The rapid increase of network traffic and network processing complexity requires the design of more efficient network devices. Specifically, devices located at edge and access nodes (e.g., edge routers and content switches) require a blend of performance and flexibility in order to meet the network processing requirements. This dissertation describes a reconfigurable network processing platform that can be adapted to diverse network processing requirements and traffic fluctuations. The inherent FPGA's programmability at the software and hardware level is exploited to design high performance adaptive network devices. An access/edge router, that can be adapted to network traffic fluctuations, is mapped and evaluated using the proposed network processing platform. Furthermore, a content switch is implemented based on the proposed platform and compared with a network processor-based implementation. The proposed platform also incorporates a reconfigurable queue scheduler that can be adapted to the number of active queues. Moreover, a configurable transactional memory controller is proposed that can facilitate the multi-processor programming of network applications. The performance evaluation of the system shows that reconfigurable hardware-based network processing platforms are most suitable for edge, access and enterprise network devices, since they can efficiently meet these devices requirements in terms of performance, power, and flexibility.

TU Delft
Delft University of Technology

ISBN 978-90-807957-4-7

Reconfigurable Network Processing Platforms

Christoforos Kachris

Reconfigurable Network Processing Platforms

Christoforos Kachris

Reconfigurable Network Processing Platforms

Reconfigurable Network Processing Platforms

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op dinsdag 11 december 2007 om 10:00 uur

door

Christoforos KACHRIS

Electronic and Computer Engineer
Technical University of Crete
geboren te Athene, Griekenland

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. K. G. W. Goossens

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter	Technische Universiteit Delft
Prof. dr. K. G. W. Goossens, promotor	Technische Universiteit Delft
Prof. dr. ir. P. Dewilde	Technische Universiteit Delft
Prof. dr. N. Dimopoulos	University of Victoria, Canada
Prof. dr. A. Dollas	Technical University of Crete, Greece
Prof. dr. D. Liu	Linköping University, Sweden
Dr. C. J. Glossner	Sandbridge Technologies, USA
Dr. J.S.S.M. Wong	Technische Universiteit Delft
Prof. dr. C.I.M. Beenakker, reservelid	Technische Universiteit Delft

Professor dr. Stamatis Vassiliadis provided substantial guidance and support in this thesis.

Kachris, Christoforos
Reconfigurable Network Processing Platforms
Thesis Technische Universiteit Delft. – With ref. –
Met samenvatting in het Nederlands.

ISBN 978-90-807957-4-7

Subject headings: reconfigurable computing, FPGAs, network processing

Cover design: Popy Panagopoulou

Printed in The Netherlands

in memory of Professor Stamatis Vassiliadis

Reconfigurable Network Processing Platforms

Christoforos Kachris

Abstract

This dissertation presents our investigation on how to efficiently exploit reconfigurable hardware to design flexible, high performance, and power efficient network devices capable to adapt to varying processing requirements of network applications and traffic. The proposed reconfigurable network processing platform targets mainly access, edge, and enterprise devices. These devices have to sustain less bandwidth compared to those utilized in core networks. However the processing requirements on a per packet basis are much higher in these devices (e.g., payload processing). Furthermore, devices in these networks have to be flexible in order to support emerging network applications. A promising technology for the implementation of these devices is the Field-Programmable Gate Arrays (FPGAs). FPGAs are typical devices that combine flexibility (through the reconfiguration) and performance (through the inherent hardware nature that can exploit parallelism), therefore they can efficiently address the requirements of the edge and access network devices.

A reconfigurable network processing platform is presented that includes reconfigurable hardware accelerators, a reconfigurable queue scheduler, and a configurable transactional memory controller. Furthermore, the performance and the constraints of the platform are formulated as an integer optimization problem and an integrated design flow is presented for the platform. Both static and dynamic reconfiguration are explored in this dissertation. Static reconfiguration is utilized to address the different processing requirements of network applications, while dynamic reconfiguration is utilized to adapt to network traffic fluctuations.

Two representative devices were implemented and evaluated in the proposed platform; a multi-service edge router and a content-based (web) switch. In the former device, dynamic reconfiguration is utilized to deal with network

traffic fluctuations. The device monitors the traffic and adapts to the network traffic fluctuations taking into account the reconfiguration overhead. In the latter device, a reconfigurable architecture for a content-based switch is utilized and compared to a mainstream network processor in terms of performance and power. The device accommodates several co-processors that can be interchanged to perform specific type of switching (e.g., URL-based or cookie-based switching). Moreover, the exploitation of reconfigurable logic is investigated for queue scheduling in network devices. A reconfigurable queue scheduler is presented that adapts to the network traffic requirements (number of active queues) and can be used both in edge routers and web switches. Finally, configurable transactional memories are proposed which can be used to efficiently deploy multi-processing platforms for network processing applications. The proposed configurable transactional memory controller can be configured based on the application and device features (e.g., number of processors), can offer an easier programming framework for multi-processor reconfigurable platforms, and provides increased performance compared to traditional locking schemes. The results of the research presented in this dissertation show that the FPGAs can be an efficient alternative to network processors and can be used not only for lower network layers, but also as a complete platform for emerging network processing applications.

