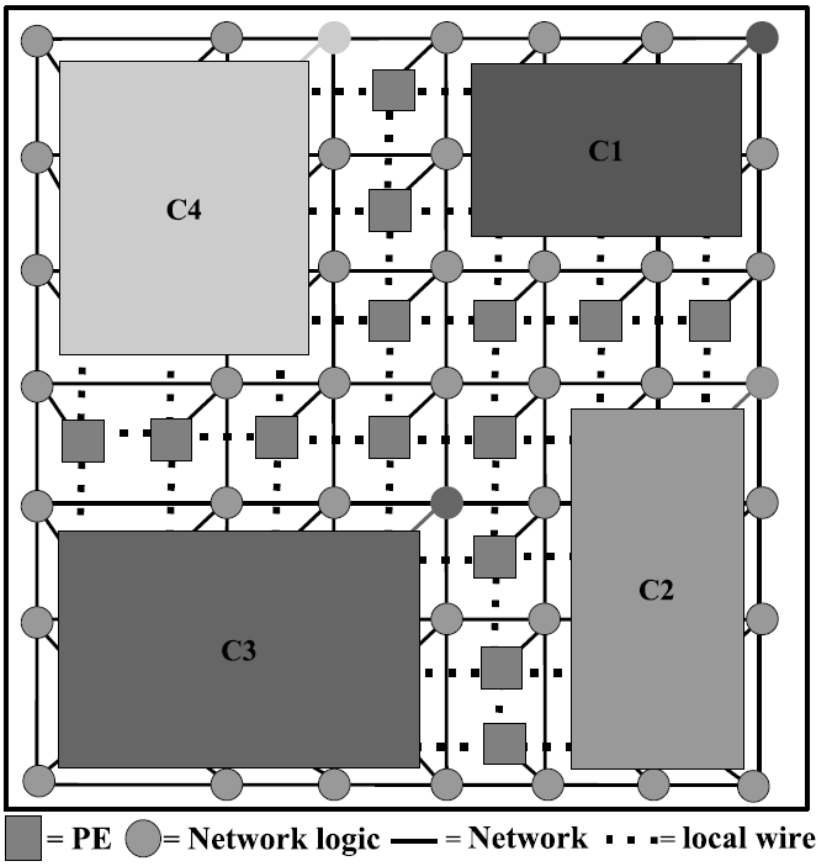


Figure 2.18: DyNoC architecture (Bobda *et al.*, 2005)

blocks that can be reconfigured as part of a hardware module, a switch or a horizontal or vertical communication link. This approach uses the minimum number of routers required for the network and hence saves area. In addition, the CoNoChi topology can change to adapt to the needs of different applications. CoNoChi uses a deterministic routing policy that is based on off-line topology. When the topology changes due to dynamic reconfiguration, a new routing table is calculated and updated to all routers in the network.

DRNoC proposed by (Krasteva *et al.*, 2010) provides the possibility of implementing different applications and their communications onto the same system. DRNoC uses modified HERMES routers (Moraes *et al.*, 2004). These routers incorporate features from both DyNoC and CoNoChi routers. They have fixed positions in the network like DyNoC routers, but can be loaded or removed like CoNoChi routers. The DRNoC

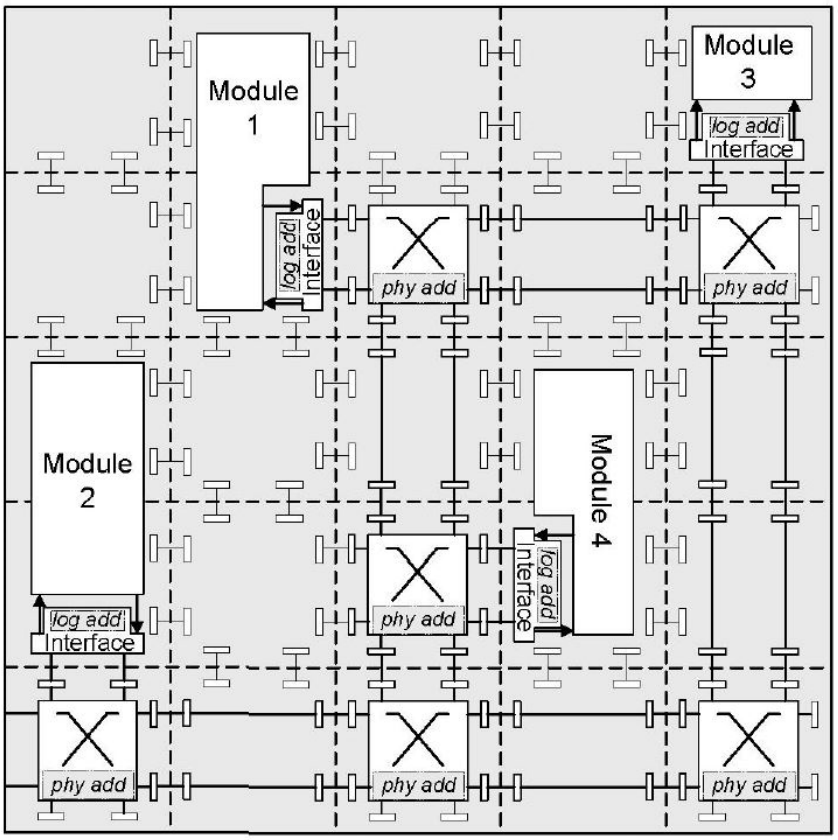


Figure 2.19: CoNoChi architecture (Pionteck *et al.*, 2008)

topology can be changed to suit a specific application. DRNoC provides a complete reconfiguration solution compared to other approaches. It is flexible but has a high hardware cost.

One of the state-of-the-art reconfigurable NoCs, OCEAN (Devaux and Pillement, 2014) as shown in Fig. 2.20, is a highly flexible NoC that supports a diversity of applications. It is a fat tree NoC with two sub-networks, one for data and one for control. The control network is responsible for routing and dynamic reconfiguration processes while the data network transmits data based on the paths defined by the control network. The use of the control sub-network can accelerate the routing process and ensure correct asynchronous operation, but it also has a significant hardware cost. Another particularity of OCEAN is that it uses its own dynamic configuration scheme and test platform, which is independent from the dynamic reconfiguration limitations of FPGA technology. Therefore, it cannot be fully realized without a custom integrated circuit.

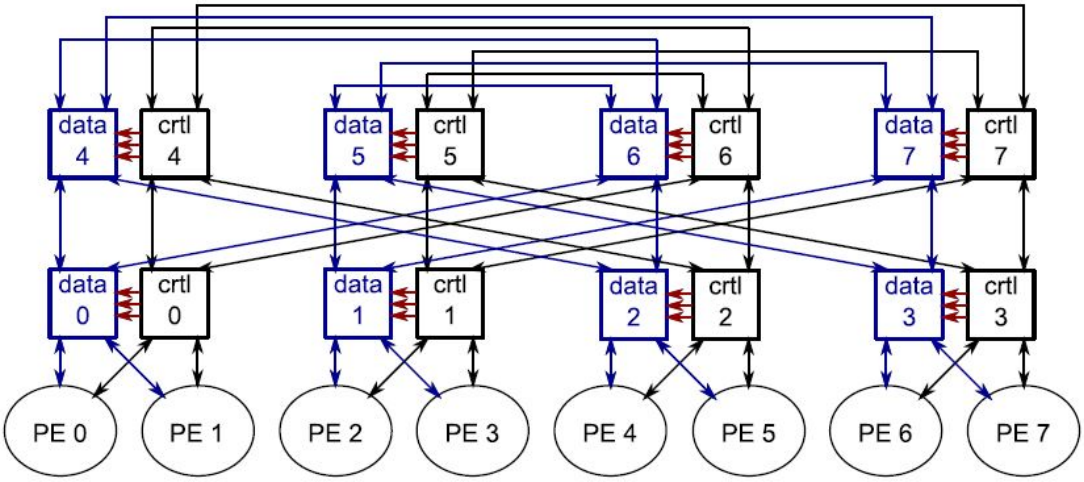


Figure 2.20: OCEAN architecture (Devaux and Pillement, 2014)

In summary, most of the studies on dynamically reconfigurable NoCs to date focus on topology, router architecture and routing strategy. However, one key consideration when implementing a dynamically reconfigurable NoC is its reconfiguration time in the context of the target application. Few of the published studies demonstrate the use of a dynamically reconfigurable NoC in a fully realized practical application. To the best of the author's knowledge, only DyNoC has been applied to a traffic light controller and CoNoChi has been demonstrated in a dynamically reconfigurable network coprocessor called DynaCORE (Albrecht *et al.*, 2006). However, they have not been fully realized and validated. It is necessary to use these NoCs in realistic applications to verify their practicality and performance, to demonstrate their potential, and to identify their limitations.

2.5 Conslusion

This chapter has provided an overview of FPGAs and a comprehensive review of different NoC and bus interconnect architectures for on-chip systems in general and more specially for FPGA-based dynamically partially reconfigurable systems. Commercial FPGAs differ in their underlying programming technologies and their architectures. SRAM-based FPGAs are the main platform that is used for experiments and evaluation in this research due to their dominance in modern commercial FPGAs.

Most of the studies on NoCs and buses to date focus on physical implementation and routing strategy. Few of the published studies demonstrate this in a fully realized practical application. In addition, little research has been carried out to examine and compare benefits and limitations of different communication architectures in partially reconfigurable systems as well as in general FPGA-based designs. These problems will be successively addressed in the next three chapters of this thesis.

Chapter 3

Interconnect Architectures for FFT Implementations

THIS chapter presents the implementations of FFT systems using NoC and bus communications on FPGAs. Hardware design of the FFT is discussed in details. Bus and NoC architectures customized for the FFT systems are implemented and evaluated in terms of area, power consumption, and performance. The proposed FFT systems are also compared with other hardware implementations as well as software implementations on CPUs.

3.1 FFT Algorithm and Hardware Implementation

The FFT is chosen because it is widely used in a diverse variety of applications in engineering, science and mathematics (Brigham, 1988). It is commonly implemented using FPGAs, which can exploit parallel hardware to achieve power-efficient, high-speed performance. In addition, different FFT hardware architectures provide the opportunity to evaluate the performance of different interconnection architectures.

3.1.1 FFT Algorithm

The Fast Fourier Transform, transforms a vector of N data samples x_i, $i = 0, ..., N-1$ into a vector of Discrete Fourier Transform coefficients X_i, $i = 0, ..., N-1$. The basic formula of the N-point DFT is:

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn} \tag{3.1}$$

W_N^{kn} are so-called twiddle phase factors defined as:

$$W_N^{kn} = e^{\frac{-i2\pi kn}{N}} \tag{3.2}$$

The Cooley-Tukey algorithm is the most commonly-used FFT algorithm. It recursively divides the FFT into smaller FFTs. The smallest unit of operation is called a butterfly. Different butterfly architectures are possible according to the radix chosen for the FFT. Radix-2, the best known of the FFT Cooley-Tukey algorithms, divides the FFT by two at each stage. The operation of a radix-2 butterfly consists of a complex multiplier and two complex adders as shown in Fig. 3.1. This approach leads to the smallest butterfly units; hence it provides simplicity and flexibility. Higher radices reduce the number of operations but increase the complexity of the hardware. The diagram of the FFT Cooley-Tukey algorithm can be constructed as decimation in frequency or decimation in time as shown in Fig. 3.2 (a) and (b) respectively.

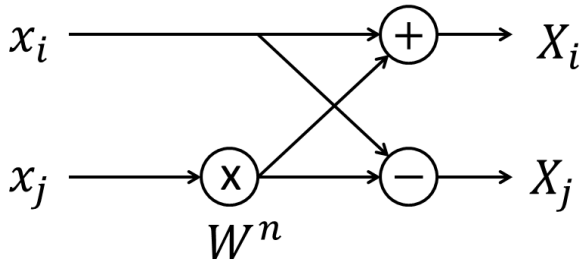
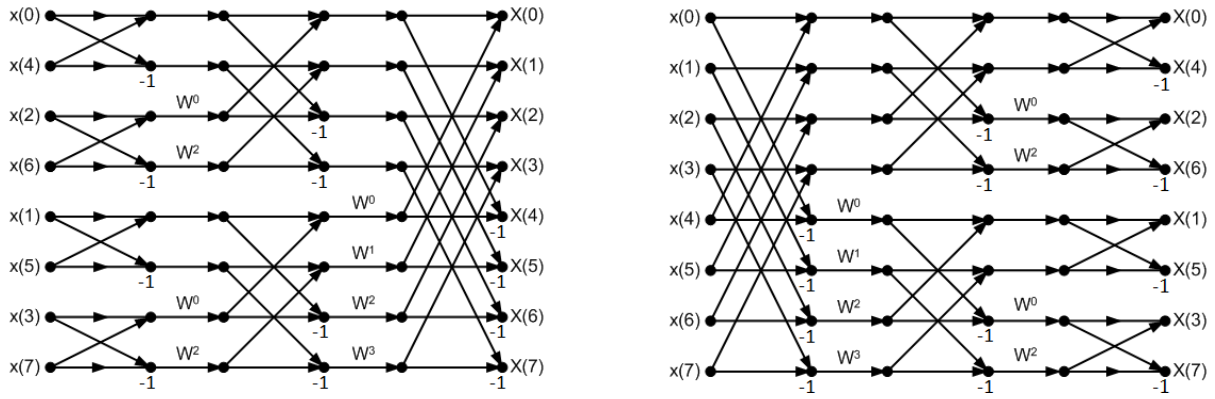


Figure 3.1: Radix-2 butterfly datapath.



(a) (b)

Figure 3.2: Two variations of radix-2 8-point FFT. (a) Decimation in frequency. (b) Decimation in time.

3.1.2 FFT Hardware Implementation

Many different hardware architectures for the FFT have been proposed. These include the parallel architecture (Palmer and Nelson, 2004), pipelined architecture (He and Torkelson, 1998; Jung *et al.*, 2003; Garrido *et al.*, 2013; Chang and Parhi, 2003), and combined parallel-pipelined architecture (Ayinala and Parhi, 2012; You and Wong, 1993).

In the pipelined design, one butterfly unit is used at each stage of the FFT. The first unit reads data from the memory, performs the butterfly operation and outputs the results to the following unit. The next unit has a memory storage buffer able to store inputs until its first two corresponding input elements are available. In this way, data is passed from the first unit to the last unit, from which results are written back to the memory. Fig. 3.3 is an illustration of a pipelined architecture.

The major advantages of the pipelined architecture are low external memory bandwidth and on-chip memory storage. However, it has a significant disadvantage. The utilization of butterfly units is relatively low since the real computation time of each unit is much lower than the total computation time. Many units stay idle at initialization and after they complete their computation. Some units even sit idle all the time when the number of FFT stages is smaller than the implemented stages. When the number of FFT stages is not a multiple of the number of implemented stages, some units are not used at the final iteration. The low utilization of the pipelined design results in its high latency. This is especially true when the number of stages in the FFT is greater than the number of hardware stages. In this case, there is a significant delay from the last unit to the first unit at the next iteration.

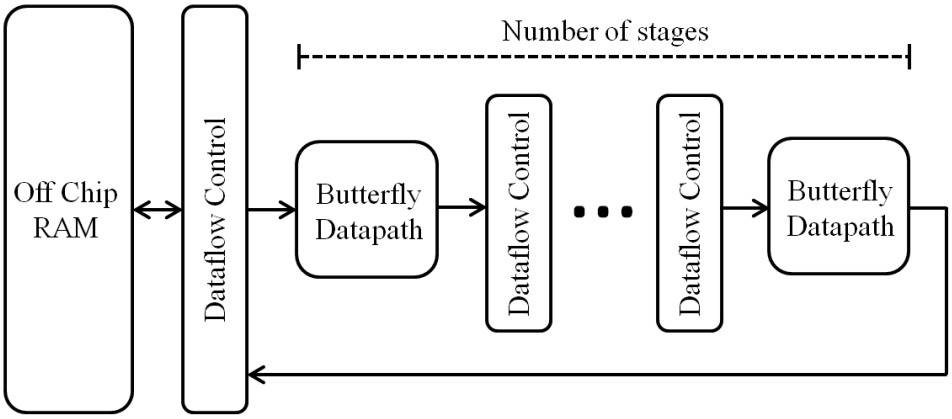


Figure 3.3: Pipelined FFT architecture (Hemmert and Underwood, 2005).

At another extreme, the parallel design exploits the data parallelism within a stage. At each stage, the input data set is equally divided among butterfly units. After finishing a stage, each unit stores the intermediate results in its memory storage or sends them to another unit according to the data scheduling of the next iteration. Fig. 3.4 shows an example of a parallel architecture with a parallelism of 4.

The main advantage of the parallel architecture is its efficient hardware utilization. All butterfly units are fully used at all stages of the FFT; therefore the number of computation cycles of the parallel design is smaller than that of the pipelined design with the same number of butterfly units. However, the memory bandwidth requirement to ensure the parallel operation of all units is much greater. Another disadvantage is that its

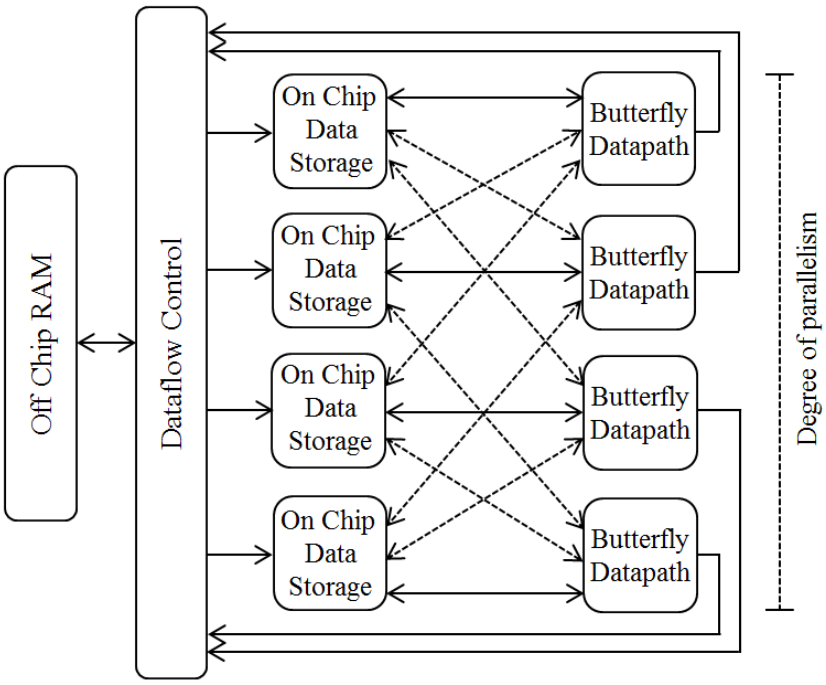


Figure 3.4: Parallel FFT architecture (Hemmert and Underwood, 2005).

routing complexity increases with the number of hardware butterfly units. With P parallel units, there are $log_2 P$ stages in which each unit needs to communicate with other units. This significantly affects the design performance. This is the main reason why the parallel architecture is not as commonly used as the pipeline architecture although it provides smaller latency.

The parallel-pipelined design as shown in Fig. 3.5 is a combination of the parallel architecture and the pipelined architecture. Several butterfly units are dedicated at each stage instead of only one as in the pipelined architecture. With the same number of butterfly units, the parallel-pipelined architecture has lower memory bandwidth requirement and routing complexity than the parallel architecture but has the same latency as the pipelined design.

3.2 NoC-based and Bus-based FFT Implementations

In this chapter, both parallel and parallel-pipelined FFT architectures are implemented. The pipelined architecture is not used as it requires communications among adjacent

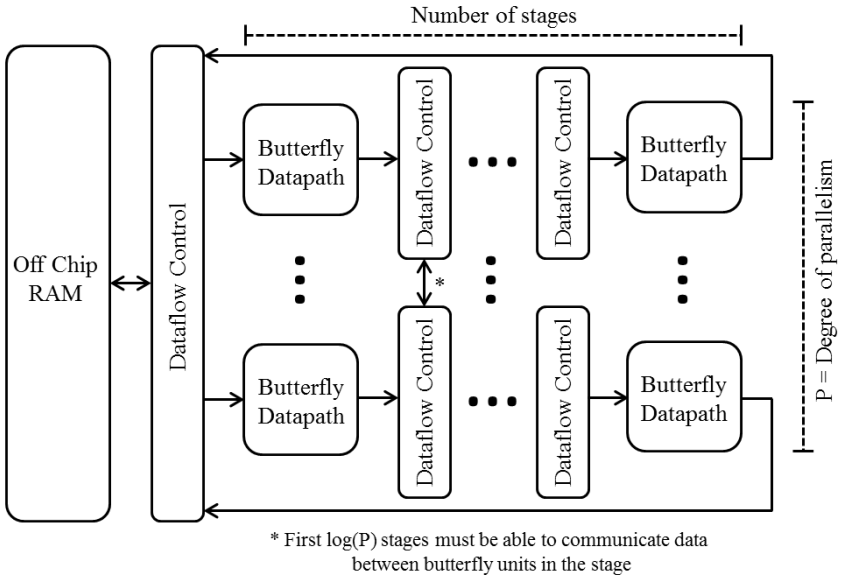


Figure 3.5: Parallel-pipelined FFT architecture (Hemmert and Underwood, 2005).

PEs only so that point-to-point connections are a better interconnect approach than NoCs and buses. For each architecture, both bus-based and NoC-based communication are evaluated for performance and power consumption comparison. The bus and NoC architectures are customized to best suit FFT communication patterns.

3.2.1 Processing Elements

A processing element (PE) performs a radix-2 butterfly operation on double-precision floating-point input data. Each PE consists of: four floating-point multipliers and six floating-point adders for the full radix-2 butterfly operation as shown in Fig. 3.6; memory storage for input data, twiddle phase factors and intermediate results; a local controller; and a NoC/bus interface. The local controller coordinates the read/write memory operation and defines which PE to communicate with at each stage. The NoC interface works as a bridge between the PE and the NoC. The architecture of a PE is shown in Fig. 3.7.

The floating-point multipliers and adders support IEEE 754 double-decision floating-point normal and abnormal numbers. They are deeply pipelined to improve speed. The floating-point multiplier is based on the design by (Jovanovic and Milutinovic, 2012).

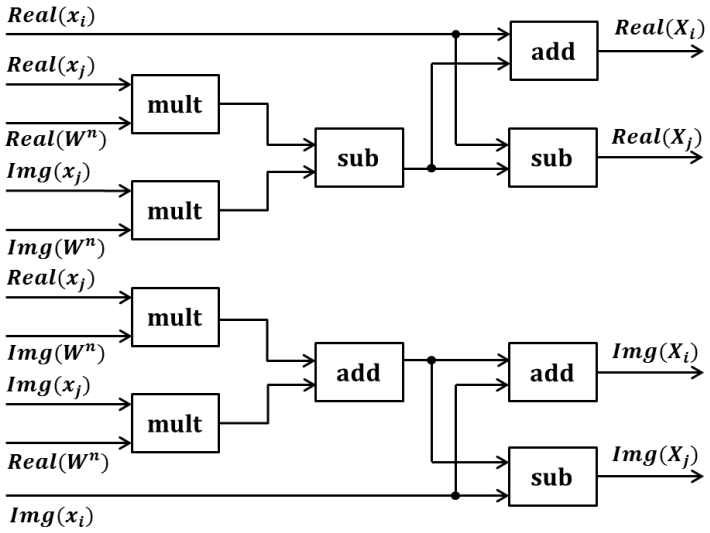


Figure 3.6: Radix-2 butterfly implementation.

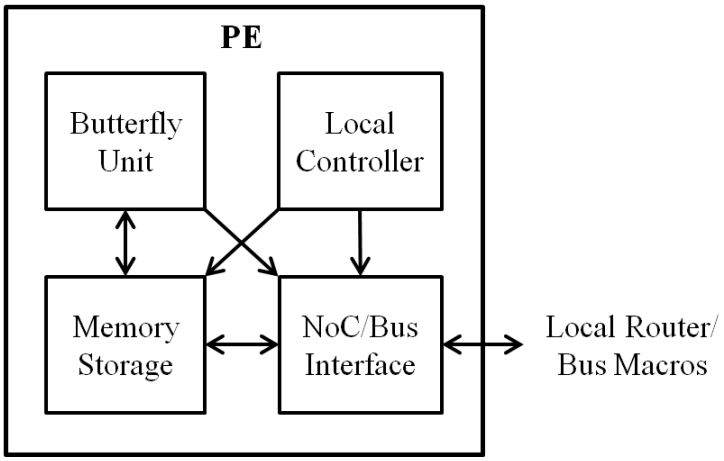


Figure 3.7: PE Architecture.

Its operation includes multiplying the mantissas, adding the exponents, calculating the result sign, normalizing and finally rounding the result according to the IEEE 754 double-decision floating-point standard. The multiplier consists of 17 pipeline stages.

The floating-point adder is also based on the design by (Jovanovic and Milutinovic, 2012). It consists of several steps as follows: ensure that the exponents of the two operands are equal by increasing the smaller one and shifting right its corresponding mantissa; add/subtract the mantissas if they have the same/opposite signs; normalize and finally round the result according to the IEEE 754 double-decision floating-point

standard. The adder is implemented with 10 pipeline stages. The subtractor can be simply implemented using an adder by inverting the sign of the second operand.

The on-chip memory storage of each PE is determined according to the number of block RAMs available on the FPGA and the number of PEs implemented. For example, a single bus-based PE synthesized for a Xilinx Virtex-7 XC7VX485T FPGA occupies 8670 register slices, 13069 LUT slices and 36 DSP48E1 slices. This accounts for 1%, 4% and 1% of the available registers, LUT and DSP slices respectively. Up to 25 PEs could be implemented on a XC7VX485T device. The largest Virtex-7 currently available, a XC7VH870T, could implement 45 PEs. In a parallel architecture, the number of PEs must be a power of two; hence only 16 PEs are implemented. There are a total of 1030 x 36Kb block RAMs; hence each PE can include up to 64 block RAMs.

The total on-chip memory required for an N-point FFT is approximately $2N$ elements (including data elements and twiddle phase elements). With elements of 128 bits for double precision complex numbers (64 bits for the real part and 64 bits for the imaginary part), the maximum FFT size the design can process is $(1030 \times 36 \times 1024)/(2 \times 128) = 148320$ points. The actual maximum size is 131072 (2^{17}) points due to the use of a radix-2 design.

3.2.2 Data Scheduling

Data scheduling is based on the decimation-in-frequency variation of the FFT Cooley-Tukey algorithm. The twiddle phase factors are pre-calculated and stored in an external ROM. The input data and twiddle phase factors are loaded from the external memory to the on-chip memory of each PE by a global controller. To ensure the parallel implementation of PEs, all input elements are transferred to the PEs before they start their operation. The number of elements transferred per cycle depends on the bandwidth of the external memories. In this implementation, one element is transferred per cycle. If the system runs at 100 MHz, the transfer of two 128-bit elements per cycle requires the memory bandwidth of 25.6 Gbps. This can be easily obtained by current memory technology.

With an N-point FFT and P-PE at the first stage, PE_i $(0 \leq i \leq P-1)$ is scheduled with the $\{x_j : j = [i : P : N-P+i]\}$. With this scheduling strategy, PEs transfer their

Table 3.1: Communications among 16 PEs in the first four stages of a parallel architecture

Stage	Communications among PEs
0	$PE_i \rightleftharpoons PE_{i+1}$ $(i = 2j, 0 \leq j \leq 7)$
1	$PE_i \rightleftharpoons PE_{i+2}$ $(i = 0,1,4,5,8,9,12,13)$
2	$PE_i \rightleftharpoons PE_{i+4}$ $(i = 0,1,2,3,8,9,10,11)$
3	$PE_i \rightleftharpoons PE_{i+8}$ $(0 \leq i \leq 7)$

results to each other in the first $\log_2 P$ stages of the parallel architecture. There is no communication among the PEs from the stage $\log_2 P$ of the parallel architecture; PEs feedback their own results to their inputs for the next stage. Fig. 3.8 is an illustration of data scheduling of a parallel architecture of 4 PEs for a 16-point FFT. Communication patterns of the first PE are also shown in the figure. In this chapter, the parallel FFT architecture is implemented with 16 PEs. Table 3.1 provides a summary of communications among PEs in the first four stages for a parallel architecture of 16 PEs.

In the parallel-pipelined architecture, there are communications among PEs in adjacent stages of the same pipelined stream in all stages. In the first $\log_2 P$ stages, there are also communications between PEs in adjacent stages of the different pipelined streams. Fig. 3.9 is an illustration of data scheduling in a parallel-pipelined architecture of 2 PEs each stage for a 16-point FFT. In this figure, PE_{ij} is the PE in the stage i of the pipelined stream j. Communication flow of the the first pipelined stream is also shown in Fig. 3.9. In this chapter, the parallel-pipelined FFT architecture is implemented with 8 stages and 2 PEs each stage.

This data scheduling requires the smallest memory storage for twiddle phase factors in a PE. At each stage, the twiddle phase factors are read in a manner so that they can be reused. Communications among PEs and between PEs and external memory use the NoC or the bus.

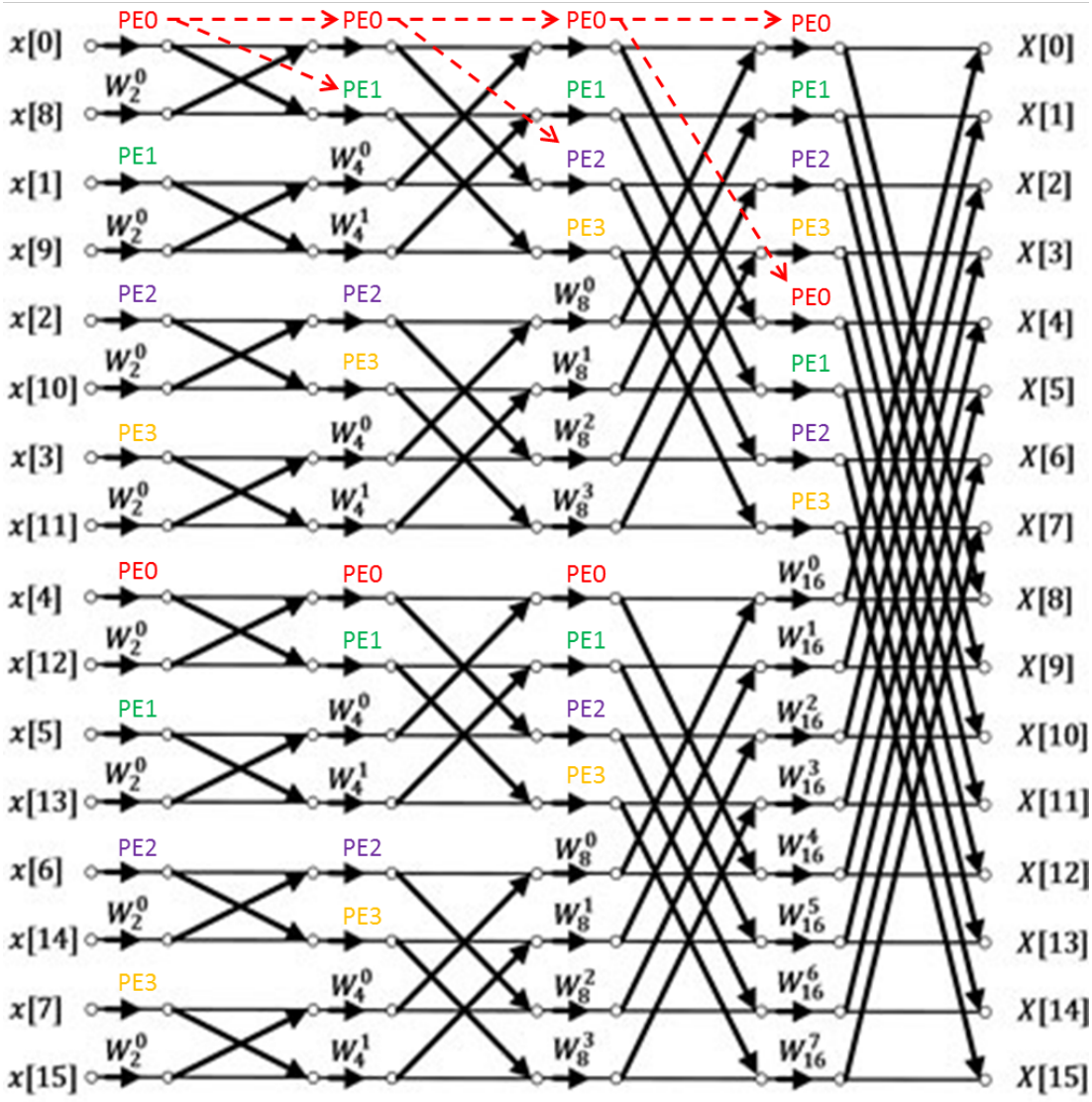


Figure 3.8: Data scheduling of a parallel architecture of 4 PEs for a 16-point FFT.

3.2.3 NoC-based Communication

The NoC customized for FFT communications is a light-weight 2-D mesh that uses small routers, circuit switching and deterministic routing. Circuit switching is used for flow control because it does not require input buffers on the NoC routers. Deterministic routing is used to maximize the throughput of the NoC.

Each router has five input/output ports: four from the four cardinal directions (North, East, South and West) and one from the local PE as shown in Fig. 3.10. There are no

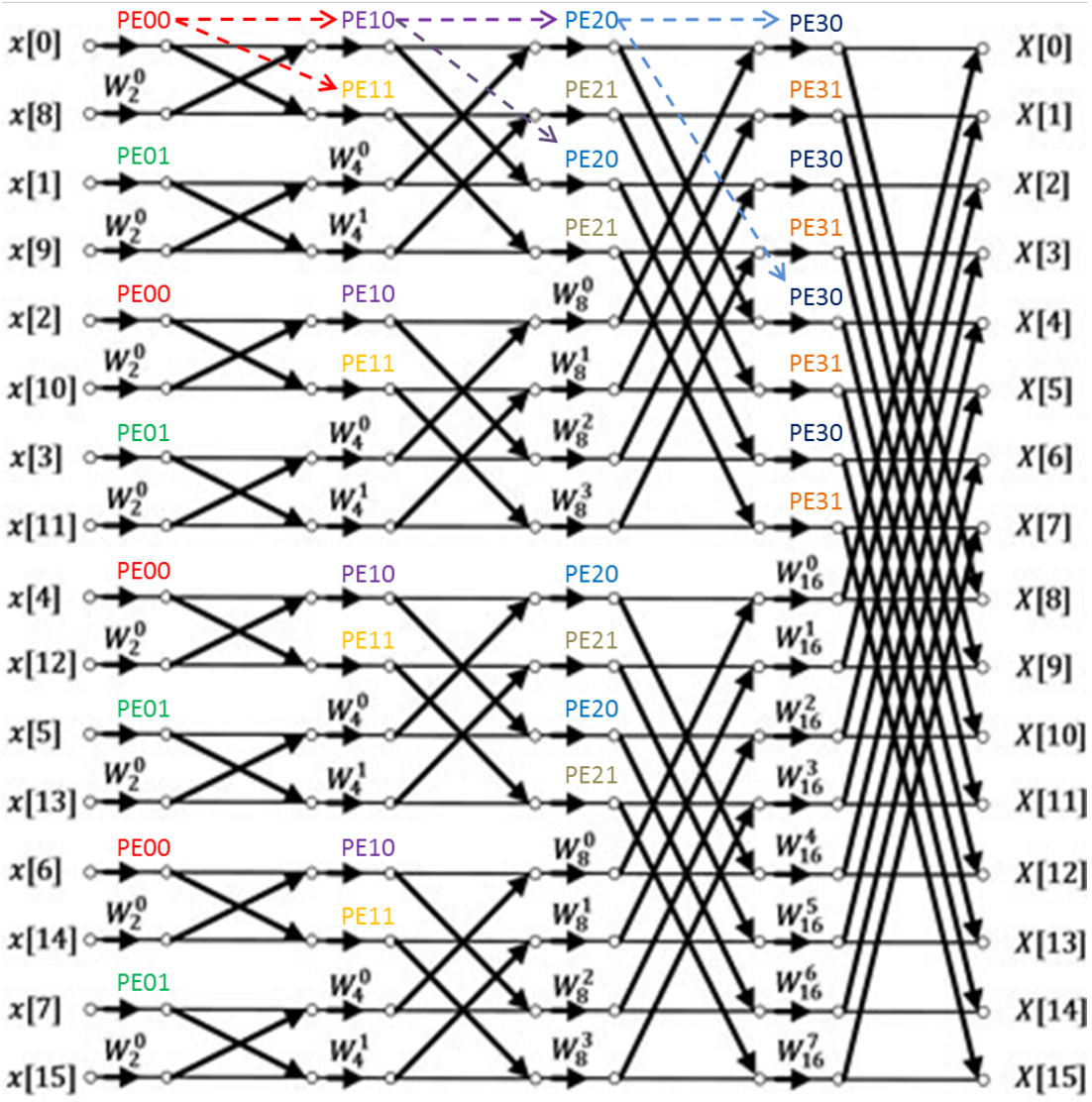


Figure 3.9: Data scheduling of a parallel-pipelined architecture of 2 PEs each stages for a 16-point FFT.

input buffers at the input ports. Each output port consists of an arbiter and a crossbar switch. The arbiter determines which input port is selected to proceed in the next stage. Then, the crossbar switch connects the successful input port to the desired output port. Each router is connected to its neighbors and local PE through a bidirectional link with 130 bits for each direction.

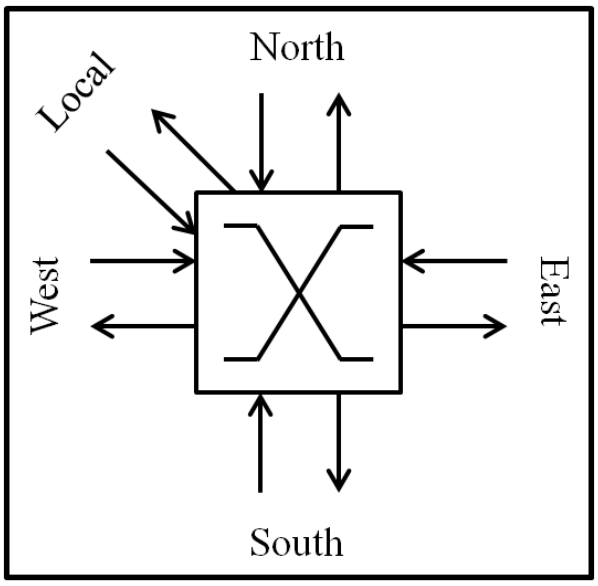


Figure 3.10: Router datapath.

In circuit switching flow control, a link between source and destination PEs is set up before a transfer is performed and is maintained throughout the transfer. Data transferred in the NoC is in 130-bit flit format. When a PE wants to communicate to another, it will assert a *Send* signal and send a set-up flit to its target PE through the NoC. If the target PE accepts the request, it will assert an *Accept* signal and a channel is constructed between the two PEs. Data is transferred and the channel is released when the last flit is received. There are 4 types of flits defined by the two most significant bits: set-up flits, data flits, tail flits and control flits.

In the deterministic routing algorithm, the route is defined by the source PE and stored in the set-up flit from the least significant bit. The routing information is a set of 2-bit directions from the source to the destination: 00 for North, 01 for South, 10 for East and 11 for West. When a router receives a set-up flit, it extracts the two least significant bits to find out the routing direction and then shifts the set-up flit two bits to the right before sending the flit to the next router. When the output direction is equal to the input direction, the flit is transferred to its attached PE.

Fig. 3.11 shows the positions of the PEs in the NoC and routes for the different stages in the parallel FFT as defined in Table 3.1. Using these routes ensures the highest throughput in the NoC as there will be no collisions in the first three stages. However, in the

last stage, only four of eight communications can be simultaneously performed. For all eight simultaneous communications, four additional bidirectional connections need to be implemented: between north ports of router-5 and router-13; north ports of router-7 and router-15; south ports of router-4 and router-12; and south ports of router-6 and router-14. The arrangement of the PEs in the parallel-pipelined FFT is as shown in Fig. 3.12. With this arrangement, communications among PEs are mostly performed by adjacent routers.

3.2.4 Bus-based Communication

The bus for an FFT implementation is a global shared bus with a 128-bit data channel and a 4-bit address channel. The bus can connect up to 16 PEs. Bus access arbitration and address decoding are implemented in a centralized manner as shown in Fig. 3.13. A simple static priority scheme is used for arbitration, in which PE_i has a higher priority than PE_{i+1}. A single channel bus is firstly implemented and compared with the NoC in terms of area, power consumption and performance. Additional channels can be added to the bus when the performance achieved by a single channel bus is not optimal for the FFT in comparison with the NoC.

3.3 Area, Power and Performance Comparison

3.3.1 Area and Power Consumption

The design has been implemented using Verilog and verified using Modelsim. Synthesis and power analysis were performed with the Xilinx ISE Design Suite, targeting a Xilinx Virtex-7 XC7VX485T FPGA. Table 3.2 shows the results. The synthesis results of a single NoC-based PE and and a single bus-based PE are quite similar. The entire bus consumes an amount of hardware resources and power even lower than a single router. It is noted that the bus here can connect up to 16 PEs. Therefore, to compare the communication overhead of the bus and the NoC, the results of a single router need to be multiplied by 16. It can be seen that the bus is better than the NoC in terms of area and power dissipation.

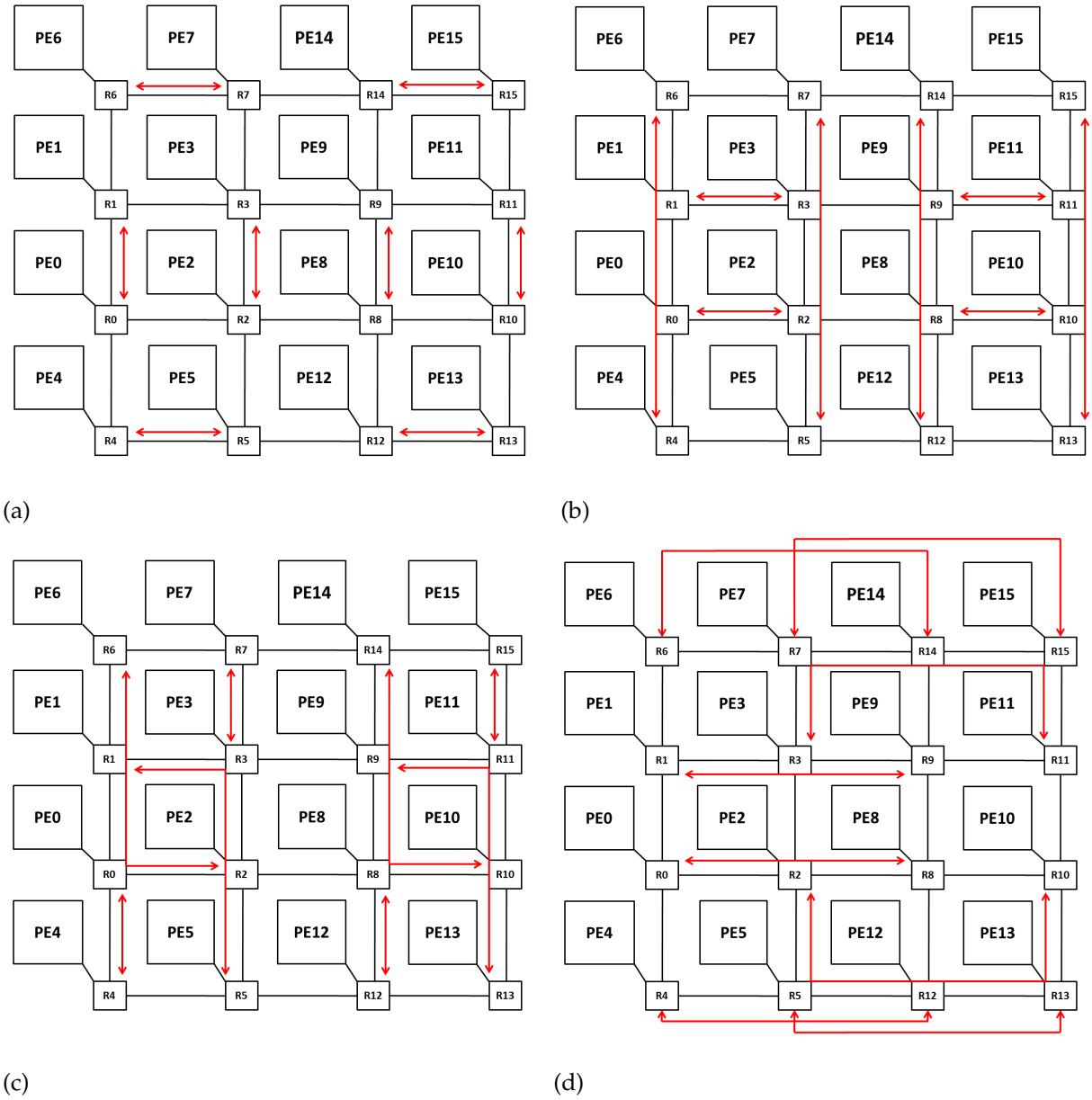


Figure 3.11: Routes defined for communications among 16 PEs in the NoC-based parallel architecture. (a) Stage 0. (b) Stage 1. (c) Stage 2. (d) Stage 3.

Fig. 3.14 and Fig. 3.15 respectively show the comparison of resource and power consumption between the NoC-based and bus-based FFT systems with different number of PEs. It can be seen that although the NoC of 16 routers consumes much more resources and power than the bus, the complete NoC-based systems only consume slightly more resources and power than the corresponding bus-based systems. This is because the

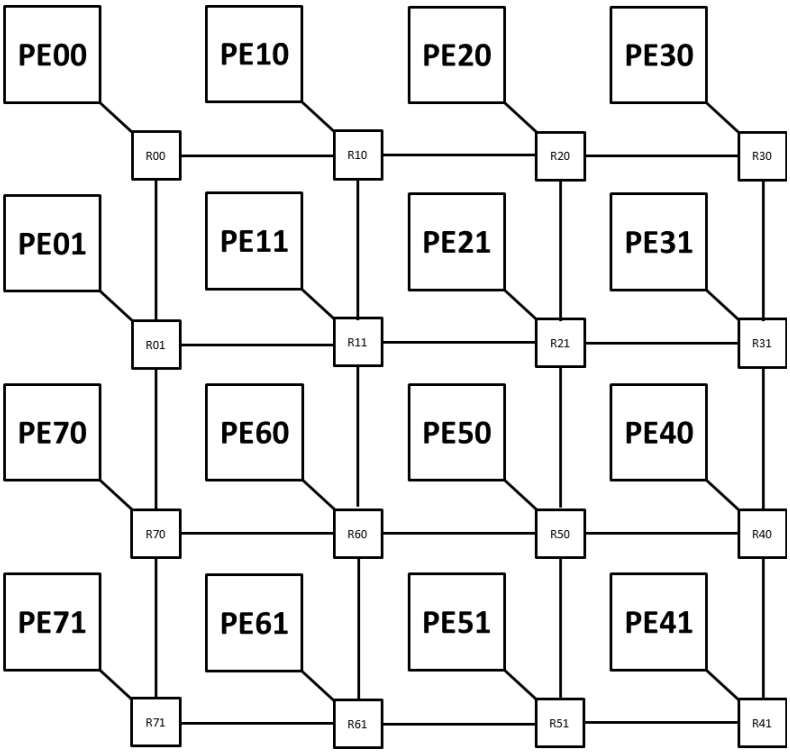


Figure 3.12: Positions of PEs in the NoC-based parallel-pipelined architecture.

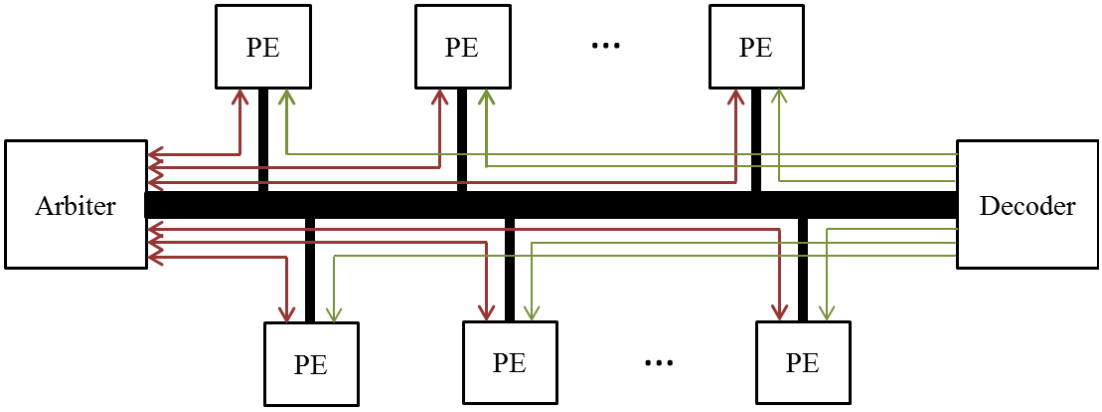


Figure 3.13: Bus architecture for FFTs.

interconnects only account for a small portion of the resources and power of the entire system. This proves the need for considering and comparing different interconnect architectures in the context of an entire system to obtain a more complete comparison.