Acknowledgements

First of all, I would like to express my deepest gratitude to Professor Stamatis Vassiliadis. Besides being a great mind and a great scientist he was also a really respectable, honorable and admirable person. He was a great mentor that inspired me to be a better person both in life and in science, and he provided substantial guidance and support in this thesis. I feel very lucky that I had the honor to be one of his students.

I am also grateful to Stephan Wong, who helped me a lot in structuring, organizing and writing the dissertation and gave me valuable feedback. I also owe him the translation of the abstract and the propositions in Dutch. I also want to thank Professor Kees Goossens for serving as a promoter, and the examination committee for the useful feedback and comments on the dissertation. Furthermore, I would like to thank all the faculty members, staff and students of the Computer Engineering Laboratory that created such a pleasant and cheerful working environment.

I would also like to thank Chidamber Koulkarni and Gordon Brebner from Xilinx in San Jose for providing substantial guidance during my internship in a great research environment and a sunny place at the same time. The experience of working with great people at the research labs of Xilinx, in the heart of Silicon Valley, was unique.

Although this dissertation is the result of my research in TU Delft, I am grateful to all those who helped me acquire the essential knowledge to pursue a PhD in network processing and reconfigurable computing. First, I would like to thank Professor Apostolos Dollas, from the Technical University of Crete, Greece, who introduced me to the world of reconfigurable computing and computer architecture. I also would like to thank my former colleagues in Ellemedia Technologies, Athens, for the fruitful cooperation in several projects that helped me acquire significant experience in the area of network processing.

I would also like to acknowledge Sandbridge Technologies - John Glossner in

particular - and the TU Delft Computer Engineering laboratory for financially supporting my research.

Finally, I would like to express my deepest gratitude to my parents, Dimitris and Maria. After all, the long journey till this PhD started when my father brought me my first electronic kit and an 8086-based computer that triggered my curiosity to learn how all these “toys” work. I also would like to thank all of my friends, in Greece, in the USA, and in the Netherlands; spending time with good friends is most valuable for facing the stressful work during the PhD. Many thanks also to Popy Panagopoulou for designing such a beautiful cover for this thesis. Lastly, I would like to apologize to my niece, Vasiliki, and my nephew, Dimitris, for being so far away during their very first years in this world. I hope to make it up to them in the future.

Christoforos Kachris Delft, The Netherlands, 2007

Contents

List of Tables

List of Figures

List of Acronyms

ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
ASSP	Application Specific Standard Product
BRAM	Block RAM (embedded RAM)
CAM	Content Addressable Memory
CMU	Configuration Management Unit
CRC	Cyclic Redundancy Check
CTMC	Configurable Transactional Memory Controller
DES	Data Encryption Standard
DMA	Direct Memory Access
DSL	Digital Subscriber Loop
DWRR	Deficit Weighted Round Robin
FIFO	First-in First-out
FPGA	Field-Programmable Gate Array
FSL	Fast Simplex Link
FSM	Finite State Machine
GMAC	Gigabit Media Access Controller
GPP	General Purpose Processor
GPS	Generalized Processor Sharing
HTTP	Hypertext Transfer Protocol
IMS	IP Multimedia Systems
IP	Internet Protocol
ISP	Internet Service Provider
LZC	Lempel-Ziv Compression
NoC	Network-on-Chip
OPB	On-chip Peripheral Bus
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
QoS	Quality of Service
SoC	System-on-a-Chip
SOHO	Small Office, Home Office
SSL	Secured Socket Layer
TTM	Time-to-Market
URL	Uniform Resource Locator
UTM	Unified Thread Management
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

VLIW	Very Long Instruction Word
VPN	Virtual Private Networks
WFI	Worst-case Fair Index
WFQ	Weighted Fair Queuing
WF2Q	Worst-case Fair Weighted Fair Queuing

Chapter 1

Introduction

With the advent of new network applications in recent years, network traffic has increased exponentially. In the early days of Internet, the traffic mainly comprised of text emails and simple web traffic. Currently, new applications such as on-line video sharing, tele-conferencing, multi-player gaming, and shared databases have significantly contributed to the increase of network traffic. In addition to the network traffic increase, the number of operations performed on each network packet has also increased. In particular, the processing requirements per packet have increased significantly compared to the first network devices. For example, many network devices have to support new functions to fulfill security requirements, such as firewall, anti-spam, anti-virus, intrusion detection, and intrusion prevention applications.

The exponential growth of network traffic and the increase of application complexity have sparked the need for more powerful devices in order to deal with the heavier processing requirements. Furthermore, the network traffic is inherently dynamic, therefore efficient implementation platforms are required that can adapt to network traffic fluctuations.

One possible implementation platform for network processing requirements is the Field-Programmable Gate Arrays (FPGAs). FPGAs are semiconductor devices containing hardware programmable logic units that can be reconfigured based on the application requirements. The current FPGAs have evolved to powerful System-on-a-Chips (SoCs) able to accommodate both reconfigurable logic and specialized cores such as DSP units, embedded memory, processors and network interfaces. Therefore, in the literature FPGAs are also referred to as reconfigurable platforms.