Table 3.2: Synthesis and power analysis results

	A NoC-based PE	A Bus-based PE	A single router	Bus
Number of slice registers	8670	8670	758	178
Number of slice LUTs	13194	13069	1442	1228
Number of 36Kb BRAMs	62	62	0	0
Number of DSP48E1s Slices	36	36	0	0
Post synthesis frequency (MHz)	343	343	522	483
Estimated power at 100MHz (W)	0.839	0.837	0.34	0.29

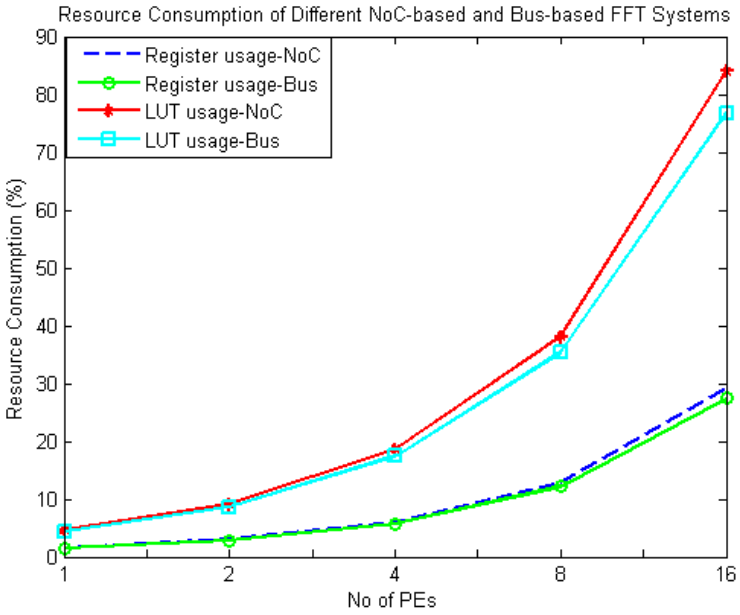


Figure 3.14: Resource consumption of different NoC-based and bus-based FFT systems.

3.3.2 Latency

NoC Latency

The NoC latency L of a transfer can be calculated as:

$$L = S + I + T \tag{3.3}$$

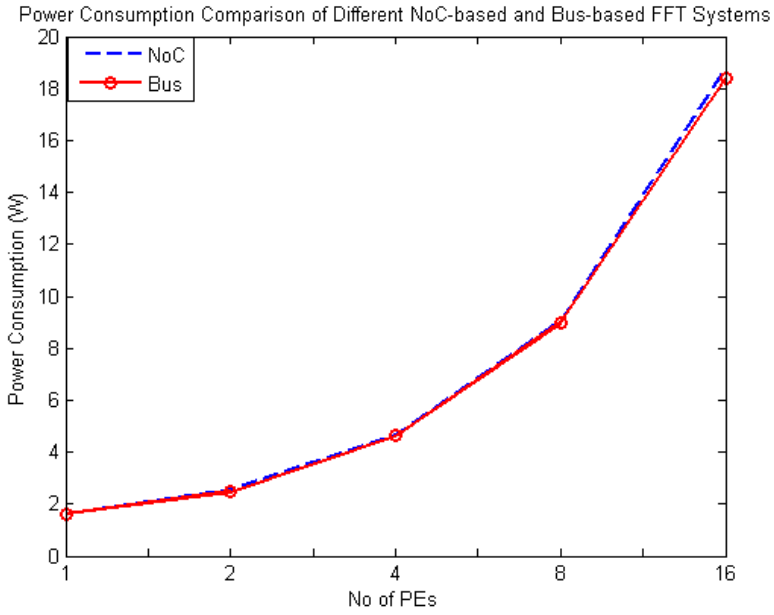


Figure 3.15: Power consumption of different NoC-based and bus-based FFT systems.

S is the link set-up time. It depends on the number of hops for the link (h). For each hop, two cycles are needed to process a Send signal and one cycle to process an Accept signal, hence $S = 3h$. The second term, I, is the initial latency, which is equivalent to the number of hops. The last term, T, is the transfer time, which is the largest contribution to the NoC latency. The transfer time is equal to the number of flits in each packet. For the parallel architecture, the transfer time is overlapped with the calculation time as a PE only transfers one output values to another PE for every operation. Therefore, the NoC latency of a transfer in the parallel architecture is only $4h$, which is insignificant compared to the total number of calculation cycles.

For the parallel-pipelined architectures, a communication between two PEs consists of two output values per operation. Only one of these is overlapped with the calculation time. Therefore, there is an additional $\frac{N}{2P}$ cycles in the NoC latency of each transfer in the parallel-pipelined architecture. These additional cycles can be saved by doubling the data width of the routers but at a higher hardware resources cost of the NoC.

Bus Latency

The bus latency of a data transfer consists of an arbitration time and a transfer time. The arbitration takes two cycles if the bus is free at the time the transfer is requested. This time is greater and depends on the priority of the requested PE when the bus is occupied. Similar to the NoC-based system, the transfer time is overlapped with the calculation time in the parallel architecture. There is also an additional $\frac{N}{2P}$ cycles in the bus latency of each transfer in the parallel-pipelined architecture.

3.3.3 Performance

In this section, performance in terms of floating-point operations per cycle (FPOPs/cycle) of different interconnection strategies are calculated and compared in each of the parallel and pipelined implementations of the FFT. Computing an N-point FFT takes $5N\log_2 N$ floating-point operations. With the number of calculation cycles (C) obtained from simulation, the performance can be calculated as $5N\log_2 N/C$ (FPOPs/cycle). It is noted that the calculation time here includes the raw calculation time and the NoC/bus latency.

Parallel FFT systems

Fig. 3.16 shows the performance of the NoC-based parallel FFT systems with different numbers of PEs and FFT sizes. It can be seen that the performance increases with an increase in the number of running PEs and FFT size. The difference in the performance of different designs also increases with larger FFT size. The highest performance of 33.5 FPOPs/cycle is achieved by the 16-PE design with the 131072-point FFT. It can be concluded that the use of a NoC for a parallel FFT implementation allows system scalability with scalable and sustainable performance.

Fig. 3.17 shows the performance of the bus-based parallel FFT systems with different numbers of PEs and FFT sizes. It can be seen that similar to the NoC-based FFT systems, the performance of the bus-based parallel FFT also increases with an increase in the number of FFT size. However, this is not the same with an increase in the number of running PEs. The performance only increases when the number of PEs increases from

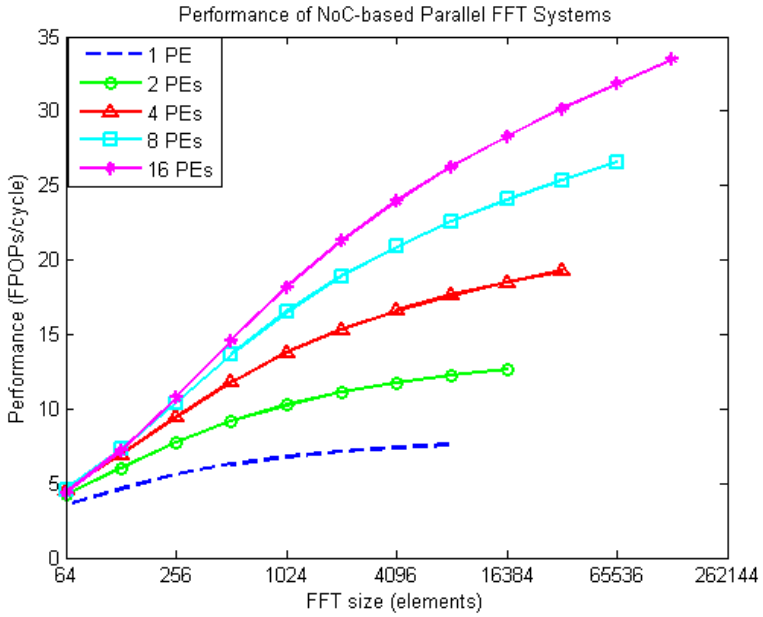


Figure 3.16: Performance of NoC-based parallel FFT systems.

1 to 4 but the performance increase from the system of 2 PEs to the system of 4 PEs is smaller than the increase from the system of 1 PE to the system of 2 PEs. Furthermore, the performance of the systems with 8 PEs is lower than that of the system of 4 PE for FFT size smaller than 1024 points. It is even higher than that of the system of 16 PEs for all of the FFT sizes except the 131072-point FFT. This is because the decrease in calculation time gained from using more PEs is smaller than the increase in the bus latency when the number of PEs increases. It can be concluded that a bus is not a good interconnect approach for a parallel FFT implementation. It suffers from low performance when the design size increases.

Fig. 3.18 shows the performance comparison of different NoC-based and bus-based parallel FFT systems. It can be seen that even the NoC-based system with 4 PEs can have higher performance than the bus-based systems with 8 and 16 PEs. In order to obtain a complete comparison of NoCs and buses for parallel FFT implementation, it is necessary to compare their performance in conjunction with their power consumption. Fig. 3.19 show the comparison of performance/power of the NoC-based and bus-based parallel FFT systems with 8 and 16 PEs. It can be seen that the NoC-based systems have much higher performance/power than the bus-based systems with the same PEs.

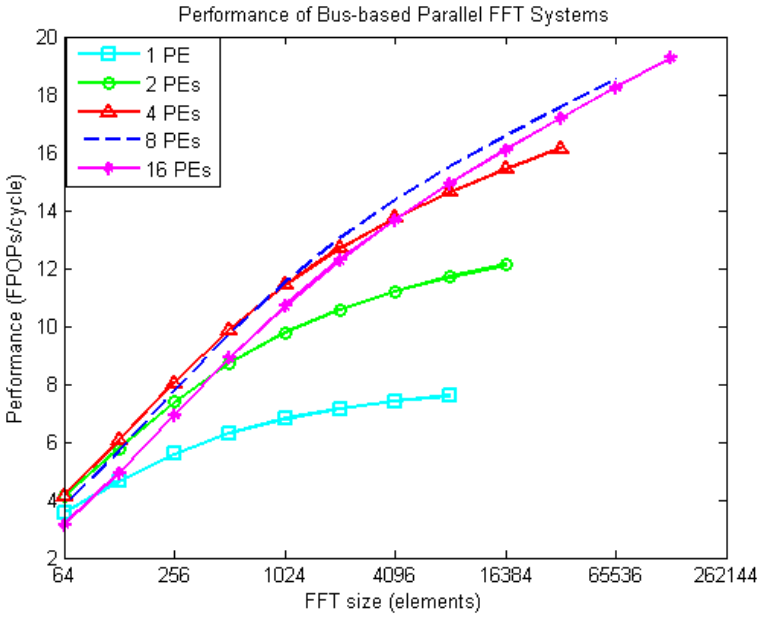


Figure 3.17: Performance of bus-based parallel FFT systems.

Therefore, it can be concluded that a NoC is a better communication scheme for a parallel FFT implementation than a bus.

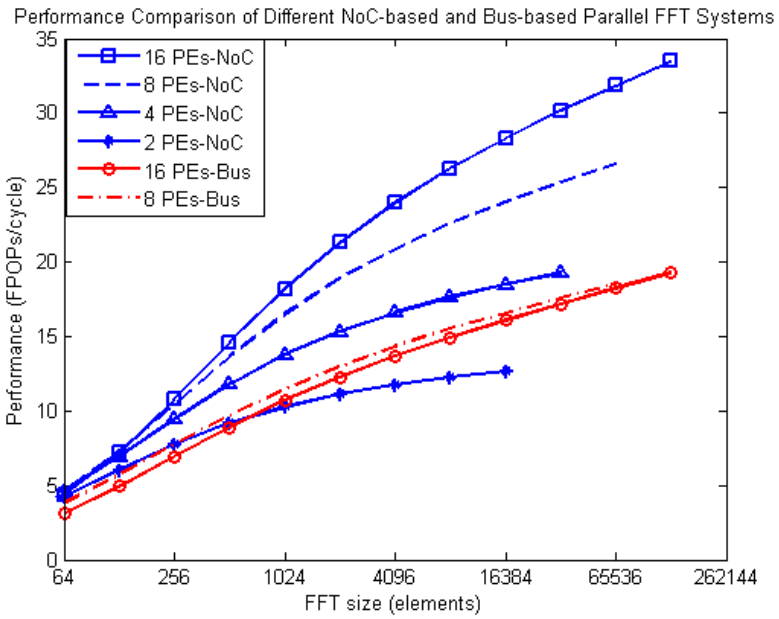


Figure 3.18: Performance comparison of different NoC-based and bus-based parallel FFT systems.

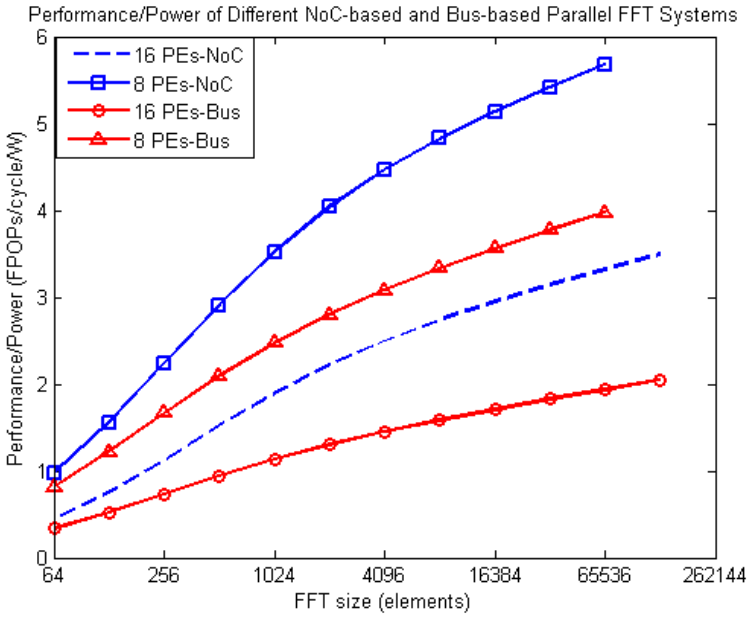


Figure 3.19: Performance/Power Comparison of different NoC-based and bus-based parallel FFT systems.

The performance of the best NoC-based parallel design can also be compared with with the Xilinx FFT IP core (Xilinx, 2012a) in terms of calculation cycles to demonstrate the advantage of a NoC-based parallel FFT implementation to the more common pipelined architecture in commercial FFT cores. The Xilinx FFT IP core supports the transformed size of $N = 2^m$ $(3 \leq m \leq 16)$, data precision of 8 to 34 bits, and fixed-point and block floating-point data format. The core provides four architectures: Radix-2, Radix-2 Lite, Radix-4 and Pipelined Radix-2. The Radix-2 Lite and Radix-4 architectures are not considered here because they have different butterfly structures, either lighter or heavier than the Radix-2.

The new 1-PE design has similar performance to that of the Xilinx Radix-2 FFT core. For example, the 1-PE design processes a 1024-point FFT in 7538 cycles; and the Xilinx Radix-2 FFT core needs 7367 cycles. The performance comparison between the dynamically reconfigurable 16-PE design and the Xilinx Pipelined Radix-2 FFT core is shown in Fig. 3.20. However in making this comparison it should be noted that the number of PEs in the Xilinx Pipelined Radix-2 architecture in this case is smaller than 16. For example, only 13 PEs are used in the Pipelined Radix-2 architecture to perform

the 8192-point FFT. This comparison aims to illustrate the advantage of the parallel architecture over the pipelined architecture, in which more than n PEs can be used to perform a 2^n-point FFT to achieve smaller latency. Power and area comparison is not carried out here as the Xilinx FFT cores do not support double-precision floating-point data format.

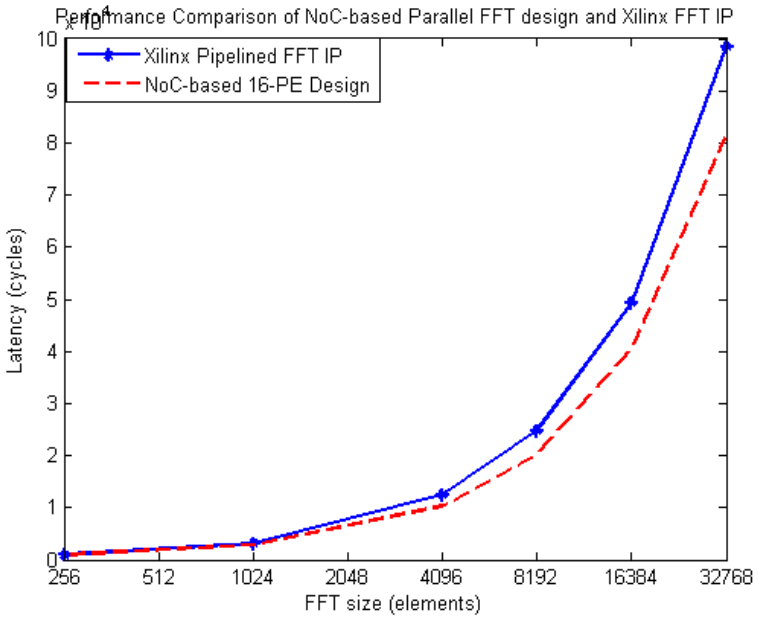


Figure 3.20: Performance comparison of NoC-based parallel FFT design and Xilinx FFT IP.

Parallel-pipelined FFT systems

Fig. 3.21 and Fig. 3.22 respectively show the performance and performance/power comparison of the NoC-based and bus-based parallel-pipelined systems of 16 PEs with different FFT sizes. It can be seen from the figures that performance of the parallel-pipelined systems increases with an increase in the FFT size. However there is a decrease in performance with the 2^9-point and 2^{17}-point FFTs. This is because parallel-pipelined architecture is implemented with 8 stages only. The implementation of a 2^9-point FFT requires a transfer of the output values in the stage 8 to the external memory and the implementation of a 2^{17}-point FFT requires two transfers. This leads to higher latency in these implementations and results in lower performance. Despite that, a

NoC still outperforms a bus in terms of both performance and performance/power for a parallel-pipelined FFT implementation.

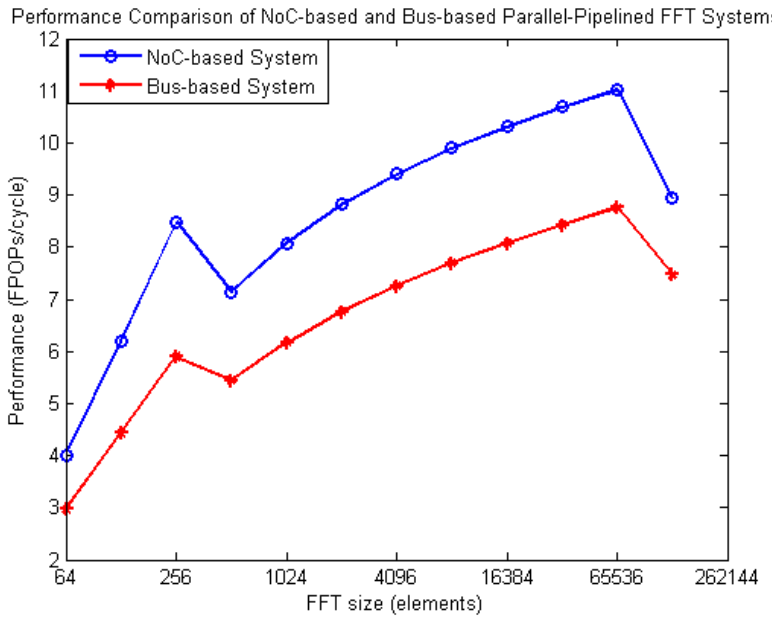


Figure 3.21: Performance comparison of NoC-based and bus-based parallel-pipelined FFT Systems.

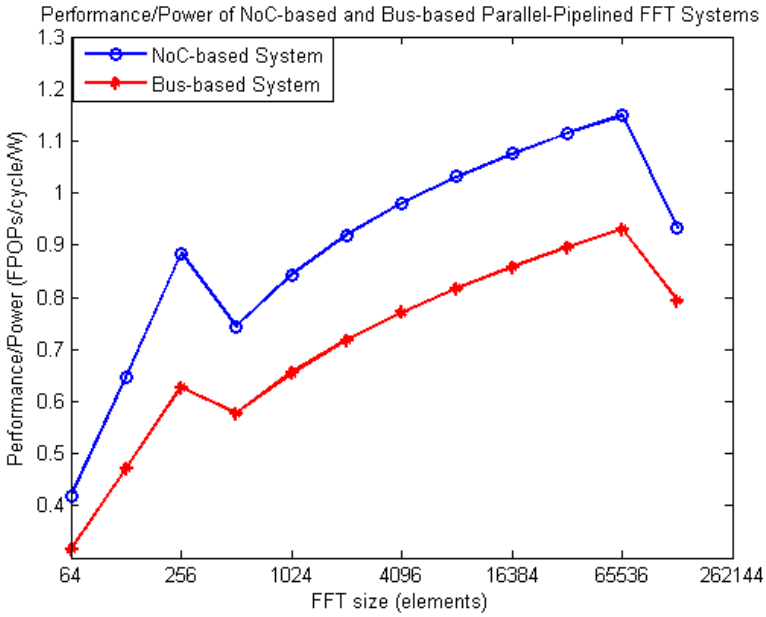


Figure 3.22: Performance/power comparison of NoC-based and bus-based parallel-pipelined FFT Systems.

3.4 Multi-channel Buses

The results in the previous section show that buses have lower resource overheads than NoCs but achieve lower performance due to bus contention. This raises the question: can additional resources be added to a bus so that its performance matches or exceeds that of a NoC at lower overhead?

Most publications on buses consider a bus with only one channel (Cordan, 1999; Winegarden, 2000; Lahiri *et al.*, 2006; Shanthi and Amutha, 2011); however, (Jung *et al.*, 2008) describe a bus with multiple channels . Their overall structure is equivalent to using a crossbar between PEs. For a system with 16 PEs and a 16-channel bus, any PE can send data to any other without contention.

In this section, a 16-channel bus with 128 bits each channel is implemented based the work by (Jung *et al.*, 2008). The bus synthesized for a Xilinx Virtex-7 XC7VX485T FPGA occupies 2661 register slices and 11150 LUTs slices that are respectively 15-times and 9-times higher than those occupied by a single channel bus. These results are much higher than those of a single router but still lower than a network of 16 routers. The post synthesis maximum frequency of the 16-channel bus is 480 MHz.

Fig. 3.23 shows the comparison of resource consumption of the NoC-based, single channel bus-based and 16-channel bus-based FFT systems with different number of PEs. It can be seen from the figures that the 16-channel bus-based systems consume more resources than the corresponding single bus-based systems but consume less resources than the corresponding NoC-based systems. The power consumption of the three kinds of systems at 100 MHz are given on Table 3.3. It can be seen from the table that 16-channel bus-based systems consume more power than the corresponding single bus-based systems and also consume more power than the corresponding NoC-based systems despite of their lower resource consumption.

Fig. 3.24 shows the performance of the 16-channel bus-based parallel FFT systems with different numbers of PEs and FFT sizes. It can be seen that the performance increases with an increase in the number of running PEs and FFT size. With the use of a full 16-channel bus for a system of 16 PEs, the performance can now be scalable with the increase in the number of running PEs.

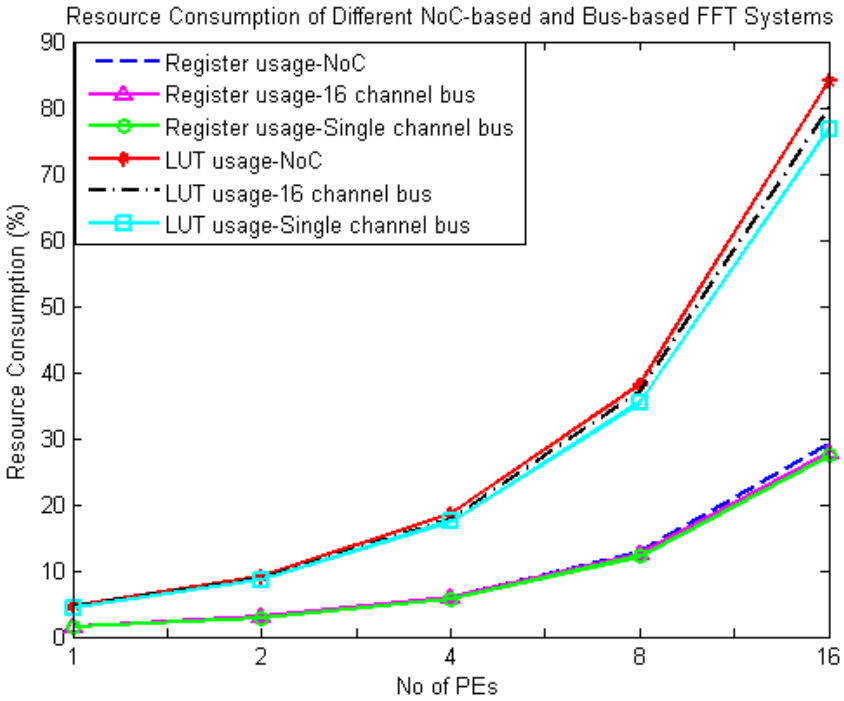


Figure 3.23: Resource consumption of different NoC-based and bus-based FFT systems.

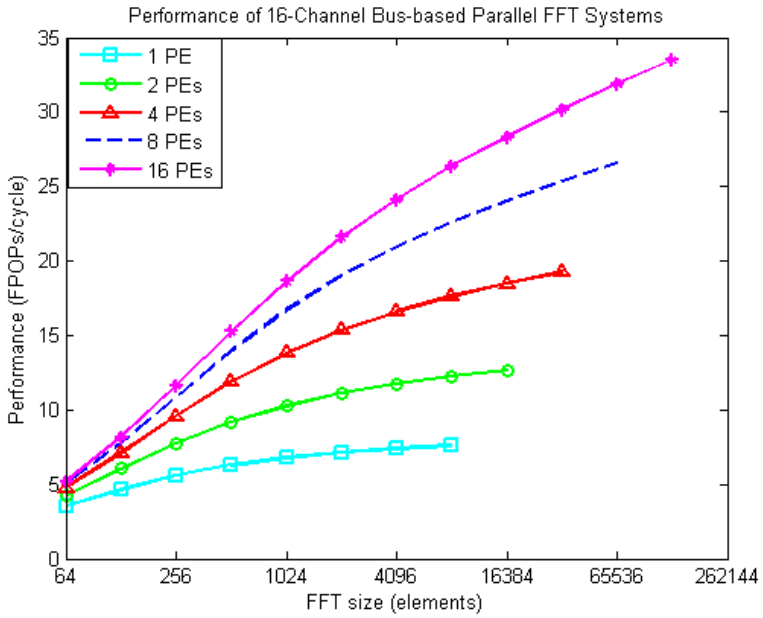
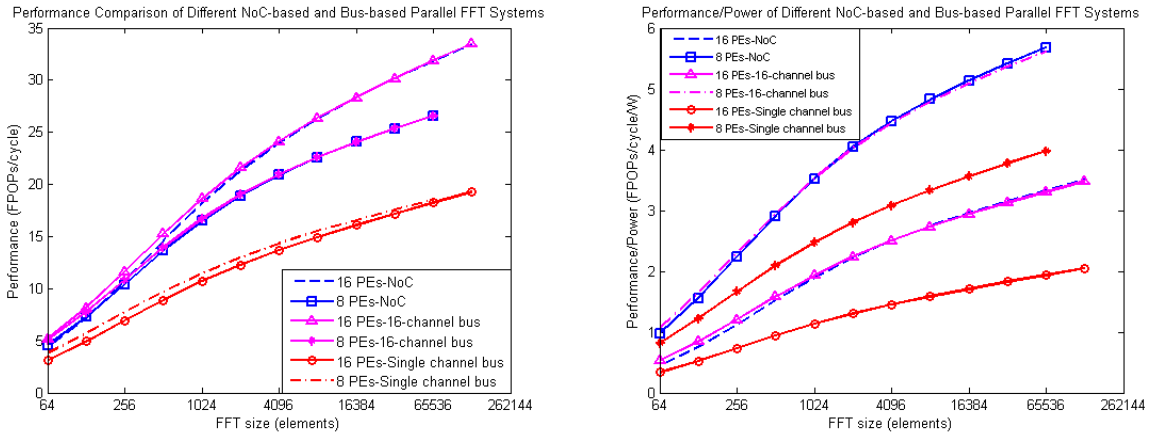


Figure 3.24: Performance of 16-channel bus-based parallel FFT systems.

Table 3.3: Power consumption [W] of different FFT systems at 100 MHz

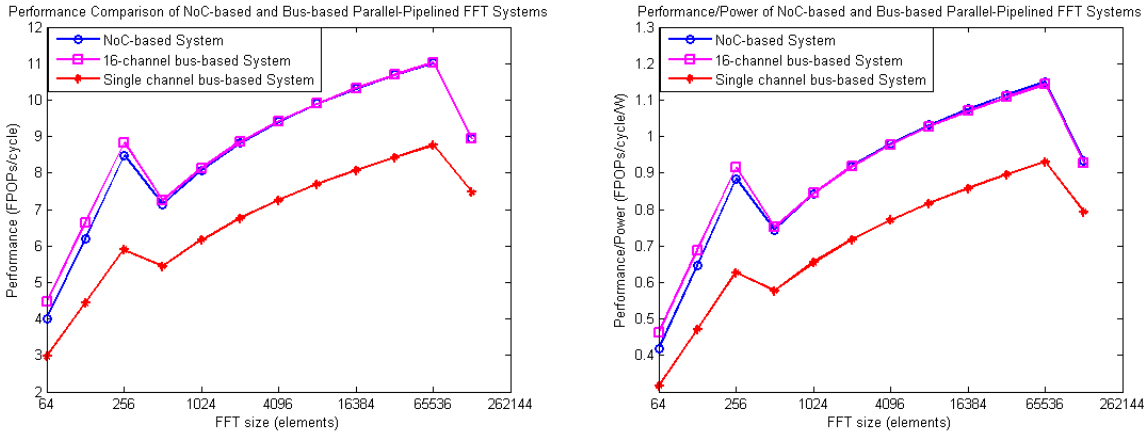
	NoC-based systems	16-channel bus-based systems	Single channel bus-based systems
1 PE	1.165	1.155	1.149
2 PEs	1.406	1.378	1.374
4 PEs	2.469	2.478	2.458
8 PEs	4.682	4.733	4.651
16 PEs	9.589	9.644	9.415

Fig. 3.25 (a) and (b) respectively show the performance and performance/power comparison of the NoC-based, single channel bus-based and 16-channel bus-based parallel FFT systems with different number of PEs. It can be seen from the figures that the performance and performance/power of the 16-channel bus-based systems are much higher than those of the corresponding single channel bus-based systems. The 16-channel bus-based systems achieve similar performance and performance/power to the corresponding NoC-based systems. The same results are obtained with the parallel-pipelined systems of 16 PEs as shown in Fig. 3.26 (a) and (b).



(a) (b)

Figure 3.25: Comparison of different NoC-based and bus-based parallel FFT systems. (a) Performance. (b) Performance/Power.



(a) (b)

Figure 3.26: Comparison of different NoC-based and bus-based parallel-pipelined FFT systems. (a) Performance. (b) Performance/Power.

In summary, the implementation of multiple channels allow a bus to achieve similar performance to a NoC. A multi-channel bus-based FFT system consumes slightly less area and more power consumption than a NoC-based FFT system. However, a NoC still has advantage of system scalability in comparison to a multi-channel bus. It is noted that only 16 PEs can be implemented in this case. With larger number n of PEs, the bus also needs to be implemented with full n channels to obtain an optimal performance. In a NoC-based system, only additional routers need to be added to the NoC when the system size increases. There is no need for changing the entire interconnect like in a bus-based system.

3.5 Conclusion

In this chapter, the FFT systems using different interconnection strategies have been implemented on a Xilinx Virtex-7 XC7VX485T FPGA. NoCs and buses are compared in terms of area, power consumption and performance. The buses are implemented with a single channel and multiple channels for comparison. The NoC provides much better performance than the single channel bus and similar performance to the multi-channel bus in both parallel and parallel-pipelined FFT implementations. The only disadvantage of NoCs is their higher area and power consumption. However, this

disadvantage is not significant when considered in the context of the entire system. The area and power consumption of a NoC-based system is just slightly higher than these of a single channel bus-based system but with much higher performance. This leads to a better ratio of performance/power of the NoC-based system. The performance and power consumption results for NoCs and multi-channel buses are very similar; however a NoC-based systems has the advantage of of system scalability. Based on these analysis and experimental results, it can be concluded that NoCs are a better solution for the FFT than buses.

The NoC-based interconnect approach provides scalable and sustainable performance when the design size increases. The best NoC-based parallel design provides smaller latency than Xilinx pipelined FFT cores. This demonstrates the advantage of using a NoC for a parallel FFT implementation to provide higher design parallelism and lower latency than the more common pipelined architecture in commercial FFT cores. Results of the FFT case study also prove the need for considering and examining different interconnect architectures in fully realized systems for realistic applications to obtain a complete and comprehensive comparison. The next chapter will examine NoCs and buses for another case study, a neural network implementation.

Chapter 4

Interconnect Architectures for Neural Network Implementations

THIS chapter presents another case study, a feed-forward neural network for handwritten digit recognition. Dedicated NoC and bus interconnects are implemented to best suit the massive communication requirements of the network. Different design configurations are discussed and examined. Experiments are carried out to compare area, power consumption, and performance of the two interconnect architectures for the neural network application. The advantages of the proposed NN implementations compared to traditional implementations are also highlighted. Combining with the results in Chapter 3, this chapter gives recommendations on the applications and communication requirements each interconnect architecture is better used for.

4.1 Background

Artificial neural networks (NNs) have been used for a variety of problems that are hard to solve using other programming approaches and they have proved especially useful for signal processing tasks such as computer vision and speech recognition (Basheer and Hajmeer, 2000). NNs can be implemented in hardware or in software on a general purpose processor. While software implementations provide flexibility, and have received the most attention to date, hardware implementations can take advantage of the inherent parallelism of NNs. There is growing interest in the use of platforms such as FPGAs for high performance implementations of NNs (Misra and Saha, 2010).

Many FPGA-based implementations of NNs have been proposed for different applications such as speed estimation for a two-mass drive system (Orlowska-Kowalska and Kaminski, 2011), face recognition (Sudha *et al.*, 2011), and infant cry recognition (Suaste-Rivas *et al.*, 2006). Some implementations have been proposed for effective utilization of the limited resources on an FPGA. (Himavathi *et al.*, 2007) present a layer multiplexing implementation of feed-forward NNs. In this implementation, only the largest layer in the network is implemented. This layer is emulated to behave as the other layers by a control block. Stochastic-based NN implementations are proposed by (Zhang and Li, 2008; Canals *et al.*, 2016) to simplify the computational elements of the NN and reduce the hardware resources required. Stochastic-based implementations are simple but not always efficient (Misra and Saha, 2010). Most of these implementations have been described in small NNs of several tens of neurons. Few have discussed implementations for large networks of many neurons.

In recent years, the significant increase in hardware resources and computing power of FPGAs has made it possible to implement many neurons on a single FPGA, but the interconnect between these neurons has become a challenge. In a fully connected feed-forward NN every neuron in a layer is connected to all the neurons in adjacent layers as shown in Fig. 4.1. The mass of interconnections in a large NN can lead to communication bottleneck. Most current NN hardware implementations use point-to-point connections but this scheme cannot scale for the massive communication requirements of a large number of neurons.

In this chapter, a pre-trained large feed-forward NN is implemented on an FPGA. Dedicated NoC and bus interconnects are used for communications among the neurons in the NN. A large feed-forward NN is chosen as its massive interconnection requirements provide the opportunity to evaluate the performance of different communication architectures. In addition, a full realization of a large NN on a single FPGA is still a challenging problem due to its complex communication and computation requirements. In this chapter, different design parameters are discussed in conjunction with FPGA resources. The systems are evaluated for performance, area and power consumption. Their performance is compared with a software implementation on CPUs in an example of handwritten digit recognition. This comparison is useful to confirm that FPGA realization does indeed have the potential to exceed the performance that can be achieved on a CPU.

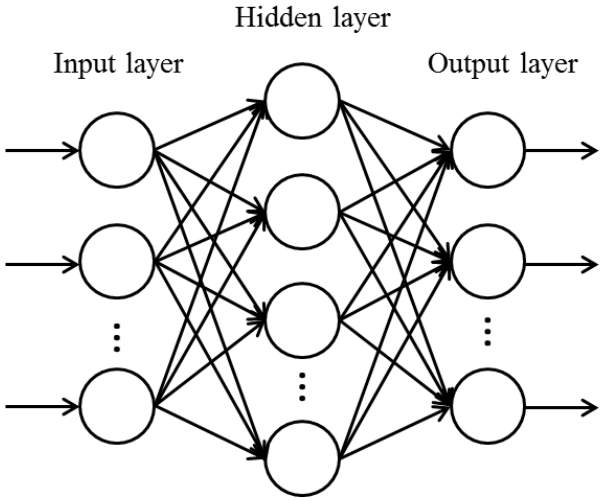


Figure 4.1: Two-layer feed-forward neural network.

4.2 Neurons and Processing Elements

4.2.1 Neurons

The operation of a N-input neuron is based on the formula:

$$y = K(\sum_{i=1}^{N} w_i g_i(x)) \tag{4.1}$$

where the coefficient w_i is known as the weight for the interconnection g_i. K is known as the activation function and is typically the hyperbolic tangent or sigmoid function. The sigmoid function is used in this implementation.

For these experiments, a single neuron is realized as simply as possible while ensuring it can be efficiently reused for any layer of the network. It consists of m multipliers, $(m-1)$ adders, an accumulator, m sign-magnitude/two's complement convertors, a two's complement/sign-magnitude convertor and a look-up table (LUT) for the sigmoid function. A neuron processes m input values at a time. Efficient selection of m will be presented in detail in the next section. An implementation with $m = 4$ is illustrated in Fig. 4.2.

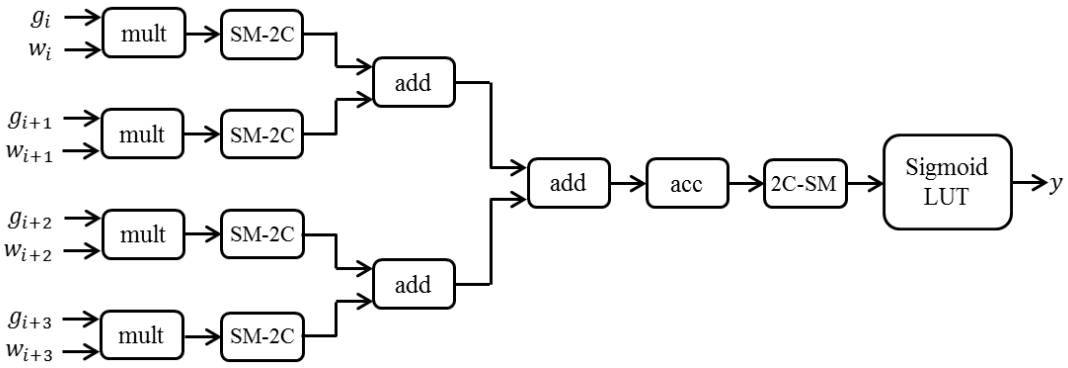


Figure 4.2: 4-multiplier neuron implementation.

Fixed-point data format with different precisions for each sub-module is used. In this implementation, weights and outputs of multipliers and accumulators are in 16-bit format, with 1 bit for sign, a 3-bit integer part, and a 12-bit fractional part. Input data are in 13-bit format, with 1 bit for sign and a 12-bit data part. Outputs of the activation function are in 12-bit format with all bits being fractional. These precisions can be adjusted according to different application precision requirements. The sign-magnitude results of multipliers are converted to two's complement format for addition. The two's complement results of the accumulators are converted back to sign-magnitude for the sigmoid function. This allows us to take advantage of the symmetry in the function and hence halve the size of the look-up table. The sigmoid activation function is implemented using a look-up table. The size of the look-up table is chosen based on the

application precision requirements. In this chapter, the look-up table is implemented with 151 sample values in the range [-7.5 : 7.5] with step 0.1.

Increasing m, the number of multipliers in the neurons, reduces the number of cycles required to complete the calculations for a layer. However, using too many parallel multipliers in a single neuron can lead to an inefficient use of hardware resources, especially the block RAMs (BRAMs) used to store the inputs to a layer and the weights. BRAMs in the latest devices of the two largest FPGA providers Xilinx and Altera can be set to dual-port mode with maximum data width of 72 bits. The use of m multipliers requires reading m input values and m weight values from the input and weight memories at a time. This can be done by two different memory implementations.

In the first implementation, m sub-memories are used for each of the input and weight memories. Since each BRAM can only have up to two read ports, at least $\frac{m}{2}$ BRAMs are required. In the second implementation, the data width of the memories is set to m times the data precision. In other terms, $13m$-bit data words are used for the input memories and $16m$ bit data width are used for the weight memories. Each BRAM can have its data width double to 72 bits; hence, a minimum number of $\frac{16m}{72}$ BRAMs are required.

With the same value of m, the latter implementation consumes less than half the number of BRAMs required by the former one. Therefore, it is used in this paper. Each BRAM can support a maximum number of 4 parallel multipliers implemented in a single neuron. Therefore, m is chosen as a multiple of 4 for efficient use of BRAMs. In addition, the selection of m needs to be considered in conjunction with the available hardware resources on the FPGA so that as many neurons and parallel multipliers can be implemented as possible.

The design is intended for large NNs with hundreds of neurons in each layer. It exploits the parallelism within a layer to improve end-to-end latency of the network. One layer is processed at a time. The number of implemented neurons depends on the available resources of the FPGA and does not exceed the number neurons of the largest layer. Each implemented neuron can perform the operation of one or several logical neurons in the NN according to the number of logical neurons in each layer.

4.2.2 Processing Elements

A PE consists of n parallel neurons. This saves on memory because the neurons in the PE can share the memory used for input data. In addition, it reduces communication resources (i.e. NoC routers and bus logic). A router is attached to a single PE rather than a single neuron. However, too large values of n can also reduce the speed of the PE. A PE also has memory storage for input data, weights and intermediate results; a local controller; and a NoC/bus interface as shown in Fig. 4.3. The local controller coordinates the read/write memory operation of neurons at each layer. The NoC/bus interface works as a bridge between the PE and the NoC/bus.

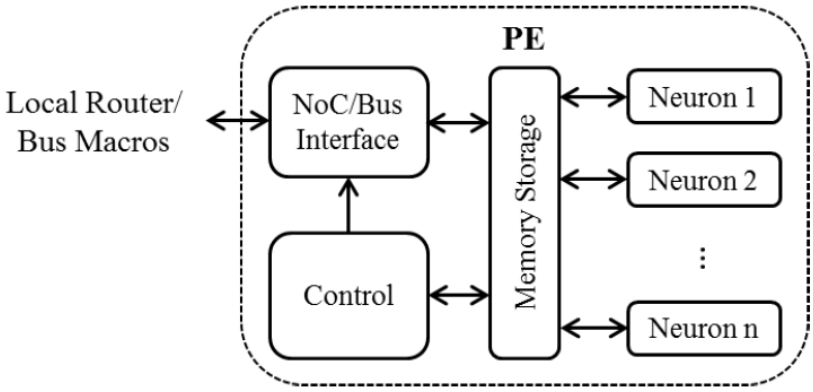


Figure 4.3: PE architecture.

4.3 Interconnect Architectures for Neural Networks

4.3.1 NoC-based Interconnect

The NN implementation involves complex routing infrastructure for transferring input data and weights from external memory to PEs, transferring intermediate results among PEs, and outputting the final results from each PE to the external memory. These can all be efficiently implemented by a NoC. Some researchers have mapped NNs on to a NoC (Theocharides *et al.*, 2004; Mand *et al.*, 2012; Vainbrand and Ginosar, 2010; Diguet *et al.*, 2013; Dong *et al.*, 2010). However, the NoCs used in these works are mostly regular NoCs without customized characteristics for NNs. Therefore, performance achieved by these systems is usually non-optimal.

In this implementation, a 2-D mesh topology is used as it is a good match with the organization of an FPGA. An example of a 4×4 mesh NoC is shown in Fig. 4.4. Wormhole flow control is used to minimize packet latency and input buffers. Two different routing algorithms are implemented. A broadcast algorithm is used for transferring input data from external memory to the PEs and distributing intermediate results among PEs. An all-to-all broadcast algorithm is used for distributing intermediate results among PEs rather than a multicast algorithm as it can be efficiently used for all layer configurations without readjustments. The X-Y routing algorithm is used for transferring weights from external memory to PEs and results from PEs to external memory.

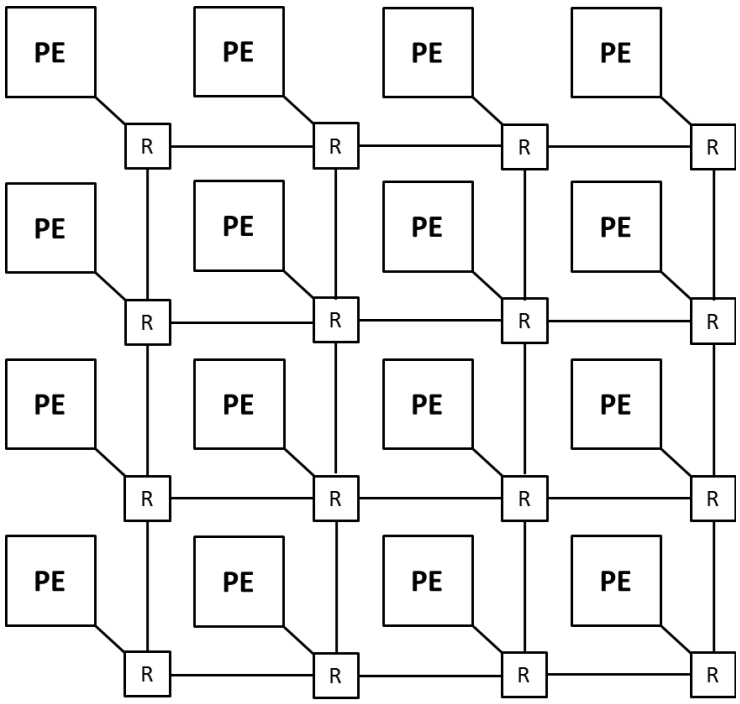


Figure 4.4: 4×4 mesh NoC.

Each router has five input/output ports as shown in Fig. 4.5: four from the four cardinal directions (North, East, South and West); and one from the local PE. There is an input buffer in each input port. Input buffer size is chosen so that it is as small as possible while ensuring there is no deadlock in the network. The correct size depends on the NoC size. A router also contains a direction calculator, an arbiter and a crossbar switch. The direction calculator determines the route for the current packet in each input port. The arbiter determines which input port is selected to proceed in the next stage. Then,

the crossbar switch matches the successful input port with the desired output port. For high speed, the routers are pipelind with 5 stages: buffer write, direction calculation, arbitration, switch matching and buffer read. Each router is connected to its neighbors and local PE through a bidirectional link. The link bandwidth is chosen based on the data requirements of the PEs.

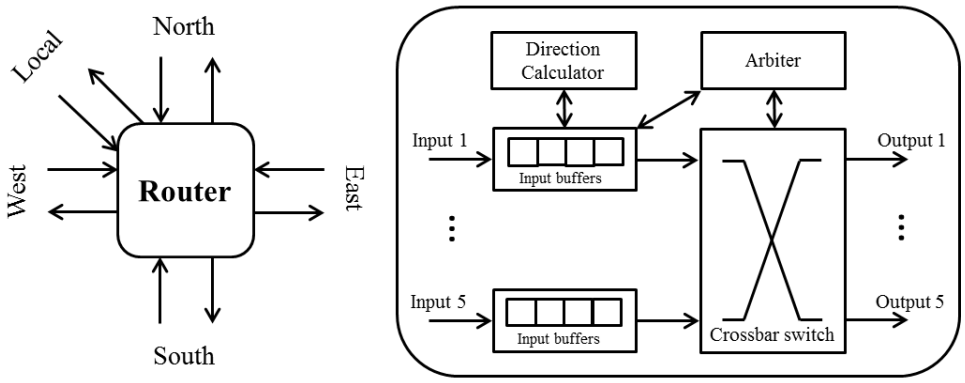


Figure 4.5: Router datapath and microarchitecture.

Wormhole flow control, a flit-based flow control mechanism is used. This allows a flit to leave the router as soon as a downstream buffer is available for it. Therefore, this reduces packet latency as well as input buffer size. There are 3 types of flits identified by the two most significant bits: 10 for header flits, 00 for data flits and 01 for tail flits. Header flits contain information for the routing process such as broadcast or XY routing mode, source node and destination node co-ordinates.

The broadcast algorithm is implemented based on the work by (Yang and Wang, 2001). Its broadcast pattern is illustrated in Fig. 4.6. In the X-Y routing algorithm, packets are first sent to the X direction and then the Y direction until they get to their destination co-ordinates. For both routing algorithms, a priority-to-the-right rule is applied in arbitration. This means a port sets the highest priority for the incoming packets from its right side port.