The main goal of this thesis is to investigate how to exploit the flexibility of reconfigurable platforms to create more efficient network processing platforms that can adapt themselves to the network traffic fluctuations and can address the current and future network processing challenges. Furthermore, the performance and the power consumption of the FPGAs are evaluated and compared with other devices (such as network processors).

This chapter presents a short introduction to the area of network processing and the area of reconfigurable logic. Section 1.1 presents a short introduction to network processing in terms of applications and platforms. Section 1.3 presents the current and the future challenges in the area of network processing. The challenges related to the network traffic are presented followed by an explanation of the challenges of implementation of network devices. Section 1.4 presents a short introduction to reconfigurable logic and its application in the area of network processing. Section 1.5 outlines the research questions we address in this thesis and the goals of this research. Finally, Section 1.6 presents an overview of the dissertation.

1.1 Introduction to Network Processing

Network processing refers to the tasks performed on packets by network devices. Although network devices range from simple switches to advanced unified thread management (UTM) devices, most of them perform some common functions. This section presents a short introduction into several general network processing functions that are performed by network devices.

Header processing encompasses all the operations needed to interpret the packet header information and, additionally, the ensuing operations that need to be taken based on the header information.

Each network packet has one or more headers that contain information about the packet. The information range from source and destination addresses to many control fields (e.g., checksum, fragmentation indexes, expiration timers, priority information, etc.). The number of headers that a packet contains depends on the network protocol. The majority of the Internet networks protocols use the Open Systems Interconnection (OSI) model [6] that consists of seven layers. Each layer defines its own header structure although some layers are optional (e.g., many protocols do not use the Presentation and the Session layer). The most widely utilized headers are from the data-link, the network, the transport, and the application Layer. The most common processing tasks are the classification of the packets based on the headers and the header mod-

Table 1.1: Number of instructions per program, Source: C.Patrick *et al* [1]

Program	Type	# Instructions
IP Route	Header	~ 200
MD5	Payload	~ 2000
3DES	Payload	~ 40000

ifications (e.g., update counters, checksum recalculation, etc.). In many cases, the header processing is performed at bit level. General-purpose processors are inefficient in header processing since the instructions process the data at the byte level. Therefore, multiple cycles are required to process the data at the bit level (e.g., shift-ing and logically and-ing instructions). A typical header processing task is the calculation of the header checksum. Every header includes a field that contains a checksum to certify that the header data are correct. This field is calculated using the header data during the transmission of a packet. At reception of a packet, the header checksum is recalculated and compared with the received checksum field.

Payload processing encompasses all the operations that are performed on the packet payload. The payload is the segment of the packet that carries the actual data information. In many applications, packets need not only to be routed, but they also need to be modified. For example, when packets are sent over secure connections they need to be encrypted before forwarding. Another example can be found in many wireless applications, wherein the packets need to be compressed to reduce transmission time and power consumption. This processing is usually applied only to the payload of the packet while the header remains the same. Typical payload processing tasks are the compression/decompression, encryption/decryption, intrusion detection and prevention, content-based switching and transcoding (transforming of a media stream in terms of frame rate, resolution or format for example). Payload processing can be up to two orders of magnitude more complex than header processing. Therefore, usually this processing is performed by special hardware acceleration units. Figure 1.1 highlights the typical number of instructions that are executed in header and payload functions for the Alpha 21x64 ISA [1]. IP Route is used to route packets based on the IP header, while MD5 and 3DES are payload encryption algorithms.

Traffic Management refers to all operations that involve more than one single packet (as was the case for the previous 2 types of processing). For example,

decisions are based on the information of multiple packets and operations are applied to many packets simultaneously or over a period of time. The traffic management can be applied either to aggregated traffic (class of flows) or to individual flows (class of packets).Typical traffic management tasks are:

Traffic metering Network traffic is measured for statistics and other information. Traffic metering is usually applied to packet analyzers that are utilized to detect problems in the network and measure traffic load. In many network applications several counters need to be updated for each packet.

Traffic policing Traffic policing is used to discard traffic exceed a specific threshold. This functions ensures that a given link or customer does not transmit more packets than authorized.

Traffic shaping Traffic shaping is used to delay some packets and allows others to pass through quickly. Traffic shaping is used to prevent congestion in the links. The packets are scheduled in a way that conforms to specific statistical bounds.

Queue Management The packets are usually stored to and retrieved from queues before processing. Therefore, efficient queue management is crucial to the performance of a network processor.

Traffic Scheduling Emerging network applications such as video conference or on-line video require better guarantees than best-effort to provide Quality of Service (QoS). Traffic scheduling is used to schedule the forwarding of the packets based on the application requirements in terms of latency, jitter, and bandwidth. There are several schemes for the scheduling ranging from simple priority based schemes to highly sophisticated schemes such as weighted fair scheduling.

Figure 1.1 depicts the traffic management functions that are performed in a typical router [3]. First, in the input ports, the traffic policing determines if the packets will be accepted based on the service level agreements (SLA) for the specific flow. The queue management is used to store the packet in queue or drop it in case of congestion. The traffic scheduler is used to determine which packet will be forwarded to the switch fabric based on the priority of the packets. In the output port, first the queue management and the packet scheduling is performed and then the traffic shaping is used to determine which packet will be forwarded.