4.3.2 Bus-based Interconnect

The bus customized for a NN implementation is a global shared bus with a data channel width chosen based on the data requirements of the PEs and an address channel width

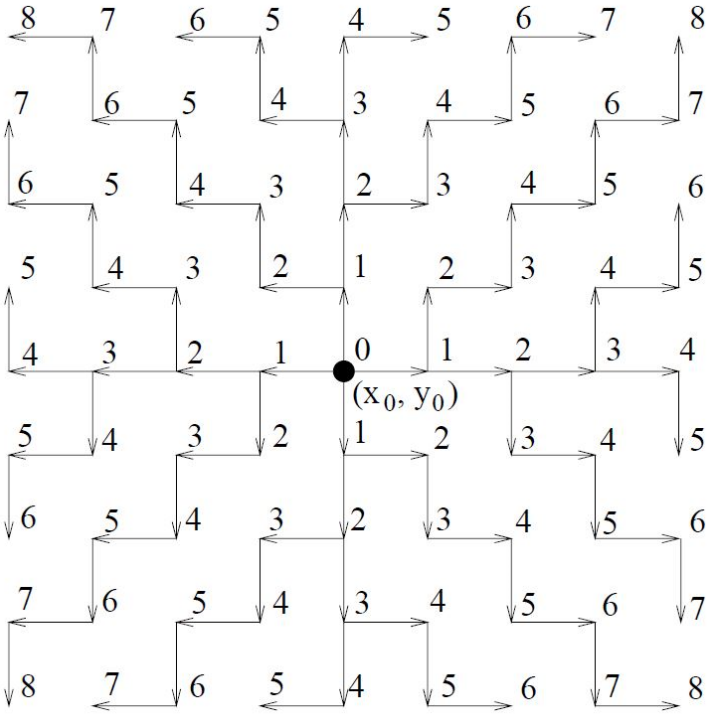


Figure 4.6: Broadcast pattern from source node (x_0, y_0).

chosen according to the maximum number of implemented PE. Bus access arbitration and address decoding are implemented in a centralized manner. There is an additional address used for broadcasting mode. When this address is put on the address channel, the decoder will send a select signal to all PEs attached to the bus. A static priority scheme is used for arbitration as it is simple but can provide high performance with NN communication requirements. Multi-channel buses are not considered in this chapter because the use of multiple channels dedicated to every PE are useless for broadcasting communications but with additional resource overheads.

4.4 The Handwritten Digit Recognition Case Study

4.4.1 Implementation

The proposed NN can be used for different network configurations and applications. This section presents the design and evaluation of a NN for handwritten digit recognition (HDR). This application was chosen because it is a well-known benchmark. The

focus of this chapter is on the use of NoC and bus interconnects so other aspects of the NN described here are conventional. I have not, for example, sought to improve implementation performance by heavily quantizing intermediates, or by using alternative activation functions. Such optimizations are the topic of related research efforts (e.g. (Lai *et al.*, 2015; Rastegari *et al.*, 2016; Huang and Babri, 1998)) and are compatible with the NoC and bus structures described here.

A two-layer network with a hidden layer and an output layer was used. The MNIST data set is used for training and testing (LeCun *et al.*, n.d.). This provides a training set of 60,000 handwritten digits and a testing set of 10,000 handwritten digits. The images have a size of 28×28 pixels. Therefore, the network had $28 \times 28 = 784$ inputs and 10 outputs. 512 neurons were used in the hidden layer. The network was trained in Matlab with accuracy rates of 92% and 93% for training and testing respectively. The hardware design of the trained network was expressed in Verilog and verified using Modelsim. The design verification in Modelsim provides an accuracy rate equivalent to that achieved by the software implementation in Matlab. There were 702 errors over the testing set of 10,000 handwritten digits. Synthesis and power analysis were performed with the Xilinx ISE Design Suite, targeting a Xilinx Virtex-7 XC7VX485T FPGA. Designs with different implementation parameters are compared below in terms of area, power consumption, and latency.

For comparison, neurons were implemented with m = 1, 4 and 8 parallel multipliers. PEs were implemented with n = 1, 4 and 8 parallel neurons. Each PE requires an input memory of 784×13 bits, n weight memories of $(784 + 512) \times 16$ bits each and n output memories of 1×12 bits each. The data width of input and weight memories varies according to the the neuron type used in the PE. They are proportional to the number of parallel multipliers in a neuron. The weight and output memories sizes here are for only one operation in each neuron in each layer. For multiple operations, the size is properly scaled.

Synthesized results of a single NoC-compatible PE and a single bus-compatible PE with different configurations ($n \times m$) for a Xilinx Virtex-7 XC7VX485T FPGA are given in Table 4.1 and Table 4.2 respectively. It can be seen from the tables that the hardware resources required for a PE increase with an increase in ($n \times m$). However, better choices of ($n \times m$) can lead to better use of BRAMs. Firstly, m should be a multiple of 4. A

PE($n \times 4$) uses the same amount of BRAM as a PE($n \times 1$) but the former is 4 times faster. The number of BRAMs required for $m = 8$ is double that required for $m = 4$. Secondly, for a given value of m, increasing n can save BRAMs. Finally, if we compare PE($a \times b$) with PE($b \times a$) for $a > b$ then we find PE($a \times b$) uses fewer BRAMs but more registers and LUTs. Hence the choice of n and m depends on the available resources of the FPGA and the desired trade between FPGA resource use, power consumption, and performance of the entire network. A NoC-compatible PE consumes slightly higher resources and power than a bus-compatible PE.

The maximum number of neurons that can be implemented on the FPGAs for each type of PE is also given in Table 4.1 and Table 4.2. It is noted that the maximum number of neurons here is a factor of 512 (i.e. number of neurons in the biggest layer of the network). All of these measurements take into account all of the resources for the NN, including the NoC and the bus. Note that different configurations place different constraints on the data link widths for the NoC and the bus.

The NoC is implemented with different data link widths (18 bits, 66 bits, 130 bits) based on different data widths needed for the weight memories. Synthesized results of a single router are given in Table 4.3. It can be seen from Table 4.3 that a router implemented with 18-bit link width is quite small and does not use any BRAM. Its buffers are quite small, so they are implemented using distributed memories to save BRAM resources on the FPGA. Routers with 66-bit and 130-bit links use 5 and 10 BRAMs respectively.

The bus is implemented with different data link widths (16 bits, 64 bits, 128 bits) and maximum numbers of attached PEs (32, 64, 128, 256, 512). Synthesized results of different bus configurations are given in Table 4.4. It can be seen from Table 4.4 that the resource consumption increases significantly with the increase in the bus data link width and maximum number of attached PEs. The number of registers and LUTs occupied by a 128-bit bus with 512 PEs are respectively 8-times and 34-times higher than those occupied by a 128-bit bus with 32 PEs. On the contrary, the maximum frequency of the bus decreases considerably with the increase in maximum number of attached PEs. A 128-bit bus with 32 PEs can run at the maximum frequency of 249 MHz while a 128-bit bus with 512 PEs can only run at the maximum frequency of 146 MHz.

Table 4.1: Synthesis and power analysis results for a single NoC-compatible PE

	PE(1x1)	PE(4x1)	PE(8x1)	PE(1x4)	PE(4x4)	PE(8x4)	PE(1x8)	PE(4x8)	PE(8x8)
Number of slice registers	190	451	796	309	807	1437	508	1290	2349
Number of slice LUTs	334	927	1682	506	1575	2979	894	2636	4977
Number of DSP48E1s slices	1	4	8	4	16	32	8	32	64
Number of 36Kb BRAMs	2	5	9	2	5	9	4	10	18
Post synthesis frequency (MHz)	214	213	212	370	370	370	370	370	370
Estimated power at 100MHz (W)	0.259	0.286	0.323	0.267	0.309	0.360	0.285	0.346	0.430
NoC router data link width	18	18	18	66	66	66	130	130	130
Maximum neurons implemented	256	512	512	128	256	512	128	128	256

Table 4.2: Synthesis and power analysis results for a single bus-compatible PE

	PE(1x1)	**PE(4x1)**	**PE(8x1)**	**PE(1x4)**	**PE(4x4)**	**PE(8x4)**	**PE(1x8)**	**PE(4x8)**	**PE(8x8)**
Number of slice registers	165	426	772	232	729	1430	340	1282	2342
Number of slice LUTs	291	886	1672	464	1498	2972	704	2449	4729
Number of DSP48E1s slices	1	4	8	4	16	32	8	32	64
Number of 36Kb BRAMs	2	5	9	2	5	9	4	10	18
Post synthesis frequency (MHz)	214	213	212	370	370	370	370	370	370
Estimated power at 100MHz (W)	0.257	0.285	0.318	0.265	0.303	0.363	0.279	0.347	0.427
Bus data link width	16	16	16	64	64	64	128	128	128
Maximum neurons implemented	512	512	512	512	512	512	256	256	256

Table 4.3: Synthesis results for a single router

	18 bits	**66 bits**	**130 bits**
Number of slice registers	331	237	237
Number of slice LUTs	992	1127	1447
Number of 36Kb BRAMs	0	5	10
Post synthesis frequency (MHz)	401	399	399

4.4.2 Latency

NoC-based System Latency

The end-to-end latency (L) for the NoC-based network to complete one digit recognition consists of the total calculation cycles (C) and the NoC latency (T). C is the sum of calculation cycles of all layers except the input layer. The calculation cycles of layer i is:

$$C_i = I + \frac{(n_{i-1}n_i)}{(mr_i)} \tag{4.2}$$

Here, I is the initial delay, which is equal to the number of pipeline registers in a single neuron. A neuron with 1 multiplier is deeply pipelined with 8 stages for high speed: 1 for read operation of input data, 1 for multiplication , 1 for sign-magnitude/two-complement conversion, 1 for addition, 1 for accumulation, 1 for two-complement/sign-magnitude conversion, 1 for the sigmoid LUT and 1 for the result write operation. For neurons with 4 and 8 multipliers, the accumulators is separated in 2 and 3 pipelined stages respectively. n_i and r_i are the number of logical neurons and the number of physical neurons in layer i respectively. m is the number of parallel multipliers in a single neuron.

The NoC latency T is given by:

$$T = T_{mp} + T_{pp} + T_{pm} \tag{4.3}$$

Table 4.4: Synthesis results for different bus configurations

(a) 16-bit data bus

Maximum PEs attached	32	64	128	256	512
Number of slice registers	101	285	428	795	1562
Number of slice LUTs	565	866	1723	5304	12545
Post synthesis frequency (MHz)	254	247	227	171	147

(b) 64-bit data bus

Maximum PEs attached	32	64	128	256	512
Number of slice Registers	145	331	582	483	1610
Number of slice LUTs	1175	1922	3787	21589	45309
Post synthesis frequency (MHz)	249	247	227	171	147

(c) 128-bit data bus

Maximum PEs attached	32	64	128	256	512
Number of slice registers	214	401	650	908	1674
Number of slice LUTs	1999	3329	6539	34664	69153
Post synthesis frequency (MHz)	249	247	227	168	146

Here, T_{mp} is the one-to-all broadcasting time of input data from the memory to PEs. It is the sum of the one-to-all broadcasting routing time and the transfer time of input data flits. The routing time depends on the hop number and the NoC size. It takes 5 cycles to proceed and route a packet at each hop or each router. The transfer time of input data flits is quite significant; however, it is overlapped with the calculation time of the neurons. A neuron starts its operation right after it receives the first flit. Therefore, for a $h \times k$ NoC, the maximum value of T_{mp} is $5(h + k)$ cycles.

T_{pp} is the all-to-all broadcasting time of intermediate results among PEs at all hidden layers. This increases with the increase in NoC size and the number of flits transferred at these layers. In (Yang and Wang, 2001), this time is proved to be no more than:

$$\alpha + \delta + F\gamma + (\frac{hk-1}{4} - 1)max(\delta, F\gamma) + min(\delta, F\gamma) \tag{4.4}$$

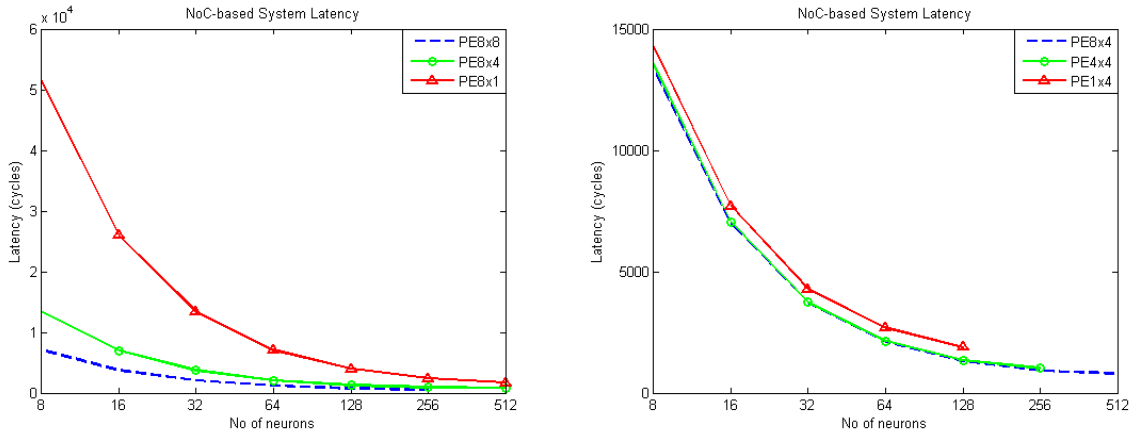
where α, δ, γ, F are the start-up time, the switching time, the transmission time per flit, and the number of flits per message respectively.

T_{pm} is the sum of the output transfer times from each PE to the memory at the output layer. This depends on the positions of PEs and the memory in the network. The transfer time of F flits from the PE at node (x_p, y_p) to the memory at node (x_m, y_m) is approximately:

$$5(|x_i - x_m| + |y_i - y_m|) + F \tag{4.5}$$

The latency for recognizing one handwritten digit of different NoC-based design configurations and running neurons is as shown in Fig. 4.7. It can be seen that the latency decreases with an increase in the number of running neurons. Fig. 4.7 (a) shows the latency of designs with 8 parallel neurons in a single PE and (1, 4, 8) parallel multipliers in a single neuron. It can be seen that for a given number of running neurons, the latency decreases as the number of parallel multipliers in each neuron is increased. Fig. 4.7 (b) shows the latency of designs with 4 parallel multipliers per neuron and (1, 4, 8) parallel neurons per PE. For a given number of running neurons, the latency decreases with the increase in the number of parallel neurons per PE. This is because with more parallel neurons in a single PE, the number of used PEs is smaller; or in other terms, the NoC size and its latency is smaller. This leads to lower latency overall.

The smallest latency of 607 cycles is achieved by the NoC-based network of 32 PEs(8×8). Running at 100MHz, it takes about 6.07μs for 1 digit recognition or 60.7ms for a test vector set of 10,000 digits. An Intel i7-3.4GHz CPU running Matlab R2014 needs 1.093s for this job. This time is the average of 100 consecutive measurements. In other terms, the best NoC-based design is 18-times faster than an i7-CPU running Matlab. It could be even faster with an FPGA large enough for 512 neurons or 64 PEs(8×8).



(a) (b)

Figure 4.7: Latency comparison of different design configurations.

Bus-based System Latency

The end-to-end latency (L) for the bus-based network to complete one digit recognition consists of the total calculation cycles (C) and the bus latency (B). The calculation cycles of layer (C) is similar to that of the NoC-based network.

The bus latency B is given by:

$$B = B_{mp} + B_{pp} + B_{pm} \tag{4.6}$$

Here, B_{mp} is the broadcasting time of input data from the memory to PEs. It consists of 2 cycles of initial arbitration and the transfer time of input data. Similar to the NoC-based network, the transfer time of input data is overlapped with the calculation time of the neurons. A neuron starts its operation right after it receives the first input data value.

B_{pp} is the broadcasting time of intermediate results among PEs at all hidden layers. The data transfer time can be overlapped with the calculation time of the neurons, hence the broadcasting time of each PE only includes 2 cycles of arbitration. For a bus-based network with p_h PEs running in the hidden layers, the total broadcasting time in each layer is $2p_h$.

B_{pm} is the sum of the output transfer times from each PE to the memory at the output layer. The transfer time of V output values from each PE to the memory is $2 + V$. Therefore, for a bus-based network of p_o PEs running in the output layer, $B_{pm} = p_o(2 + V)$.

In summary, the total bus latency is almost proportional to the number of PEs attached to it, which is calculated as:

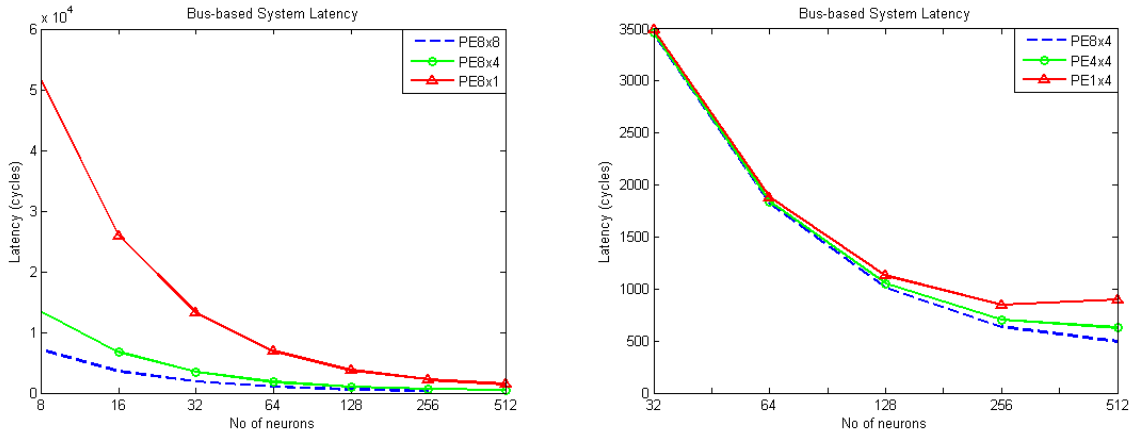
$$B = 2 + 2p_h + (2 + V)p_o \tag{4.7}$$

The latency for recognizing one handwritten digit of different bus-based design configurations and running neurons is as shown in Fig. 4.8. It can be seen that similar to the NoC-based network, the latency decreases with an increase in the number of running neurons as shown in Fig. 4.8 (a) and an increase in the number of parallel multipliers in each neuron as shown in Fig. 4.8 (b). This is because with more parallel neurons in a single PE, the number of used PEs is smaller. This leads to lower latency overall as the bus latency increases proportionally to the increase in the number of PEs. However, the latency slightly increases between the networks of 256 and 512 PE(4×1). This is because the decrease in calculation time is smaller than the increase in bus latency between these two networks. It proves that the choice of PE configurations (i.e number of parallel multipliers per neurons and the number of neurons per PE) has a significant impact on the overall network latency. The PE configuration should be chosen with high parallelism in order to increase the calculation speed and reduce the bus latency.

The smallest latency of 383 cycles is achieved by the bus-based network of 32 PEs(8×8). Running at 100MHz, it takes about $3.83\mu s$ for 1 digit recognition or 38.3ms for a test vector set of 10,000 digits, which is 28-times faster than an i7-CPU running Matlab.

4.4.3 Performance

The performance of the NoC-based and bus-based designs in terms of fixed-point operations per cycle (FPOPs/cycle) with different PE configurations and running neurons is as shown in Fig. 4.9 and Fig. 4.10 respectively. Each handwritten digit recognition by the NN takes $2 \times (784 \times 512 + 512 \times 10) = 813056$ fixed-point operations. With the total



(a) (b)

Figure 4.8: Latency comparison of different bus-based design configurations.

latency cycles (L) obtained from simulation, the performance of the design is $813056/L$ (FPOPs/cycle). It can be seen that the performance increases with an increase in the number of running neurons and the PE configuration $n \times m$.

The highest performance achieved by the NoC-based and bus-based system are 1339 and 2122 FPOPs/cycle respectively. Both are with the network of 32 PEs(8×8). Running at 100MHz, the NoC-based network of 32 PEs(8×8) can perform 133.9 billions FPOPs per second, which is much higher than the reported performance of 500 millions operations per second on a regular NoC by (Dong *et al.*, 2010).

The NoC-based system of 64 PEs(8×4) achieves a performance of 1011 FPOPs/cycle. This system has the same parallelism degree of 2048 (i.e. the total of parallel multipliers in the system) with the system of 32 PEs(8×8). However, the NoC size in a 64-PE network is double that of a 32-PE network, which results in much higher NoC latency in the 64-PE system compared with the 32-PE system. Hence the system of 32 PEs(8×8) has higher performance than the system of 64 PEs(8×4). Similar to the NoC-based systems, the bus-based system of 32 PEs(8×8) also have higher performance than those of the two systems, 64 PEs(4×8) and 64 PEs(8×4) due to smaller bus latency of a 32-PE system compared to a 64-PE system. In addition, the bus-based systems have higher performance than the same NoC-based system configuration as shown in Fig. 4.11. In order to obtain a complete conclusion on which system is the best implementation,

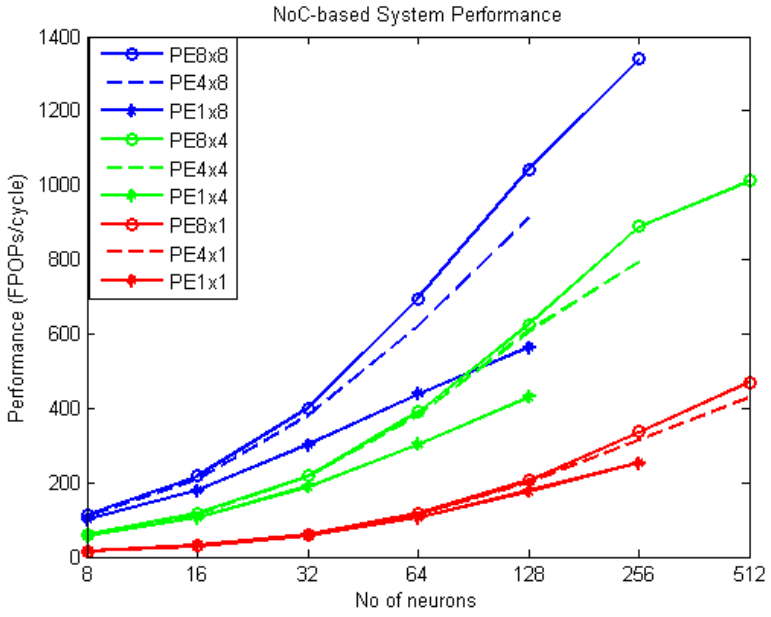


Figure 4.9: Performance comparison of different NoC-based design configurations.

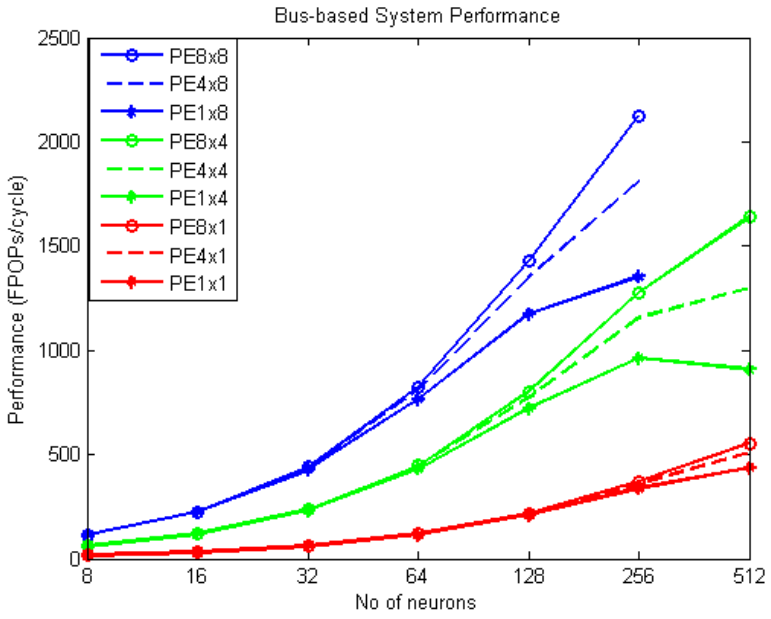


Figure 4.10: Performance comparison of different bus-based design configurations.

it is necessary to consider the performance in conjunction with the area and power consumptions of these systems.

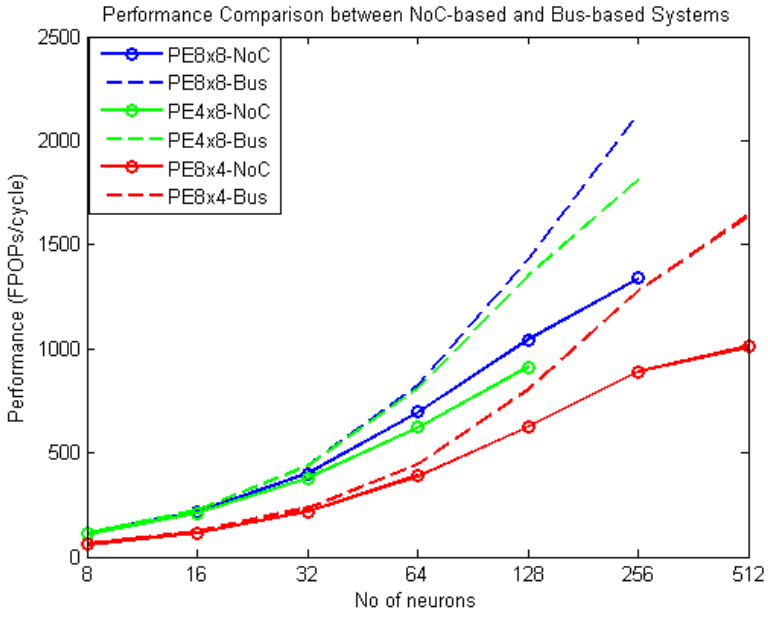


Figure 4.11: Performance comparison of NoC-based and bus-based systems.