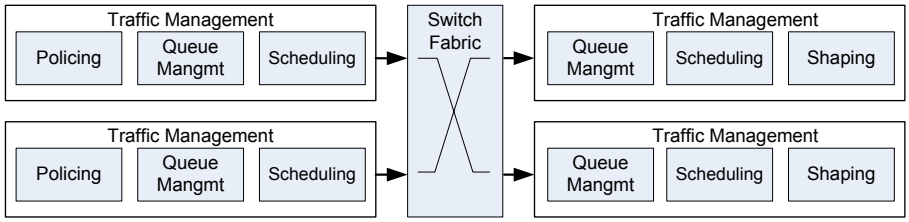

Figure 1.1: Traffic management in routers

Finally, network processing tasks can also be classified into two categories based on the functionality: the **data plane** and the **control plane**. Data plane processing refers to the processing that should be performed in the packets from the input port to the output port. Data plane processing must be performed at wire-speed in order to avoid dropping packets. Packets that belong to data plane are usually independent therefore they can be processed in parallel. On the other hand, control plane refers to the packets that are used from the network processing platform to control the data plane. The control plane packet can be used for routing table updates, encryption key exchange, etc. These packets usually do not have to be processed at wire-speed and exhibit little data parallelism [7].

1.2 Network Processing Platforms

Network processing can be implemented using different design techniques, technologies, and/or processors to constitute a network processing platform.

Definition: *A **platform** is a set of hardware and software modules that can be used as a building framework for the development and the execution of specific set of applications.*

The decision on how to design such a platform is influenced by requirements such as performance, power dissipation, and cost amongst others. The following section presents the most common components and technologies used in the implementation of network processing platforms:

General-Purpose Processors (GPP) are processors that have been designed to cover a broad range of applications. The early day network devices were mainly implemented using GPP. The processing requirements of these devices were quite simple; therefore the processors could efficiently handle the workload of the early-day networks. The major advantage of using GPPs lies in

the flexibility that is provided by simply reprogramming the processors to accommodate new protocols or applications. Furthermore, GPP are often chosen as implementation devices because they can be programmed using various, widely utilized programming languages. Currently, GPPs are used mainly for the control-plane processing, such as Routing and Management protocols, and not for the data-plane processing. The main reason is that control-plane processing is more complex than data-plane and has long code path, but usually do not have real-time constraints (e.g., management protocol processing does not have to be processed at wire speed).

Application-Specific Integrated Circuits (ASIC) are circuits dedicated to perform specific tasks or operations. Consequently, the circuits can be fully optimized for speed or power consumption and they are the preferred design choice for standardized network applications (e.g., Ethernet switch). Therefore, ASICs offer high performance but low flexibility. Currently, due to the high Non-Recurring Engineering (NRE) cost of an ASIC design, this platform is preferred only when the devices target high volume (usually over 100K) quantities and/or the major system constraints are the cost and the power dissipation.

Application-Specific Standard Product (ASSP) are semi-customized integrated circuits that target specific application markets and usually comply with specific standards but can be modified to meet specific customer's needs. The market of ASSPs is wider than that of the ASICs since they target many customers with common requirements. Common ASSPs are packet framers, encryption accelerators, compression accelerators, traffic managers, etc.

Network Processors (NP) are Systems-On-a-Chip (SoC) platforms that accommodate a number of Application-Specific Instruction Processors (ASIPs) optimized for network processing applications. In addition, NPs incorporate network and memory interfaces and some specialized co-processors (e.g., encryption accelerators). The topologies of the processors range from pipelined schemes, in which each pipeline stage is tuned for specific functions, to pool schemes, in which all of the processors run the same program (RTC: Run-to-Completion) or hybrid schemes (in which the system consists of a number of pipeline stages and each stage has a pool of processors). Moreover, the architectures of the processors vary a lot. The most common architectures are RISC (Reduced Instruction Set Computers) with specialized instructions for network processing (e.g., barrel shifters) but there are also many network processors that accommodate VLIW (Very-Long-Instruction-Word) processors or dataflow architectures [7].

Field-Programmable Gate Arrays (FPGA) are devices that can be reprogrammed (at the hardware level) to perform different tasks. The main advantage of FPGAs is that they can provide flexibility to support more than one type of tasks and at the same time the hardware parallelism can be exploited to accelerate the processing. The first FPGAs were mainly used as glue logic for several applications such as communications and signal processing. Their most common use in network processing applications is as framers or low level protocol processing. Furthermore, they are used widely for rapid system prototyping since they offer a shorter development time compared to ASICs. But current-day FPGAs have become powerful platforms that can accommodate not only reconfigurable logic but also hard- and soft-core processors, embedded memory, network and memory interfaces, and specialized modules such as small DSP units. Therefore, in literature FPGAs are referred to as reconfigurable platforms. Furthermore, some FPGA devices can also be partially reconfigured during execution time, i.e. part of the FPGA can be functional while the remaining part can be reconfigured. This feature can be exploited to design adaptive systems that can be adjusted to the workload's requirement (e.g., the network traffic fluctuations). Therefore, FPGAs can be a viable alternative for the implementation of advanced network processor device, mainly because of the combination of performance and flexibility that they provide.