Table 4.5 shows the comparison of performance, area and power consumptions of the five systems, 32 NoC-based PEs(8×8), 64 NoC-based PEs(8×4), 32 bus-based PEs(8×8), 64 bus-based PEs(8×4) and 64 bus-based PEs(4×4). These systems are the biggest designs that can be implemented on FPGA with these PE configurations and they have the same parallelism degree. Resources usage here is in terms of percentage of the available resources on the FPGA. It can be seen from the table that both the systems of 32 PEs(8×8) with NoC and bus interconnects consume less hardware resources and power while providing better performance than the other systems using the same interconnect architecture. It can be concluded that among designs with the same parallelism degree, the design with smaller NoC/bus size is a better solution.

It can also be seen from the Table 4.5 that the bus-based systems consume less hardware resources and power and also provide better performance than the NoC-based systems with the same size and PE configuration. A bus-based interconnect is a better choice for the fully-connected feed-forward NN in this case study. The broadcasting communication is more favorable for a bus than a NoC as all PEs connect to the same bus. For partially-connected NNs or other kind of NNs, the results might be different.

It can also be seen that the systems with higher parallelism in PE configurations have lower frequencies than the larger systems with lower parallelism in PE configurations. Therefore, high parallelism in PE configurations can provide smaller latency but can also reduce the system frequency. Therefore, the selection of PE configurations should take into consideration the system latency-frequency trade-off.

Table 4.5: Performance, area and power consumptions of different designs.

	32 PEs(8x8)-NoC	64 PEs(8x4)-NoC	32 PEs(8x8)-Bus	64 PEs(4x8)-Bus	64 PEs(8x4)-Bus
Registers	14.4	18.8	11.6	11.9	14.1
LUTs	64.6	83.5	50	49.4	62.1
BRAMs	87	87	56	62	56
DSPs	73	73	73	73	73
Post implementation frequency (MHz)	147	151	144	146	149
Estimated power at 100MHz (W)	7.037	7.761	5.179	5.099	5.448
Performance (FPOPs/cycle)	1339	1011	2122	1737	1639
Performance/Power (FPOPs/cycle/W)	228	130	409	340	300

4.5 Conclusion

This chapter has implemented and examined NoC and bus interconnects for a neural network implementation on FPGAs. The NoC and bus are customized to best suit the massive communication requirements of the NN. The implementation demonstrates that the use of both interconnects in a NN brings scalable performance at an acceptable cost of additional on-chip hardware. In these systems, each processing elements consists of multiple neurons for efficient use of on-chip memory and routing resources. Each neuron performs multiple parallel multipliers and adders. Experimental results show that PE configuration has a significant impact on the overall network latency and operating frequency. Higher parallelism in PE configurations often provide faster calculation speed and smaller communication latency but can reduce the system operating frequency.

The designs can be used for different applications with different network configurations. In this chapter, an example of a NN for handwritten digit recognition has been implemented on a Xilinx Virtex-7 XC7VX485T FPGA. Area and power consumption as well as latency and performance of the design have been evaluated. Running at 100MHz, the best NoC-based and bus-based designs need only 6.07μs and 3.83μs respectively for 1 digit recognition, which is respectively 18-time and 28-time faster than an i7-3.4GHz CPU running Matlab. The best NoC-based design provides the highest performance of 1339 FPOPs/cycle or 133.9 billions FPOPs per second at 100MHz, which is much higher than the published performance on regular NoCs. The highest performance of 2122 FPOPs/cycle of the bus-based design is even higher. Therefore, these designs are especially suitable for high-speed accelerators of real-time applications.

The bus-based fully-connected feed-forward NNs provide better performance and consume less area and power than the NoC-based designs for the case study of handwritten digit recognition. The broadcasting communication is more favorable for a bus than a NoC. For partially-connected NNs or other kind of NNs, the results might be different. In addition, the bus-based interconnects suffer from lower speed when the design gets bigger. The NoC-based interconnects are more sustainable and scalable. Therefore,

a bus is a better choice for similar NN configurations like the one in this chapter. For even larger NN applications on larger FPGAs, a NoC is still a potential solution.

Combining with the results for the FFT case study in Chapter 3, it can be concluded that the most important factor that need to be considered when choosing a communication architecture for a system is its communication patterns rather than the size of the system. There is a common thought that NoCs are better for large systems than buses. However, for the case of the FFT systems with only 16 PEs, the performance achieved by a NoC-based system is better than a bus-based system while for the NN systems of 32 PEs, the performance achieved by a NoC-based system is worse than a bus-based system. This is because the FFT requires multiple simultaneous communications, which a NoC can do better while the broadcast communication in a NN is more favorable for a bus.

Area and power consumption is also an important factor when selecting a communication architecture. A bus often consumes lower area and power consumption than a NoC. However, when considering the area and power consumption in the context of an entire system, this advantage of the bus can be insignificant, as was seen in the case of the FFT. Therefore, it is necessary to consider and compare the area and power consumption in the context of the entire system to obtain a complete comparison.

Chapter 5

Dynamically Partially Reconfigurable System Implementations

THIS chapter presents the dynamically partially reconfigurable implementations for the FFT and the NN systems presented in Chapter 3 and Chapter 4. Experiments are carried out to measure resource overhead, time and power consumption of the reconfiguration process. Comparisons between dynamic and static implementations are made to highlight the advantages and disadvantages of dynamic partial reconfiguration. Based on experiments with some current FPGA devices, limitations of current FPGA architectures and design flows for partial reconfiguration are also discussed. In addition, the benefits and limitations of NoC and bus interconnect architectures for dynamic partial reconfigurable systems are also stated.

5.1 Partial Reconfiguration Design Flow

The partial reconfiguration design flow and practical experiments in this chapter are based on Xilinx FPGAs as they are the most popular commercial FPGAs providing partial reconfiguration with fully supported documentation and design tools. They are also the main platform used in much of the research on partial reconfiguration. The particular Xilinx FPGAs used is a Virtex-7 XC7VX485T device. The Virtex-7 series is one of the latest series supporting partial reconfiguration. The transfer of the design flow from this FPGA series to others can be easily performed by software tools such as a tool called Dreams presented in (Otero *et al.*, 2012).

Xilinx presents two design flows for partial reconfiguration: module based and difference based partial reconfiguration (Xilinx, 2004). The difference based approach is suitable for designs with only small changes. A bitstream to reconfigure the FPGA is generated based on the difference between two designs. This approach allows a fast switching from one implementation to another as the difference bitstream is smaller than an entire device bitstream.

For designs with large logic blocks to be configured, a module-based approach is required. In this approach, the FPGA is divided in to distinct portions referred to as reconfigurable modules or reconfigurable regions. The size of a reconfigurable region depends on the size of different modules that can be implemented in that region. However, there are some restrictions on the shape and the minimum size of a reconfigurable region. A reconfigurable region needs to have a rectangular shape. The smallest reconfigurable region must be equal to a reconfigurable frame of the FPGA. Reconfigurable frames are the minimum reconfigurable building blocks for performing partial reconfiguration. They cannot be split any smaller. Even if a region smaller than a single reconfigurable frame is chosen, the entire frame is reconfigured. The reconfigurable frame size varies among different FPGA device series.

For communications between reconfigurable modules and between reconfigurable and static modules in the module-based partial reconfiguration approach, a special so-called bus macro is used. These bus-macros are fixedly implemented in the reconfigurable regions and they are not affected during the partial reconfiguration. The bus macros allow signals to cross over the reconfigurable region boundaries and establish

unchanging routing channels between modules to guarantee the correct communications between these modules.

For old Xilinx FPGAs such as the Virtex-II the bus macros were implemented using tristate buffers. A reconfigurable frame was required to span over the full height of the device (Donthi and Haggard, 2003). In later Xilinx families that no longer supported tristate buffers such as the Virtex-4, the bus macros were implemented with logic slices. A slice-based bus macro consisted of 8 CLBs (Configurable Logic Blocks), 4 in the reconfigurable region and 4 in the static region (Lysaght *et al.*, 2006). In the Vixter-4, a minimum reconfigurable frame height was no longer restricted to the full height of the device. It was 16 CLBs high. A partial reconfiguration design flow with the use of slice-based bus macros was presented in (Xilinx, 2006)

In one of the latest Xilinx published documents on partial reconfiguration (Xilinx, 2013), a bus macro is now called a partition pin. Each partition pin is efficiently implemented by a 1-input LUT called a proxy logic. A proxy logic is automatically inserted by the software for each partition pin. It is a fixed and known interface point between static and reconfigurable regions. An illustration of timing paths to and from a reconfigurable partition through partition pins is given in Fig. 5.1. Path A is a path from a static net input to a partition pin. Path B is a reconfigurable net output from a partition pin. Path C is a reconfigurable net input to a partition pin. Path D is a static net output from a partition pin. Paths X, Y and Z are the register-to-register paths that contain a partition pin in the path.

A partial reconfiguration design flow with the use of partition pins based on the PlanAhead software design tool is presented in (Xilinx, 2012b). It is a module-based approach with 12 steps as shown in Fig. 5.2

5.2 Xilinx 7-Series FPGA Architecture

The Xilinx 7-series are heterogeneous FPGAs that comprise columns of different resources: CLBs, BRAMs, DSPs, I/O, CMT (Clock Management Tile), FIFO logic, BUFG (Global Clock Buffer), MGT (Multi-Gigabit Transceiver). An illustration of a 7-series FPGA architecture is given in Fig. 5.3.

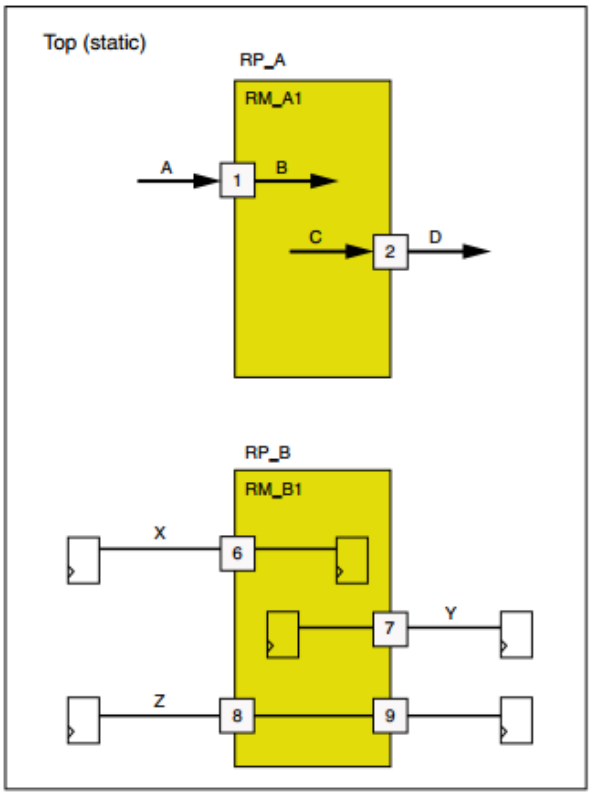


Figure 5.1: Timing paths to and from a reconfigurable partition through partition pins (Xilinx, 2013).

In addition to the column-based resource arrangement, the 7-series architecture is also divided into several rows. The Virtex-7 XC7VX485T device has 7 rows as shown in Fig. 5.4. The height of a row is equal to the height of a reconfigurable frame. A row by a column of a resource type is a reconfigurable frame. A reconfigurable frame of the 7-series is 50 CLBs high by a CLB wide. Similar reconfigurable frames exist for different resource types such as BRAMs and DSP48s. The amount of some major resources types and their arrangement in rows and columns in a Virtex-7 XC7VX485T device are summarized in Table 5.1.

It is noted that reconfigurable frames here are different from reconfiguration frames that are the smallest addressable segments of the FPGA configuration memory space. For the 7-series, a configuration frame contains 101 32-bit words (Xilinx, 2017a). Reconfigurable frames are built upon these configuration frames.

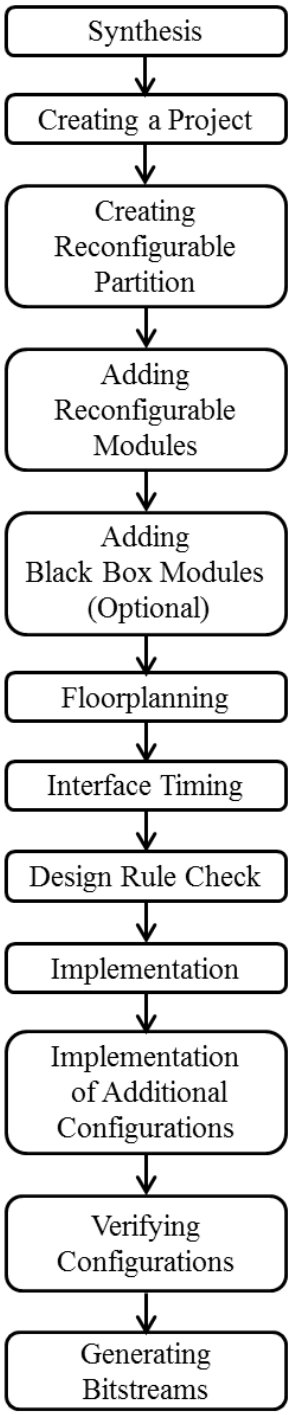


Figure 5.2: Dynamic partial reconfiguration design flow (Xilinx, 2012b).

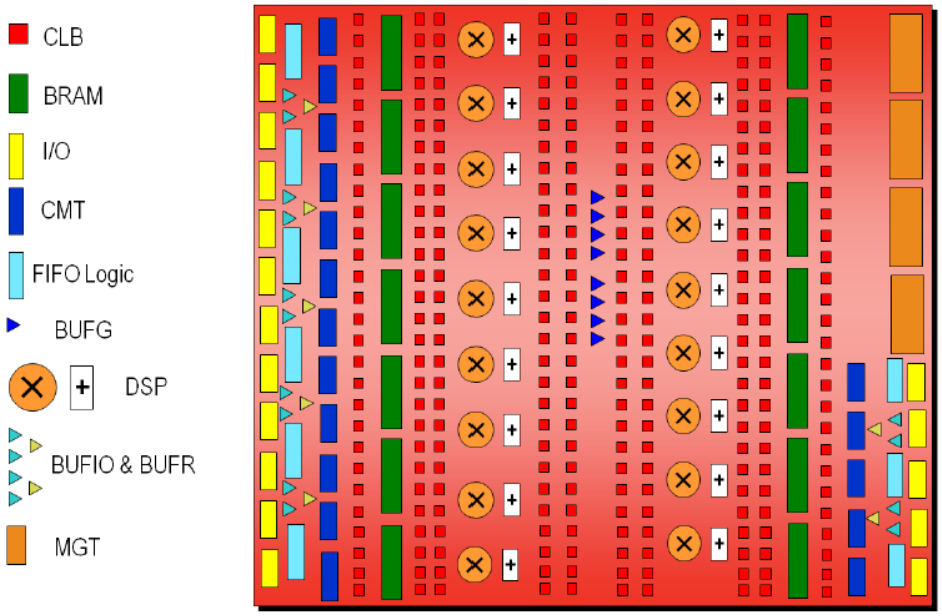


Figure 5.3: Xilinx 7-series FPGA architecture overview (Xilinx, 2016).

Table 5.1: Amount of resources and their arrangement in a Virtex-7 XC7VX485T FPGA

	CLBs	BRAMs	DSP48s
Total amount	37950	1030	2800
Number of columns	113	15	20
Amount per column	350	70	140
Amount per frame	50	10	20

5.3 Dynamic Partial Reconfiguration Implementations for FFT and NN Systems

In this section partial reconfiguration implementations based on the Xilinx design flow for the FFT and NN systems in the previous chapters are presented.

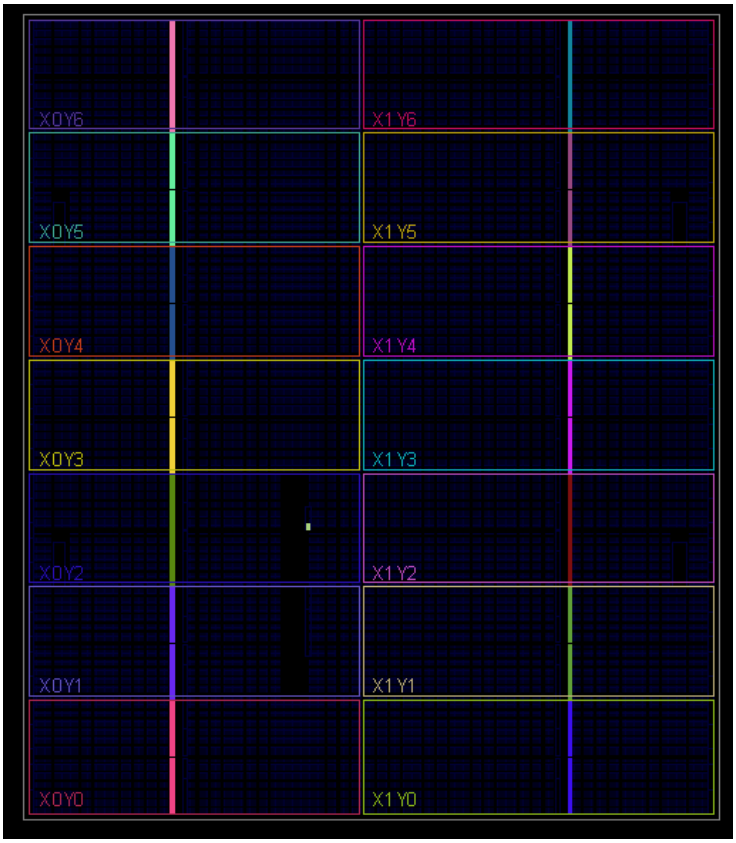


Figure 5.4: A Virtex-7 XC7VX485T row and column arrangement.

5.3.1 Partitioning

To support dynamic partial reconfiguration, the system is divided into two areas: a static area with functionality unchanged during system operation; and a dynamic area. An illustration of the NoC-based system is shown in Fig. 5.5. The partial reconfiguration approach for the NoC-based systems is based on the approach for DRNoC (Krasteva *et al.*, 2010). The dynamic area is divided into partial reconfiguration regions or partitions which can be configured as a PE or a router. Each region must contain sufficient resources to implement the modules assigned to it.

For the bus-based system as shown in Fig. 5.6, only PEs are implemented in the dynamic area. The bus logic including its arbiter and decoder are implemented in the static area. The connection points between reconfigurable PEs and the bus logic are implemented using the partition pins proposed by Xilinx instead of embedded macros

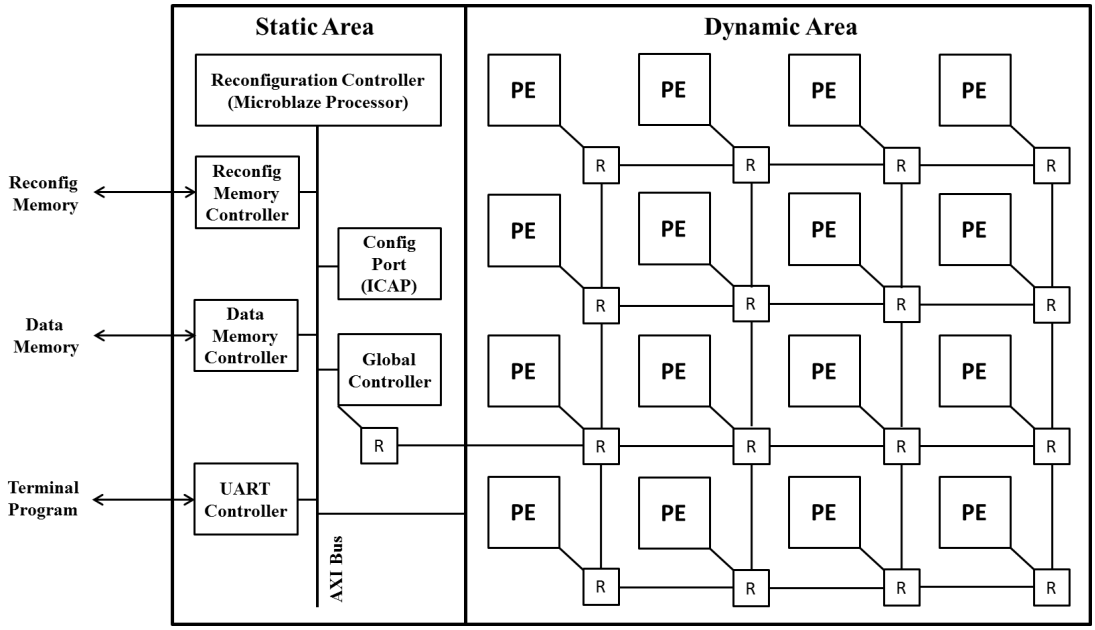


Figure 5.5: Dynamically reconfigurable NoC-based architecture.

(Koester *et al.*, 2011) or IO bars (Koch *et al.*, 2008b) with homogeneous bus architectures due to the following reasons:

1. The use of partitions pins is more resource utilization efficient. Only 1 LUT-1 is required for each 1-bit signal. An embedded macro requires 3 LUTs and 3 registers for each 1-bit bidirectional signal.

2. Each PE in the FFT and NN systems is a master and also a slave of other PEs. The homogeneous approaches were proposed and optimized for reconfigurable slave modules only. For reconfigurable master modules, these approaches come along with additional latency and more complex logic (Koch *et al.*, 2008b).

3. A point-to-point connection between a reconfigurable module and another module through their partition pins results in smaller latency than a shared connection that passes through the embedded macros/IO bars of all reconfigurable modules in the system.

Based on the user's power-performance requirement, a different number of PEs can be implemented in the dynamic region. With dynamic partial reconfiguration, time

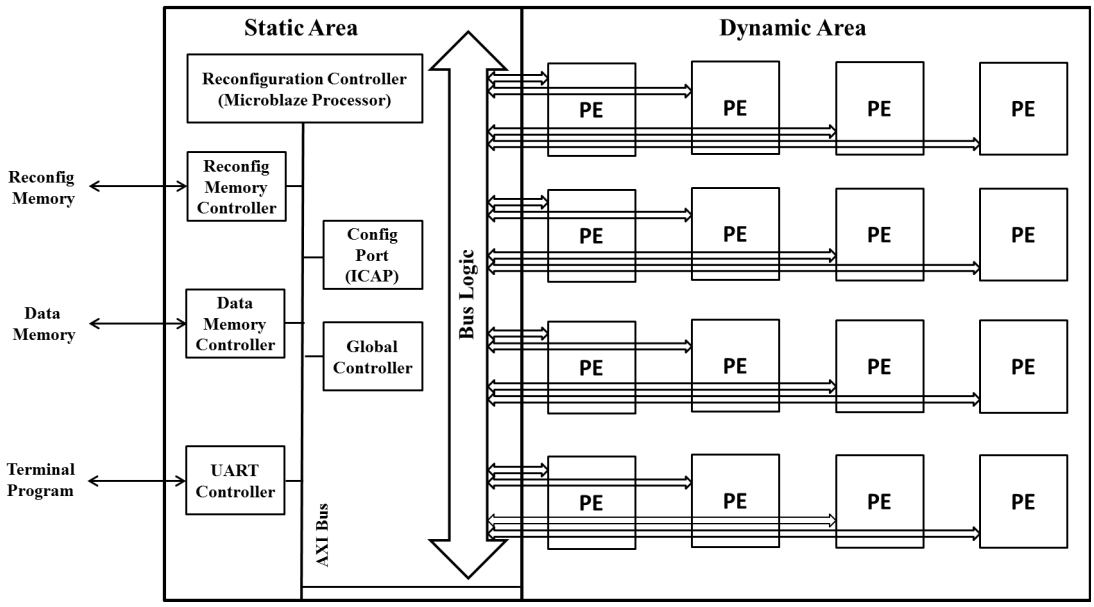


Figure 5.6: Dynamically reconfigurable bus-based architecture.

and power consumption for the reconfiguration process can be reduced significantly and the system can keep running during reconfiguration. For example, when the FFT system is running with 4 PEs and the user wants to upgrade to 8 PEs to achieve higher calculation speed, only 4 additional PEs are configured. This can save half of the time and power consumption required to fully configure a system with 8 PEs. In addition, the 4-PE design can keep running until the 8-PE design is completely configured. It is much simpler when the design is changed from a large number of PEs to a smaller one. In this case, the unused PEs are reconfigured with blanking bitstreams.