In 2006, almost 50% of the network processing platforms were implemented as an ASIP or ASSP, around 35% were implemented in ASICs, and 15% were implemented in FPGAs [8]. The main application of the FPGAs were the communication processors (42%) and the traffic management (35%).

Figure 1.2 depicts the mapping of the mentioned devices in terms of flexibility and performance. The ASICs and the GPPs represent the outer edges of performance and flexibility, respectively. The NPs are closer to the GPPs since they provide the flexibility of programmable processors. The ASSPs are closer to the ASICs, since they provide high performance but also can be configured (usually through a limited number of configuration registers). The FPGAs on the other hand, provide both high flexibility and high throughput. Although the flexibility is not as high as the GPPs, due to the higher design effort entailed for designing in hardware, new EDA tools [9] that can transform software code (e.g., C) to synthesizable RTL code (e.g., VHDL) can further increase the flexibility of FPGAs.

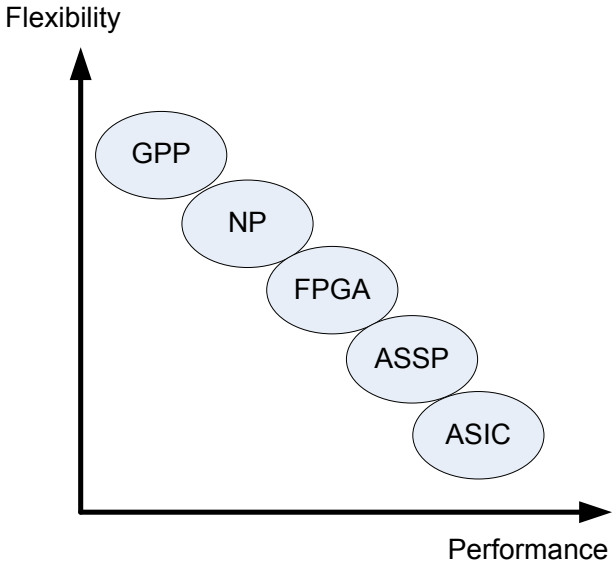

Figure 1.2: Performance-Flexibity area

1.3 Network Processing Challenges

This section presents the new challenges that have arisen in the past few years in the area of network processing. The challenges can be categorized to those related to the workload of the network processing (the network traffic) and in those related to the network processing platform. In the domain of network traffic the most important challenges to cope with are:

Increased network traffic: Network traffic keeps increasing mainly because of two reasons; the additional devices that are connected to the internet and the new applications that are utilized (e.g., streaming video)

Increased network traffic complexity: The per-packet processing requirements keep increasing. Some devices have to process the packets up to the application layer to address new requirements such as security and QoS.

Inherent network traffic fluctuation: Network traffic is both quantitatively (in terms of bandwidth) and qualitatively (in terms of applications deployed) dynamic over time.

In the domain of network processing platforms the most important challenges are:

Reduced IC power consumption: The processing requirements of current applications lead to the utilization of more powerful devices. Unfortunately, the new devices have to consume almost the same power of the previous devices due to several technology and economic constraints (packaging, cooling of the devices, etc.). Therefore, a major challenge is the design of network processing platforms with reduced power consumption.

Parallel programming of the platforms: Due to the power consumption constraints, the industry has moved to the deployment of multi-processor platforms. Although multi-processor platforms can provide a better performance per unit of power dissipation, they create several additional issues. The most important is the increased programming effort of parallel processors (such as partitioning and synchronization).

Reduced Time-to-Market (TTM) and Emerging applications: The emerging network applications (such as video streaming, e-commerce, etc.) create new requirements for network devices. Therefore, network processing platforms must be flexible to support the new requirements and to reduce the time-to-market.

1.3.1 Network Traffic Growth

Although the density of transistors per chip almost doubles every 18 months according to Moore's law, the network traffic increases even faster. The network traffic increases almost twice every 12 months [4] [10]. Figure 1.3 depicts the normalized growth of network traffic, line capacity and Moore's law. As depicted, the traffic has the highest rate of increase, followed by the line capacity that increases similar to the Moore's law (every 18 months) [11]. The increase of network traffic is due to several factors. One of the main factors is that the speed of optical links continues to improve at a higher rate than electronics causing the gap between the optical speed and the electronics speed to widen. Furthermore, new Internet applications increase significantly the network traffic. It is estimated, that network traffic will keep increasing even more rapidly mainly because of the explosion of Peer-to-Peer (P2P) file sharing applications and on-line video browsing [11]. These two application classes have injected into the network huge amounts of video files that stretch the capabilities of the network devices. The population scaling is also a significant cause of the network traffic increase. More users and enterprises are connected to the Internet every day. Furthermore, the available bandwidth to the end user has increased significantly with the deployment of cheap and readily accessible broadband connections (e.g., ADSL+, VDSL). The increase of network