The static area contains a configuration controller, a configuration port, interfaces with external memories, a serial interface with an external host, and the global controller. The reconfiguration controller is responsible for loading the partial configuration bitstreams stored in the configuration memory through the configuration port. It can be conveniently implemented using a MicroBlaze soft-core processor on the FPGA. The configuration port transmits the configuration data to the assigned region. The internal configuration access port (ICAP) primitive is used in this case since it provides access to the configuration logic of the FPGA from within the FPGA fabric. The terminal program with the external host allows users to select the number of running PEs. The

global controller is responsible for data scheduling and informing PEs when the design is changed.

5.3.2 Floorplanning

Floorplanning involves placing reconfigurable modules into the physical fabric of the FPGA. Careful floorplanning can have a significant positive impact on the system routing, timing and reconfiguration overhead. This section focuses on floorplanning of the reconfigurable modules in the dynamic region only. Floorplanning of the modules in the static is automatically performed by the software tool to achieve optimal routing and timing constraints.

To ensure the correctness of routing in a partially reconfigurable system, a reconfigurable region must have a rectangular shape. This leads to an inefficient utilization of hardware resources especially when there are many reconfigurable regions and each region has different kinds of resources. The more reconfigurable regions and heterogeneous resource types in each region, the more inefficient utilization of hardware resources.

Table 5.2 shows the resources of a physical reconfigurable region that covers all resources necessary for implementing a bus-compatible FFT PE in comparison with the resources actually used. It can be seen from the table that a region that fulfill the BRAMs requirement of a bus-compatible FFT PE leaves 75% of registers, 26% of LUTs and 75% of DSPs unused. The utilization is better in the case of a FFT router as its implementation only consists of registers and LUTs. However the utilization of registers is only around 20% for a full utilization of LUTs.

In addition to the restriction on the reconfigurable region shape, the 7-series has another restriction with interconnect tiles. These tiles are included due to the clock routing requirements in these devices (Xilinx, 2013). Interconnect tiles are structured horizontally and must not be split when defining the reconfigurable regions. With these restrictions, only 8 reconfigurable regions can be defined for the FFT PEs. In other terms, only 8 PEs can be implemented in the partially reconfigurable FFT systems in comparison with 16 PEs in the static systems. It is similar for the NN systems. Only 16 PEs(8×8) can be implemented in the partially reconfigurable NN systems.

Table 5.2: Physically defined resources vs. required resources in a reconfigurable region for a bus-compatible FFT PE.

Resource Types	Required	Available	Utilization Percentage
Resigters	8670	35328	25
LUTs	13039	17664	74
BRAMs	62	63	99
DSP48E1	36	144	25

In addition to the inefficient utilization of hardware resources, the restriction on the reconfigurable region shape also leads to higher requirements for reconfiguration memory and higher reconfiguration time. The bitstream size for a reconfigurable regions depends on the size of that region rather than the resources actually implemented; and the reconfiguration time depends on the bitstream size. The bitstream of a bus-based FFT PE with the reconfigurable region defined in Table 5.2 is 1212000 Bytes. It is noted that the result here is for a particular example of a PE. The results of other PEs can be different, depending on the size of the reconfigurable regions defined for them. As 7-series FPGAs are highly heterogeneous, it is impossible to define homogeneous regions for every PE in the systems.

5.4 Implementation Results

After floorplanning, the partially reconfigurable FFT and NN systems are implemented on the FPGA. Timing and power analysis is carried out to analyze the benefits and the costs of partial reconfiguration implementation. The comparison of NoCs and buses for partially reconfigurable systems is also performed based on these results. Results that compare the reconfiguration time and energy for different designs like this do not appear elsewhere in the pubic literature. The results in this thesis provide a useful guide for designers planning a dynamically partially reconfigurable system.

5.4.1 Benefits and Drawbacks of Partial Reconfiguration

Benefits

The benefit of implementing partial reconfiguration for the FFT and NN systems is that users can efficiently choose among different performance/power operating points at the minimum cost of reconfiguration. Without dynamic partial reconfiguration, the user can choose between two options. The first one is using a fixed design of the largest design of 8 PEs for the FFT and 16 PEs for the NN. This option is simple and does not require hardware reconfiguration but is not power-efficient since the biggest design is used for all FFT sizes. Fig. 5.7 and Fig. 5.8 respectively show the power consumption comparison of the static and dynamic FFT and NN systems with different communication architectures and number of running PEs. It can be seen from these figures that the power consumption of the largest static design is only smaller than the largest dynamic design and much higher than the rest of the dynamic design variations. Therefore, it is not power-efficient to use the largest static design for all performance/power requirements.

The second option is using a static design appropriate for each of the desired performance/power operating points. The system is fully reconfigured when changing between design variations. With this option, each design variation is smaller and more power-efficient than the corresponding variation with dynamic partial reconfiguration. However, the time and power consumption for reconfiguration process is higher, which is a disadvantage for real-time applications.

Drawbacks

The first drawback of partial reconfiguration implementation is the inefficient utilization of hardware resources as presented in Section 5.3.2. This drawback is mainly due to the heterogeneity of FPGAs and the restrictions of partial reconfiguration design flow.

Another drawback is the resource and power consumption overhead of partial reconfiguration. The resource overhead is for creating partition pins for all ports of reconfiguration modules in the systems. However, this resource overhead is insignificant as only one LUT-1 is required for a partition pin. The power consumption overhead in

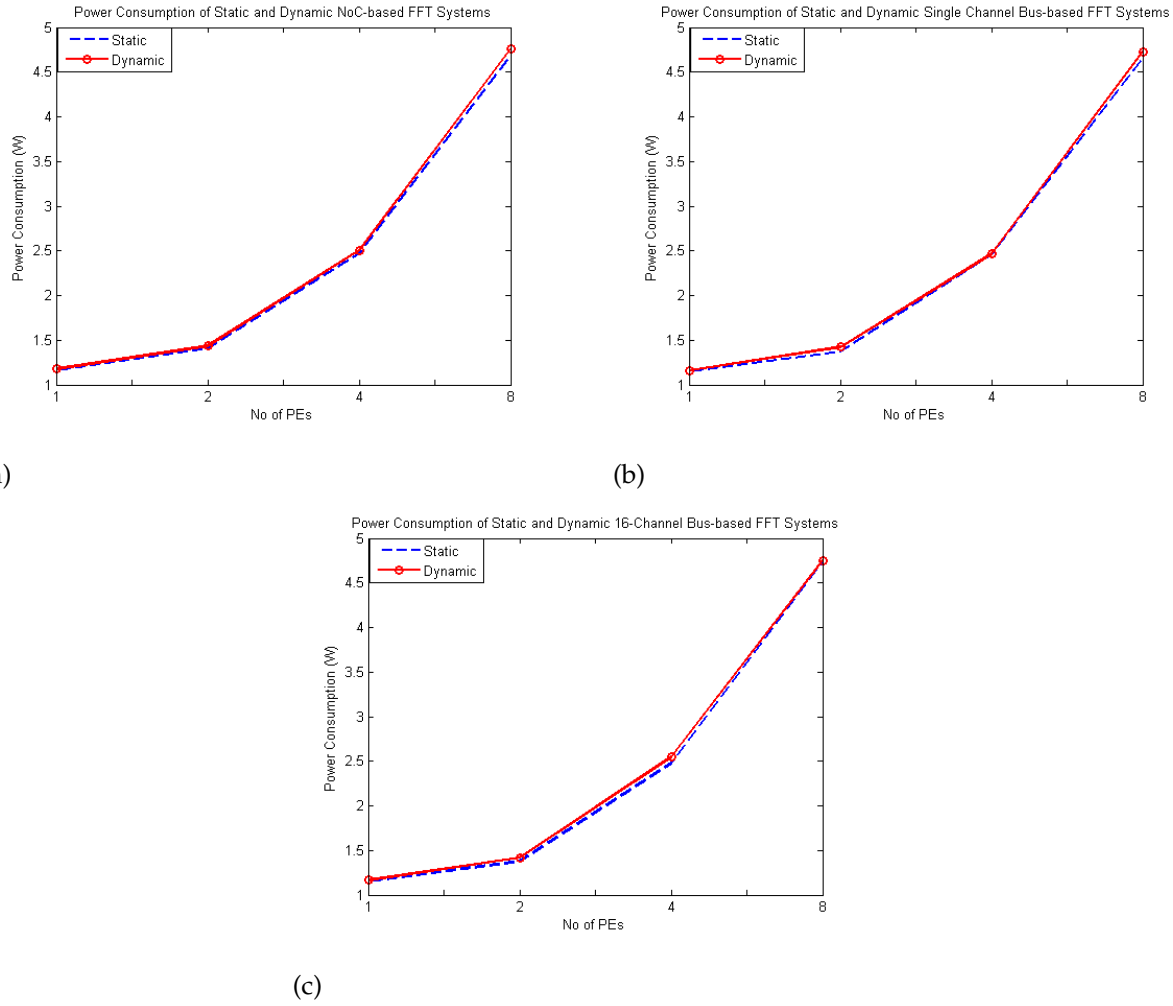
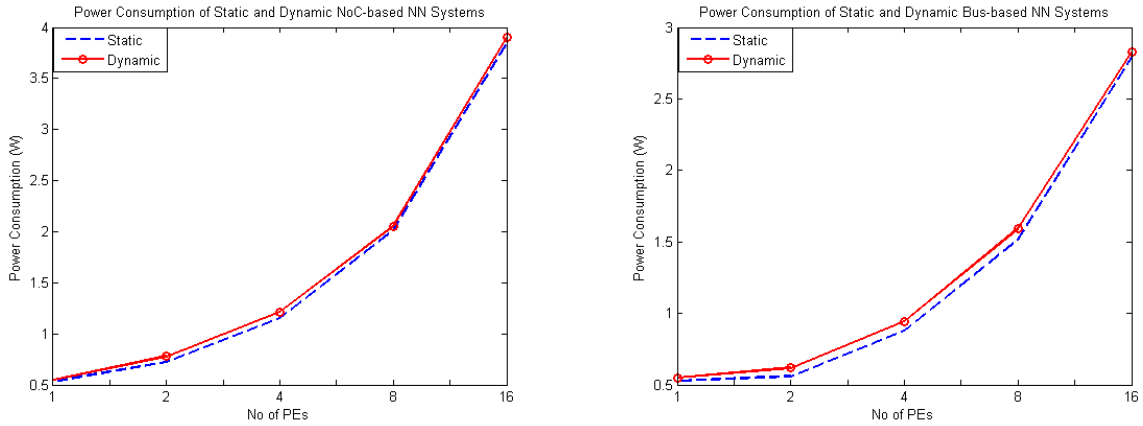


Figure 5.7: Power consumption of static and dynamic FFT systems. (a) NoC-based systems. (b) Single channel bus-based systems. (c) 16-channel bus-based systems.

this case is also insignificant. It can be seen from the Fig. 5.7 and Fig. 5.8, the power consumption of the dynamic systems is only slightly higher than the corresponding static systems. The power consumption overhead is less than 5% of the power consumption of the static system. This is because the power consumption depends on the actual logic implemented in the reconfigurable regions rather than the size of the reconfigurable regions.

The final drawback is the decrease in operating frequency of the dynamic systems compared with the static systems. Fig. 5.9 and Fig. 5.10 respectively show the maximum operating frequencies of the static and dynamic FFT and NN systems. It can be seen



(a) (b)

Figure 5.8: Power consumption of static and dynamic NN systems. (a) NoC-based systems. (b) Bus-based systems.

from these figures that the decrease in operating frequency of the dynamic system are more significant than the increase in power consumption. The more number of reconfigurable PEs in the system, the higher decrease in system operating frequency.

The percentage of frequency decrease of each system is given in Table 5.3. The highest decrease in frequency is with the NoC-based NN system of 16 PEs with a decrease of 47 MHz which accounts for 32% of the static system frequency. All dynamic systems of 1 PE have operating frequencies about 10% lower than the static system frequency. It is noted that the reported operating frequencies of dynamic systems here are not optimal results, as the system frequency is considerably affected by routing and routing is considerably affected by floorplanning. As the floorplanning process is manually performed by the designer, it is hard to obtain an optimal result especially for the systems with many reconfigurable regions. However, in general it can be concluded that the more partially reconfigurable modules in a system, the slower the system.

5.4.2 Comparison of NoCs and Buses for Partially Reconfigurable Systems

From the results above it can be seen that buses have the advantage of better resource utilization than NoCs as most of the bus logic is implemented in the static region while

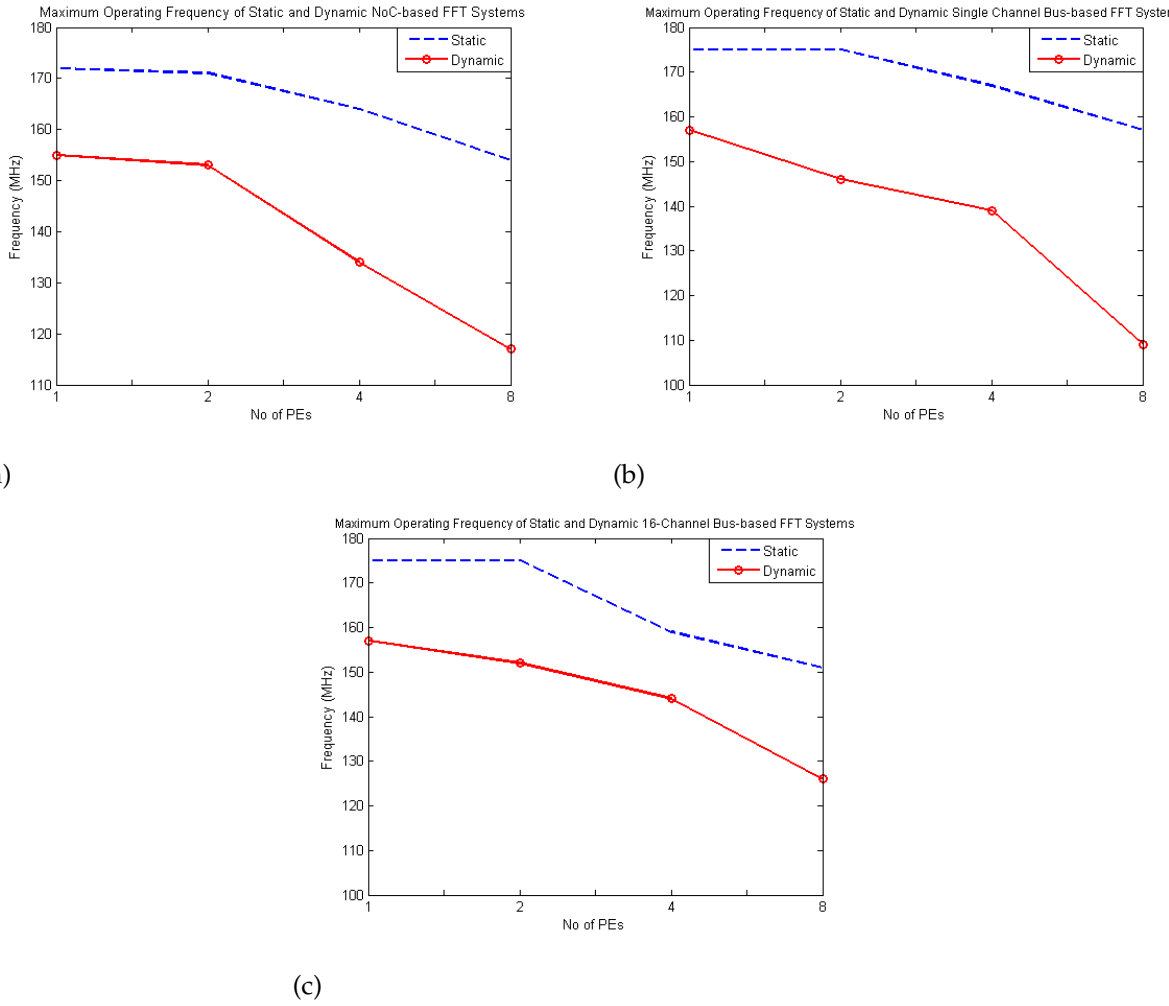
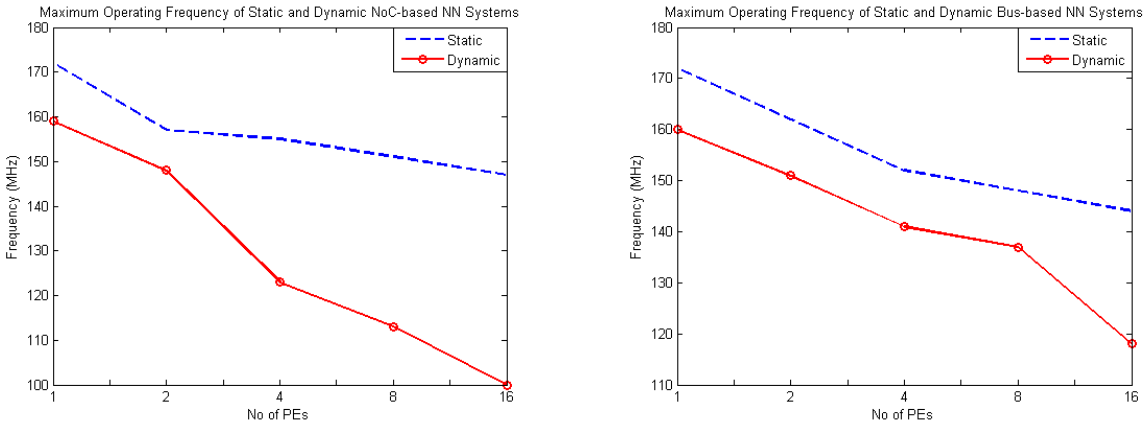


Figure 5.9: Maximum operating frequency of static and dynamic FFT systems. (a) NoC-based systems. (b) Single channel bus-based systems. (c) 16-channel bus-based systems.

all of the NoC is implemented in the dynamic region. However, this advantage comes from the restrictions of partial reconfiguration flow.

Another advantage of buses compared to NoCs is that they have smaller reconfiguration time and memory than NoCs as only PEs need to be reconfigured in a bus-based system, while both PEs and their attached routers needs to be reconfigured in a NoC-based system. For example, the bitstreams of a FFT PE and a FFT router are 1212000 Bytes and 132512 Bytes respectively. The reconfiguration memory and the reconfiguration time of a NoC-based FFT system of n PEs are about 1.11 times higher than those of the bus-based FFT systems. If the ICAP interface is 32-bits wide and clocked at 100



(a) (b)

Figure 5.10: Maximum operating frequency of static and dynamic NN systems. (a) NoC-based systems. (b) Bus-based systems.

Table 5.3: Percentage of frequency decrease compared with the static system frequency.

	1 PE	2 PEs	4 PEs	8 PEs	16 PEs
NoC-based FFT	9.88	10.53	18.29	24.03	
Single channel bus-based FFT	10.29	16.57	16.77	30.57	
16-channel bus-based FFT	10.29	13.14	9.43	16.56	
NoC-based NN	7.56	5.73	20.65	25.17	31.97
Bus-based NN	6.98	6.79	7.24	7.43	18.06

MHz, the configuration time of a single PE and a single router is 3.03 ms and 332 μs respectively. The partial reconfiguration time of n PEs is respectively $3.03n$ and $3.362n$ ms for the bus-based and NoC-based FFT systems. The bitstreams of a PE(8×8) and a 130-bit router in the NN system are 565600 Bytes and 206848 Bytes respectively, hence the reconfiguration memory and the reconfiguration time of a NoC-based NN system of n PEs are about 1.36 times higher than those of the bus-based NN systems.

However, NoCs have the advantage of flexibility and modularity. As the bus logic is mostly implemented in the static area, the bus needs to be implemented with a configuration in the static area that is large enough to be used for all system sizes. Due to the

modularity of the NoC routers, it is possible to store the bitstream of one router only, and this bitstream can be relocated for other routers in the network. Approaches for bitstream relocation have been proposed by many researchers such as (Corbetta *et al.*, 2009; Ichinomiya *et al.*, 2012; Lalevée *et al.*, 2016). With these approaches, the configuration memory for a NoC can even be smaller than the requirement for a bus as a router is often smaller than the entire bus logic.

5.4.3 Partial Reconfiguration Design Flow Challenges

The current design flow of partial reconfiguration is more complex and less automatic than the standard FPGA design flow. Some steps require designers to have a comprehensive knowledge of the target FPGA in order to obtain a successful implementation. The floorplanning step is an example. This step has an significant impact on the entire design performance and reconfiguration process. However, this step needs to be done manually by designers.

Some challenges are due to the strict architecture of the FPGA such as interconnect tiles in 7-series. Interconnect tiles are structured horizontally and must not be split when creating the reconfigurable regions. However, the granularity required cannot be visualized in the PlanAhead tool, and designers can only adjust the reconfigurable region size to avoiding splitting these interconnect tiles based on the guide of the implementation messages. This problem is significantly improved in another Xilinx partial reconfiguration design flow with Vivado Design Suite (Xilinx, 2017b) that supports 7-series and later series only. However, restrictions on reconfigurable region shape and manual floorplanning still exist. The steps of the partial reconfiguration design flow with Vivado Design Suite are similar with the one with PlanAhead.

Challenges in design flow can be a reason why partial reconfiguration is still rarely used in realistic applications. More effort is required to simplify and automate the design flow so that partial reconfiguration can be more widely used. In addition, some restrictions of partial reconfiguration implementation such as restricted shape of reconfigurable region should be improved to reduce the cost of partial reconfiguration implementation.

5.5 Conclusion

In this chapter, partial reconfiguration has been implemented for the FFT and NN systems in the previous chapters. The partial reconfiguration design is based on the Xilinx design flow with the PlanAhead software, targeting a Xilinx Virtex-7 XC7VX485T FPGA. With partial reconfiguration, a user can efficiently change the number of running PEs to choose an optimal power-performance operating point at the minimum cost of reconfiguration.

Based on the implementation results, there are some drawbacks that need to be considered before implementing a partially reconfigurable system. The major drawback is its resource utilization inefficiency due to the restricted rectangular shape of a reconfiguration region. In the case of FFT and NN systems, only half of the static design size can be implemented with partial reconfiguration. Another drawback is the power consumption overhead, however this overhead in insignificant. The observed power consumption overhead for the cases of FFT and NN systems is below 5%. The final drawback is the decrease in system operating frequency. It is observed that the more number of reconfigurable modules in a system the slower the system. The decrease in system operating frequency is quite considerable with the highest decrease of around 32% and the smallest decrease of around 7% compared to the static system frequencies for the cases of FFT and NN systems. Except for the power consumption overhead, the two major drawbacks of partial reconfiguration implementation are due to the restrictions of the design flow. More efforts is required to improve the reconfiguration design flow so that partial reconfiguration can be used more widely in realistic applications.

The partial reconfiguration design flow and implementation results have also pointed out the advantages and disadvantages of NoCs and buses for partially reconfigurable systems. Buses have the advantage of better resource utilization compared to NoCs as most of the bus logic is implemented in the static region while all the NoC is implemented in the dynamic region. They also have advantages of smaller reconfiguration time and memory than NoCs. However, most of the bus logic implemented in the static region makes the bus inflexible for a dynamic system. NoCs are more flexible and expansible. They do have the advantage of placing almost all of the communication infrastructure in the dynamic reconfiguration region. This means that different

applications running on the FPGA can use different interconnection strategies without the overhead of fixed bus resources in the static region. In addition, the modularity of NoC routers allows the ability to reduce the configuration memory requirement by using bitstream allocation approaches. With these approaches, the configuration memory for a NoC can even be smaller than the requirement for a bus as a router is often smaller than the entire bus logic.

Chapter 6

Thesis Conclusion

THIS chapter concludes the thesis with a summary of thesis contribution and significance, and proposes directions for future work and improvements.

6.1 Summary

Chapter 2 provided an overview of FPGAs and a comprehensive review of different NoC and bus interconnect architectures for on-chip systems in general and more specially for FPGA-based dynamically partially reconfigurable systems. This chapter reported different underlying programming technologies and architectures of commercial FPGAs. SRAM-based FPGAs were the main platform that was used for experiments and evaluation in this research due to their dominance in modern commercial FPGAs. This chapter pointed out that most of the studies on NoCs and buses to date focused on physical implementation and routing strategy. Few of the published studies demonstrated this in a fully realized practical application. In addition, little research had been carried out to examine and compare benefits and limitations of different communication architectures in partially reconfigurable systems as well as in general FPGA-based designs.

Chapter 3 presented the FFT implementations using different interconnection strategies on a Xilinx Virtex-7 XC7VX485T FPGA. NoCs and buses were compared in terms of area, power consumption and performance. The buses were implemented with a single channel and multiple channels for comparison. The NoC provided much better performance than the single channel bus and similar performance to the multi-channel bus in both parallel and parallel-pipelined FFT implementations. The only disadvantage of NoCs was their higher area and power consumption. However, this disadvantage was not significant when considered in the context of the entire system. The area and power consumption of a NoC-based system was just slightly higher than these of a single channel bus-based system but with much higher performance. This leaded to a better ratio of performance/power of the NoC-based system. The performance and power consumption results for NoCs and multi-channel buses were very similar; however a NoC-based systems has the advantage of of system scalability. The NoC-based interconnect approach provided scalable and sustainable performance when the design size increases. The best NoC-based parallel design provided smaller latency than Xilinx pipelined FFT cores. This demonstrated the advantage of using a NoC for a parallel FFT implementation to provide higher design parallelism and lower latency than the more common pipelined architecture in commercial FFT cores.

Chapter 4 implemented and examined NoC and bus interconnects for a neural network implementation on FPGAs. The NoC and bus were customized to best suit the massive communication requirements of the NN. The implementation demonstrated that the use of both interconnects in a NN brings scalable performance at an acceptable cost of additional on-chip hardware. In these systems, each processing elements consisted of multiple neurons for efficient use of on-chip memory and routing resources. Each neuron performed multiple parallel multipliers and adders. The designs could be used for different applications with different network configurations. In this chapter, an example of a NN for handwritten digit recognition was implemented on a Xilinx Virtex-7 XC7VX485T FPGA. Area and power consumption as well as latency and performance of the design were evaluated. Running at 100MHz, the best NoC-based and bus-based designs were respectively 18-time and 28-time faster than an i7-3.4GHz CPU running Matlab. The best NoC-based design provided the highest performance of 1339 FPOPs/cycle or 133.9 billions FPOPs per second at 100MHz, which was much higher than the published performance on regular NoCs. The highest performance of 2122 FPOPs/cycle of the bus-based design was even higher. The bus-based fully-connected feed-forward NNs provided better performance and consume less area and power than the NoC-based designs for the case study of handwritten digit recognition. However, the bus-based interconnects suffered from lower speed when the design got bigger. The NoC-based interconnects were more sustainable and scalable.

Chapter 5 implemented partial reconfiguration for the FFT and NN systems in the previous chapters. The partial reconfiguration design was based on the Xilinx design flow with the PlanAhead software, targeting a Xilinx Virtex-7 XC7VX485T FPGA. With partial reconfiguration, a user could efficiently change the number of running PEs to choose an optimal power-performance operating point at the minimum cost of reconfiguration. Based on the implementation results, there were some drawbacks that needed to be considered before implementing a partially reconfigurable system. The major drawbacks were its resource utilization inefficiency, power consumption overhead and decrease in system operating frequency. It was observed that a partially reconfigurable implementation could obtain around 50% of resource utilization with a power consumption overhead of below 5% and a decrease in frequency of up to 32% compared to the static system for the cases of FFT and NN systems. Except for the

power consumption overhead, the two major drawbacks of partial reconfiguration implementation were due to the restrictions of the design flow. This chapter also pointed out the advantages and disadvantages of NoCs and buses for partially reconfigurable systems. Buses had the advantages of better resource utilization and smaller reconfiguration time and memory than NoCs. However, NoCs were more flexible and expansible. They had the advantage of placing almost all of the communication infrastructure in the dynamic reconfiguration region. In addition, the modularity of NoC routers allowed the ability to reduce the configuration memory requirement by using bitstream allocation approaches. With these approaches, the configuration memory for a NoC could even be smaller than the requirement for a bus as a router is often smaller than the entire bus logic.

6.2 Conclusion

The results of this thesis provide some crucial conclusions and recommendations concerning the interconnect approach for FPGA-based designs in general and dynamically partially reconfigurable systems. Furthermore, these empirical results are important and beneficial for designers who consider whether to use partial reconfiguration for commercial designs.

Firstly, the experimental results of NoC and bus interconnect architectures for the cases of FFT and NN show that the most important factor that need to be considered when choosing a communication architecture for a system is its communication patterns rather than the size of the system. NoCs are suitable for systems with multiple simultaneous communications while buses are suitable for systems that do not require many simultaneous communications or systems with broadcast communications like a fully-connected feed-forward NN. The use of additional channels for a bus allows it to attain similar performance but also have similar power consumption to a NoC for systems with multiple simultaneous communications. However, NoCs are still more sustainable and scalable with system scalability than buses.

By using an appropriate communication architecture, a system can achieve better performance at an affordable cost of resource and power consumption. For example, the use of a NoC for a parallel FFT can achieve lower latency than the more common

pipelined architecture in commercial FFT cores. The use of a NoC with a routing algorithm customized for broadcast communications in a fully-connected feed-forward NN provides better performance with a regular NoC. The use of a bus for a NN is even better than a NoC and with a lower cost of resource and power consumption.

Area and power consumption is also an important factor when selecting a communication architecture. A bus often consumes lower area and power consumption than a NoC. However, when considering the area and power consumption in the context of an entire system, this advantage of the bus can be insignificant, as was seen in the case of the FFT. Therefore, it is necessary to consider and compare the area and power consumption in the context of the entire system to obtain a complete comparison.

In the context of dynamically partially reconfigurable systems, buses have the advantage of better resource utilization compared to NoCs as most of the bus logic is implemented in the static region while all the NoC is implemented in the dynamic region. They also have advantages of smaller reconfiguration time and memory than NoCs. However, having most of the bus logic implemented in the static region makes the bus inflexible for a dynamic system. NoCs are more flexible and expansible. In addition, some advantages of buses compared to NoCs are because of the limitations of partial reconfiguration design flow.

Secondly, the implementation of partial reconfiguration and its experimental results for the FFT and NN systems have clearly pointed out the advantages and disadvantages of partial reconfiguration implementation. Partial reconfiguration allows users to efficiently change the number of running PEs to choose an optimal power-performance operating point at the minimum cost of reconfiguration. However, this brings drawbacks including resource utilization inefficiency, power consumption overhead and decrease in system operating frequency. The experimental results report a 50% of resource utilization with a power consumption overhead of less than 5% and a decrease in frequency of up to 32% compared to a static implementation. The more reconfigurable regions in a system and the more heterogeneous resource types in each region, the more significant drawbacks the system suffers.

These results are very crucial for designers who consider whether to use partial reconfiguration for commercial designs. With current results, partial reconfiguration brings

more drawbacks than benefits for designs with many partial reconfigurable regions. For designs with only few of reconfigurable regions, partial reconfiguration is still beneficial. For example, the FFT and NN systems with only 1 or 2 reconfigurable PEs only incur a power consumption overhead of less than 5% and a decrease in frequency of around 10% compared to the static systems.

The results also show that most of the drawbacks of partial reconfiguration implementation come from the restrictions and limitations of partial reconfiguration design flow such as the manual floorplanning that need to performed by the designers and the restriction on rectangular shape of reconfigurable regions. The current design flow of partial reconfiguration is more complex and less automatic than the standard FPGA design flow. If these limitations can be addressed, partial reconfiguration should still be considered with its potential benefits.

6.3 Future Work

In this thesis, different interconnect architectures have been implemented and examined for FPGA-based designs in general and dynamically partially reconfigurable systems. Some topics and problems arose during the study, which could not be investigated due to the limited time.

1. The use of a bus for a fully-connected feed-forward NN provides better performance and consumes less area and power than a NoC. Future works will examine NoCs and buses for partially-connected NNs and other kind of NNs to obtain a more complete comparison. In addition, the use of both buses and NoCs in a NN brings scalable performance at an acceptable cost of additional on-chip hardware for the handwritten digit recognition. More complex NN case studies will be explored to demonstrate to benefits and limitations of these interconnect approaches compared to traditional NN implementations.

2. Floorplanning has a significant impact on the system routing, timing and reconfiguration overhead. An efficient floorplanning algorithm that takes into consideration the design requirements, the targeting FPGA devices and the timing constraints will be exploited for optimal implementation results.

3. The results of the FFT and NN implementations on FPGAs prove their ability for high-speed accelerators of realtime applications. Future works will demonstrate the extension use of mulitple FPGAs for supercomputing applications and investigate the communication architectures inter and between multiple FPGAs.

Bibliography

AGGARWAL-A., AND LEWIS-D. (1998). Routing Architecture for Hierarchical Field-Programmable Gate Arrays, *IEEE International Conference on Computer Design)*, pp. 475–478.

AHMAD-A., AMIRA-A., NICHOLL-P., AND KRILL-B. (2013). FPGA-based IP Cores Implementation for Face RecognitionUsing Dynamic Partial Reconfiguration, *Journal of Real-Time Image Processing*, **8**(3), pp. 327–340.

AHMAD-A., KRILL-B., AMIRA-A., AND RABAH-H. (2010). Efficient Architectures for 3D HWT Using Dynamic Partial Reconfiguration, *Journal of Systems Architecture*, **56**(8), pp. 305 – 316. Special Issue on HW/SW Co-Design: Tools and Applications.

AHMADINIA-A., BOBDA-C., DING-J., MAJER-M., TEICH-J., FEKETE-S. P., AND VAN DER VEEN-J. C. (2005). A practical approach for circuit routing on dynamic reconfigurable devices, *16th IEEE International Workshop on Rapid System Prototyping (RSP'05)*, pp. 84–90.

ALBRECHT-C., FOAG-J., KOCH-R., AND MAEHLE-E. (2006). DynaCORE - A Dynamically Reconfigurable Coprocessor Architecture for Network Processors, *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, pp. 101–108.

ALTERA. (2006). FPGA Architecture, White Paper, V1.0.

ARM. (2017). AMBA Speccifications.

AYINALA-M., AND PARHI-K. (2012). Parallel Pipelined FFT Architectures with Reduced Number of Delays, *Proceedings of the Great Lakes Symposium on VLSI*, GLSVLSI '12, ACM, New York, NY, USA, pp. 63–66.

AYORINDE-O., QI-H., HUANG-Y., AND CALHOUN-B. H. (2015). Using Island-Style Bi-directional Intra-CLB Routing in Low-Power FPGAs, *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–7.

BAIG-H., AND LEE-J. A. (2012). An Island-Style-Routing Compatible Fault-Tolerant FPGA Architecture with Self-Repairing Capabilities, *2012 International Conference on Field-Programmable Technology*, pp. 301–304.

BASHEER-I., AND HAJMEER-M. (2000). Artificial Neural Networks: Fundamentals, Computing, Design, and Application, *Journal of Microbiological Methods*, **43**(1), pp. 3 – 31. Neural Computting in Micrbiology.

BENINI-L., AND MICHELI-G. D. (2002). Networks on Chips: A New SoC Paradigm, *Computer*, **35**(1), pp. 70–78.

Betz, V.., Rose, J.., and Marquardt, A.. (eds.) (1999). *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, Norwell, MA, USA.

BOBDA-C., AHMADINIA-A., MAJER-M., TEICH-J., FEKETE-S., AND VAN DER VEEN-J. (2005). DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfugurable Devices, *Field Programmable Logic and Applications, 2005. International Conference on*, pp. 153–158.

BONAMY-R., CHILLET-D., BILAVARN-S., AND SENTIEYS-O. (2012). Power Consumption Model for Partial and Dynamic Reconfiguration, *2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1–8.

BORISOV-P., AND KUKENSKA-V. S. (2009). The Virtex-5 Routing and Logic Architecture.

BRIGHAM-E. O. (1988). *The Fast Fourier Transform and Its Applications*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

BROWN-S., AND ROSE-J. (1996). Architecture of FPGAs and CPLDs: A Tutorial, *IEEE Design and Test of Computers*, **13**, pp. 42–57.

BUI-T. T. T. (2017). Open Source, https://bitbucket.org/thanhthanh037/open_source.

BUI-T. T. T., PHILLIPS-B., AND LIEBELT-M. (2016). A Dynamically Reconfigurable NoC for Double-Precision Floating-Point FFT on FPGAs, *IARIA CENICS 2016, The Ninth International Conference on Advances in Circuits, Electronics and Micro-electronics*, pp. 52–57.

CANALS-V., MORRO-A., OLIVER-A., ALOMAR-M. L., AND ROSSELLÓ-J. L. (2016). A New Stochastic Computing Methodology for Efficient Neural Network Implementation, *IEEE Transactions on Neural Networks and Learning Systems*, **27**(3), pp. 551–564.

CHANG-Y. N., AND PARHI-K. (2003). An Efficient Pipelined FFT Architecture, *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, **50**(6), pp. 322–325.

CORBETTA-S., MORANDI-M., NOVATI-M., SANTAMBROGIO-M. D., SCIUTO-D., AND SPOLETINI-P. (2009). Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **17**(11), pp. 1650–1654.

CORDAN-B. (1999). An Efficient Bus Architecture for System-on-Chip Design, *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference (Cat. No.99CH36327)*, pp. 623–626.

DENNL-C., ZIENER-D., AND TEICH-J. (2012). On-the-fly Composition of FPGA-Based SQL Query Accelerators Using a Partially Reconfigurable Module Library, *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 45–52.

DEVAUX-L., AND PILLEMENT-S. (2014). OCEAN, A Flexible Adaptive Network-on-Chip for Dynamic Applications, *Microprocessors and Microsystems*, **38**(4), pp. 337 – 357.

DIGUET-J. P., STRUM-M., GRIGUER-N. L., CAETANO-L., AND SEPÚLVEDA-M. J. (2013). Scalable NoC-based Architecture of Neural Coding for New Efficient Associative Memories, *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 1–9.

DONG-Y., LI-C., LIN-Z., AND WATANABE-T. (2010). Multiple Network-on-Chip Model for High Performance Neural Network, *Journal of Semiconductor Technology and Science*, **10**(1), pp. 28–36.

DONTHI-S., AND HAGGARD-R. L. (2003). A Survey of Dynamically Reconfigurable FPGA Devices, *Proceedings of the 35th Southeastern Symposium on System Theory, 2003.*, pp. 422–426.

ENRIGHT-N., AND PEH-L. S. (2009). *On-Chip Networks*, Morgan Claypool.

FAWCETT-B. K. (1994). Taking Advantage of Reconfigurable Logic, *Proceedings Seventh Annual IEEE International ASIC Conference and Exhibit*, pp. 227–230.

GARRIDO-M., GRAJAL-J., SANCHEZ-M., AND GUSTAFSSON-O. (2013). Pipelined Radix- 2^k Feedforward FFT Architectures, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, **21**(1), pp. 23–32.

GRAND VIEW RESEARCH. (2016). Field Programmable Gate Array (FPGA) Market Analysis By Technology (SRAM, EEPROM, Antifuse, Flash), By Application (Consumer Electronics, Automotive, Industrial, Data Processing, Military Aerospace, Telecom), And Segment Forecasts, 2014 - 2024, *Technical report*.

HAGEMEYER-J., KETTELHOIT-B., AND PORRMANN-M. (2006). Dedicated module access in dynamically reconfigurable systems, *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pp. 1–8.

HEMMERT-K. S., AND UNDERWOOD-K. D. (2005). An Analysis of the Double-Precision Floating-Point FFT on FPGAs, *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, pp. 171–180.

HE-S., AND TORKELSON-M. (1998). Design and Implementation of a 1024-Point Pipeline FFT Processor, *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*, pp. 131–134.

HIMAVATHI-S., ANITHA-D., AND MUTHURAMALINGAM-A. (2007). Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization, *IEEE Transactions on Neural Networks*, **18**(3), pp. 880–888.

HUANG-G.-B., AND BABRI-H. A. (1998). Upper Bounds on the Number of Hidden Neurons in Feedforward Networks with Arbitrary Bounded Nonlinear Activation Functions, *IEEE Transactions on Neural Networks*, **9**(1), pp. 224–229.

ICHINOMIYA-Y., AMAGASAKI-M., IIDA-M., KUGA-M., AND SUEYOSHI-T. (2012). *A Bitstream Relocation Technique to Improve Flexibility of Partial Reconfiguration*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 139–152.

JOVANOVIC-S., TANOUGAST-C., WEBER-S., AND BOBDA-C. (2007). CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs, *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pp. 753–756.

JOVANOVIC-Z., AND MILUTINOVIC-V. (2012). FPGA Accelerator for Floating-Point Matrix Multiplication, Computers Digital Techniques, IET.

JUNG-Y., KIM-O., LEE-B., JUNG-H., AND RYOO-K. (2008). SoC Platform Design with Multi-Channel Bus Architecture, *2008 International SoC Design Conference*, Vol. 03, pp. III–48–III–49.

JUNG-Y., YOON-H., AND KIM-J. (2003). New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Applications, *Consumer Electronics, IEEE Transactions on*, **49**(1), pp. 14–20.

KOCH-D., BECKHOFF-C., AND TEICH-J. (2008a). ReCoBus-Builder - A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS, *2008 International Conference on Field Programmable Logic and Applications*, pp. 119–124.

KOCH-D., HAUBELT-C., AND TEICH-J. (2008b). Efficient Reconfigurable On-Chip Buses for FPGAs, *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pp. 287–290.

KOESTER-M., LUK-W., HAGEMEYER-J., PORRMANN-M., AND RUCKERT-U. (2011). Design Optimizations for Tiled Partially Reconfigurable Systems, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **19**(6), pp. 1048–1061.

KOH-S., AND DIESSEL-O. (2006). Comma: A communications methodology for dynamic module-based reconfiguration of FPGAs, *In ARCS Workshops 2006*, pp. 173–182.

KRASTEVA-Y. E., DE LA TORRE-E., AND RIESGO-T. (2010). Reconfigurable Networks on Chip: DRNoC Architecture, *Journal of Systems Architecture*, **56**(7), pp. 293 – 302. Special Issue on HW/SW Co-Design: Systems and Networks on Chip.

KUON-I., TESSIER-R., AND ROSE-J. (2008). FPGA Architecture: Survey and Challenges, *Foundations and Trends® in Electronic Design Automation*, **2**(2), pp. 135–253.

LAHIRI-K., RAGHUNATHAN-A., AND LAKSHMINARAYANA-G. (2006). The LOTTERYBUS On-Chip Communication Architecture, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **14**(6), pp. 596–608.

LAI-H., PAN-Y., LIU-Y., AND YAN-S. (2015). Simultaneous Feature Learning and Hash Coding With Deep Neural Networks, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

LALEVÉE-A., HORREIN-P. H., ARZEL-M., HÜBNER-M., AND VATON-S. (2016). AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs, *2016 Euromicro Conference on Digital System Design (DSD)*, pp. 14–21.

LECUN-Y., CORTES-C., AND BURGESE-C. J. (n.d.). The MNIST Database of Handwritten Digits, http://yann.lecun.com/exdb/mnist/. [Online; accessed 07-July-2017].

LYSAGHT-P., BLODGET-B., MASON-J., YOUNG-J., AND BRIDGFORD-B. (2006). Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs, pp. 1 – 6.

MAND-N. P., ROBINO-F., AND ÖBERG-J. (2012). Artificial Neural Network Emulation on NOC-based Multi-core FPGA Platform, *NORCHIP 2012*, pp. 1–4.

MISRA-J., AND SAHA-I. (2010). Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress, *Neurocomputing*, **74**, pp. 239–255.

MITIĆ-M., AND STOJČEV-M. (2006). An Overview of On-Chip Buses, *Facta universitatis - series: Electronics and Energetics*, **19**(3), pp. 405–428.

MORAES-F., CALAZANS-N., MELLO-A., MÖLLER-L., AND OST-L. (2004). HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip, *Integration, the VLSI Journal*, **38**(1), pp. 69 – 93.

ORLOWSKA-KOWALSKA-T., AND KAMINSKI-M. (2011). FPGA Implementation of the Multilayer Neural Network for the Speed Estimation of the Two-Mass Drive System, *IEEE Transactions on Industrial Informatics*, **7**(3), pp. 436–445.

OTERO-A., DE LA TORRE-E., AND RIESGO-T. (2012). Dreams: A Tool for the Design of Dynamically Reconfigurable Embedded and Modular Systems, *2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1–8.

PALMER-J., AND NELSON-B. (2004). A Parallel FFT Architecture for FPGAs, *in* J. Becker., M. Platzner., and S. Vernalde. (eds.), *Field Programmable Logic and Application*, Vol. 3203 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 948–953.

PASRICHA-S., AND DUTT-N. (2008). Chapter 2 - basic concepts of bus-based communication architectures, *in* S. Pasricha., ., and N. Dutt. (eds.), *On-Chip Communication Architectures*, Systems on Silicon, Morgan Kaufmann, Burlington, pp. 17 – 41.

PIONTECK-T., ALBRECHT-C., KOCH-R., AND MAEHLE-E. (2008). Adaptive Communication Architecture for Runtime Reconfigurablle System-on-Chips, *Parallel Processing Letters*, **18**(02), pp. 275–289.

RASTEGARI-M., ORDONEZ-V., REDMON-J., AND FARHADI-A. (2016). *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*, Springer International Publishing, Cham, pp. 525–542.

ROSE-J., GAMAL-A. E., AND SANGIOVANNI-VINCENTELLI-A. (1993). Architecture of Field-Programmable Gate Arrays, *Proceedings of the IEEE*, **81**(7), pp. 1013–1029.

SCHMIDHUBER-J. (2015). Deep Learning in Neural Networks: An Overview, *Neural Networks*, **61**(Supplement C), pp. 85 – 117.

SHANTHI-D., AND AMUTHA-R. (2011). Design of Efficient On-Chip Communication Architecture in MpSoC, *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, pp. 364–369.

SILVA-M. L., AND FERREIRA-J. C. (2008). Generation of partial FPGA configurations at run-time, *2008 International Conference on Field Programmable Logic and Applications*, pp. 367–372.

SUASTE-RIVAS-I., DÍAZ-MÉNDEZ-A., REYES-GARCÍA-C. A., AND REYES-GALAVIZ-O. F. (2006). *Hybrid Neural Network Design and Implementation on FPGA for Infant Cry Recognition*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 703–709.

SUDHA-N., MOHAN-A. R., AND MEHER-P. K. (2011). A Self-Configurable Systolic Architecture for Face Recognition System Based on Principal Component Neural Network, *IEEE Transactions on Circuits and Systems for Video Technology*, **21**(8), pp. 1071–1084.

TESSIER-R., POCEK-K., AND DEHON-A. (2015). Reconfigurable Computing Architectures, *Proceedings of the IEEE*, **103**(3), pp. 332–354.

THEOCHARIDES-T., LINK-G., VIJAYKRISHNAN-N., INVIN-M. J., AND SRIKANTAM-V. (2004). A Generic Reconfigurable Neural Network Architecture as a Network on Chip, *IEEE International SOC Conference, 2004. Proceedings.*, pp. 191–194.

TRIMBERGER-S. M. (2015). Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology, *Proceedings of the IEEE*, **103**(3), pp. 318–331.

TSU-W., MACY-K., JOSHI-A., HUANG-R., WALKER-N., TUNG-T., ROWHANI-O., GEORGE-V., WAWRZYNEK-J., AND DEHON-A. (1999). HSRA: High-speed, Hierarchical Synchronous Reconfigurable Array, *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99, ACM, New York, NY, USA, pp. 125–134.

VAINBRAND-D., AND GINOSAR-R. (2010). Network-on-Chip Architectures for Neural Networks, *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '10, IEEE Computer Society, Washington, DC, USA, pp. 135–144.

WANG-Y., ZHOU-X., WANG-L., YAN-J., LUK-W., PENG-C., AND TONG-J. (2013). SPREAD: A Streaming-Based Partially Reconfigurable Architecture and Programming Model, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **21**(12), pp. 2179–2192.

WINEGARDEN-S. (2000). Bus Architecture of a System on a Chip with User-Configurable System Logic, *IEEE Journal of Solid-State Circuits*, **35**(3), pp. 425–433.

XILINX. (2004). Two Flows for Partial Reconfiguration: Module Based or Difference Based, XAPP290, V1.2.

XILINX. (2006). Early Access Partial Reconfiguration User Guide, UG208.

XILINX. (2008). FPGA Design FLow Overview.

XILINX. (2012a). LogiCORE IP Fast Fourier Transform v8.0.

XILINX. (2012b). Partial Reconfiguration Tutorial - PlanAhead Design Tool, UG743, V14.1.

XILINX. (2013). Partial Reconfiguration User Guide, UG702, V14.5.

XILINX. (2016). 7-Series Architecture Overview.

XILINX. (2017a). 7-Series FPGAs Configuration User Guide, UG470, V1.12.

XILINX. (2017b). Vivado Design Suite User Guide - Partial Reconfiguration, UG909, V2017.1.

YANG-Y., AND WANG-J. (2001). Pipelined All-to-all Broadcast in All-port Meshes and Tori, *IEEE Transactions on Computers*, **50**(10), pp. 1020–1032.

YOU-J., AND WONG-S. (1993). Serial-Parallel FFT Array Processor, *Signal Processing, IEEE Transactions on*, **41**(3), pp. 1472–1476.

ZHANG-D., AND LI-H. (2008). A Stochastic-Based FPGA Controller for an Induction Motor Drive with Integrated Neural Network Algorithms, *IEEE Transactions on Industrial Electronics*, **55**(2), pp. 551–561.

ZHANG-X., DING-Y., HUANG-Y., AND DONG-X. (2010). Design and Implementation of a Heterogeneous High-performance Computing Framework using Dynamic and Partial Reconfigurable FPGAs, *2010 10th IEEE International Conference on Computer and Information Technology*, pp. 2329–2334.