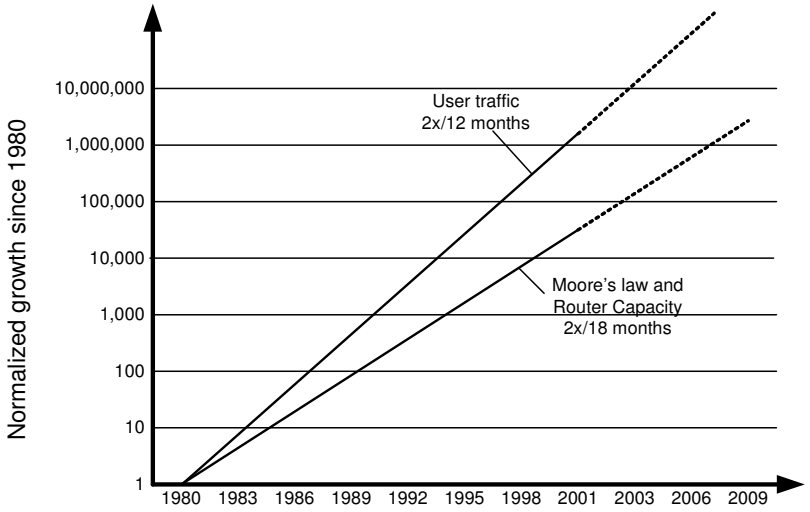

Figure 1.3: Network traffic growth (Source: McKeown, [4])

traffic translates to the need for more powerful network devices that can cope with the increased traffic. The obvious solution of utilizing parallelism can not always solve the problem efficiently. For example, the scaling of routers (basic switching devices) using crossbar switches can be limited due to several other constraints (e.g., head of line blocking, efficient high speed scheduling, etc.) [12].

1.3.2 Network Traffic Complexity

Besides the growth of network traffic, the complexity of network processing is increasing significantly too. While the first network processing applications required header processing in the lower OSI layers, emerging applications require more processing in the higher layers as depicted in Figure 1.4. For example, the first network devices had to process only the network layer header. Nowadays, many devices have to process information up to the application layer in order to provide service and security guarantees.

A representative example is the Unified Threat Management Device. In this case, the device has to support several functions for network security such as content filtering, intrusion detection and intrusion prevention, anti-spam, anti-spyware, and anti-virus processing. All these functions are performed in the application layer and additionally utilize state information (the process-

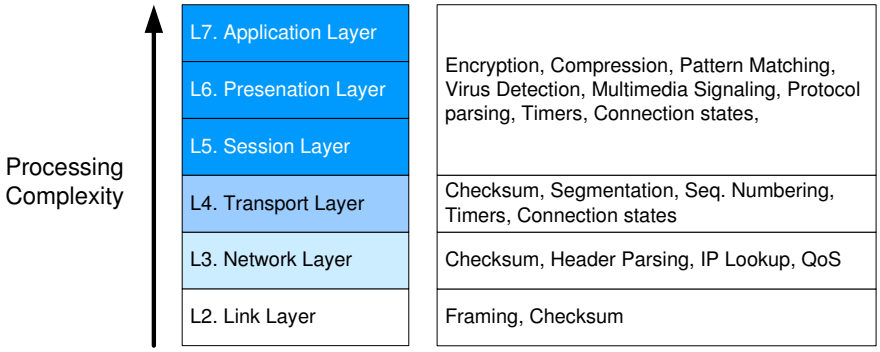

Figure 1.4: Protocol processing complexity

ing of the current packet requires information from the previously processed packets belonging to the same flow). Another representative example is the content-based switch. In large server farms, the workload balancing was usually performed by simple network layer header processing such as the source IP of the packets. However, emerging Internet requirements such as regional-based information created the need for more advanced load balancing devices: the content-based switches. These switches perform the load balancing of the cluster servers using information from the application layer (e.g., http header).

The processing of packets at the application layer requires more powerful network devices that can face the increased processing requirements of new applications. Since generally the network processing can be parallelized, the advanced network processors use multiple processors to face the increased workload. Nevertheless, the processing requirements also include payload processing therefore many functions (such as pattern matching, packet encryption and packet compression) are performed in specialized hardware co-processors inside or outside of a network processor.

1.3.3 Network Traffic Fluctuation

An important observation of network traffic is that it is not constant, but continuously fluctuates. This network traffic fluctuation entails the constant change of the volume and the characteristics of the network traffic. The fluctuation can be in terms of bandwidth, number of flows, average packet size, application, or protocol. Several papers have been presented that show the characteristics of the network traffic (e.g., the LAN traffic [13], the WAN traffic [14], the web

server traffic [15], etc.). In [16], there is a synopsis of some common features of the network traffic. It is shown that while the average utilization may be very low, the peak utilization can be very high.

In literature, the nature of network traffic of several topologies (e.g., LAN, WAN, etc.) were presented. Furthermore, it is shown how the features of network traffic change over time (e.g. the average packet size, the number of packets, the type of protocol etc.). Although network traffic is quite random there are some noticeable general trends. For example, in [17], [18] it is shown how the number of business phone calls and the residential phone calls change over time. The number of business phone calls reaches its maximum during noon and starts to decline afterwards. Similar behavior is noticed for other kind of traffics. For example, the number of VPN (Virtual Private Network) connections is usually high during working hours, while during non-working hours the majority of traffic is from Peer-to-Peer (P2P) networks and on-line video sharing. In [19], a study on P2P network traffic shows the network traffic fluctuation for several applications in a period of a week. In this paper, it is shown that during the working hours the majority of traffic comes from web browsing while during the non-working hours the majority of the traffic comes from peer-to-peer file exchange applications. In [20], an analysis of the network traffic for a local area wireless network was presented. The paper shows how the number of stationary and mobile users changes over time. The maximum number of users is reached at noon both for stationary and mobile users during the weekdays while in the weekends the maximum number of users is reached in the afternoon. An extensive study on internet traffic characterization was presented in [21]. Internet traffic is studied in terms of jitter, bandwidth, burstiness, etc. The study indicated that the proportion of packets per protocol (or per application) change significantly between the morning hours and the afternoon.

Figure 1.5 shows the network traffic fluctuations for several types of traffic during a week [22]. The *Data Transfers* traffic includes applications such as HTTP, FTP, etc. The *File Sharing* traffic includes applications such as eDonkey, Gnuttela, etc. The *Encrypted Traffic* includes applications such as SSH, HTTPS, IPSec, etc. Finally, the *Undefined* traffic includes applications that can not be identified by the TCP port (or have not been identified yet) and usually refers to other file sharing applications (such as torrents-files sharing applications). As depicted in the figure, the traffic fluctuations can be quite large (e.g., *Data Transfer* traffic fluctuates from 10% to 55%).

Conclusively, a major challenge in the domain of network processors is to

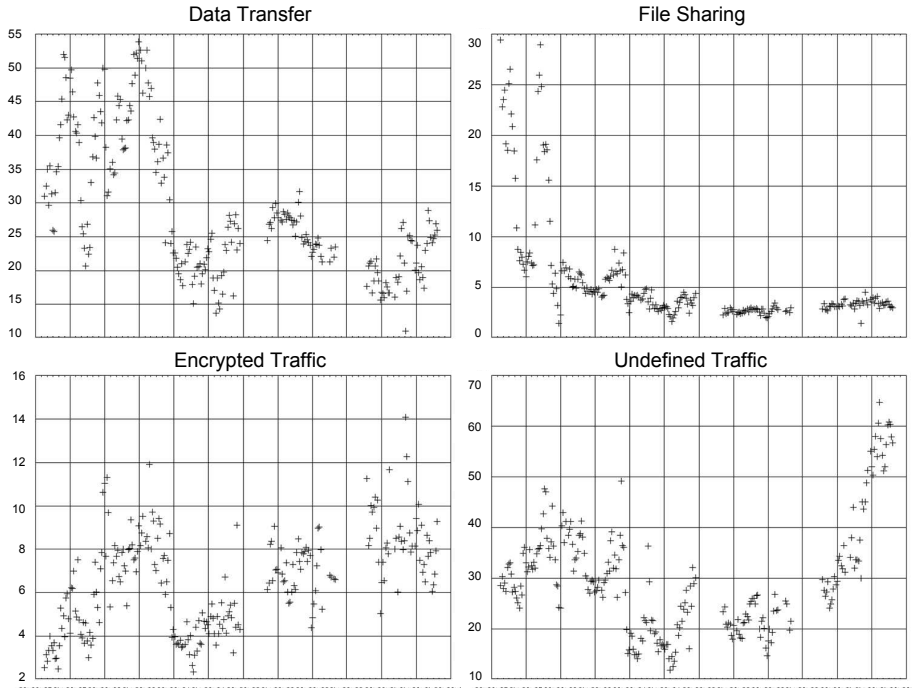

Figure 1.5: Network traffic fluctuations during a week

design a system capable to adapt itself to fluctuations of network traffic. Currently, network processors are designed to sustain a specific maximum bandwidth for the worst case traffic (usually using the smallest packet sizes) of a specific flow. A typical example is the maximum IP forwarding throughput. In this case, the network processors can sustain a specific data rate for simple IP forwarding, but this data rate drops significantly if the packets need further processing (e.g., encryption). The use of hardware accelerators can reduce significantly the processing time, therefore a common challenge in network-processor design is the software-hardware partitioning of the system (e.g., the number of processors/co-processors and the type of the co-processors). On the other hand the use of hardware accelerators occupies major portion of the chip that could be used instead to provide higher processing power when the packets do not require payload processing. Therefore, the reconfigurable platforms (with inherent dynamic reconfigurability) seems ideal for the design of network processing systems that adapt themselves to network traffic fluctuations.

Table 1.2: Power requirements for a PDA [2]

Process technology (nm)	90	65	45	32
Operational voltage (V)	1	0.8	0.6	0.5
Processing Performance (GOPS)	2	14	77	461
Energy Efficiency (MOPS/mW)	20	140	770	4160
Required Average Power (mW)	100	100	100	100
Required Standby Power (mW)	2	2	2	2
Battery Capacity (Wh/kg)	200	200	400	400

1.3.4 Power Consumption

A major challenge in the design of current and future Integrated Circuits (ICs) is power consumption. While the number of transistors continues to increase, the thermal budget of the packages and the battery capacity has remained almost the same during the last few years [2]. For example, the required average power consumption and the standby power consumption for a PDA (Personal Digital Assistant) have to remain the same due to battery life, although the processing performance has to be increased significantly to cover the future processing requirements, according to the International Technology Roadmap for Semiconductors (ITRS) [2] (Table 1.2). One of the most obvious results of this constraint is the shift from increasing the clock frequency and the complexity of the processors to the integration of low frequency simpler multi-processors. Especially in the area of network devices the power dissipation increases exponential. The major server clusters that are used by big companies consume huge amounts of energy not only for the operation of the servers but also for the cooling systems that are used to keep the temperature of the devices under specific thresholds. Furthermore, in the 90nm process technology, the industry crossed a major inflection point. Using older processes the static power of the ICs was always less than the dynamic power consumption as it is shown in Figure 1.6. Using the 65nm process the static power became almost equal to the dynamic power and it is estimated that in the future a major challenge will be the reduction of the static power [5].

Figure 1.7 depicts the static power consumption for several Xilinx Virtex 5 FPGAs, built with 65nm process technology. As it is shown the static power consumption is increasing significantly as we move to larger devices. As a figure of merit, the dynamic power consumption of 20K slices (each slice con-

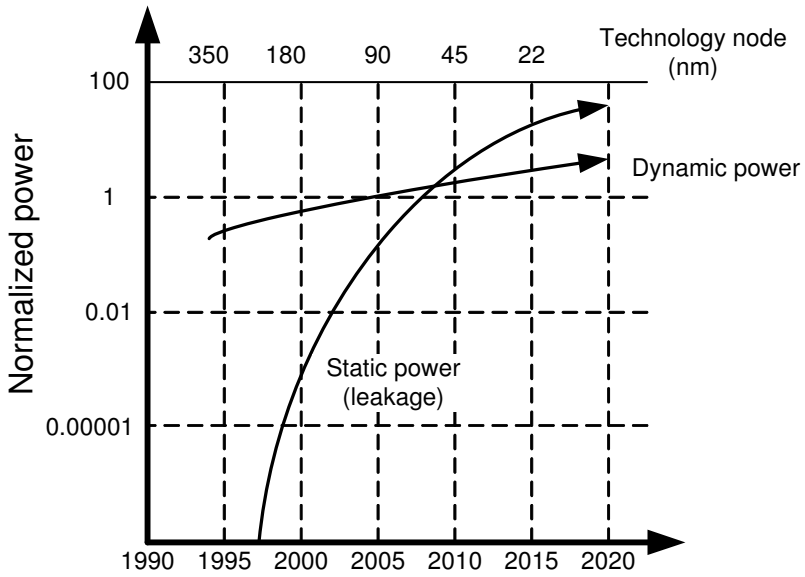

Figure 1.6: Dynamic vs. Static power consumption, [Source: Xilinx Inc., [5]

tains 4 LookUp Tables (LUT) and 4 Flip-Flops (FF)), clocked at 100MHz and with 12.5% toggle ratio is shown in the last column (around 660mW). This figure depicts that the use of smaller devices can have a significant impact not only on the cost of the device but also in the power consumption. The efficient exploitation of dynamic reconfiguration can help implement a system using a smaller device (by loading the required functions at the proper time) in some applications. Ideally, a system could have the same performance as a larger device (in terms of number of gates) if the application uses specific modules in each time frame.

1.3.5 Parallel Programming

The power constraints of the integrated circuits created the need for more power efficient architectures. Many vendors shifted to the development of multi-processor platforms to address this problem. The use of simpler processors clocked at low frequencies instead of a single highly sophisticated processor clocked at high frequency can achieve better performance dissipating less power. The improvement of performance is considerably significant in applications that can be parallelized. Network processing applications are inherently parallel, since each packet can be processed independently of the other

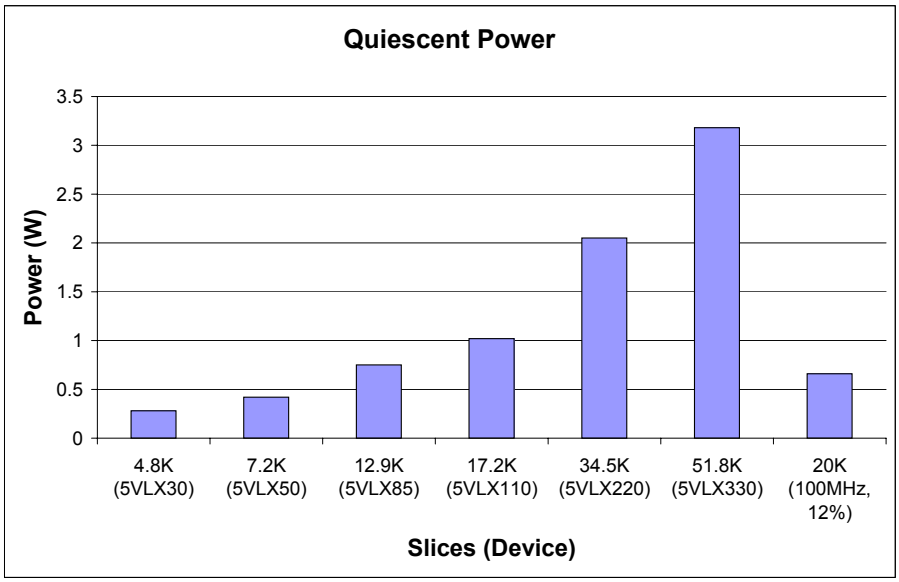

Figure 1.7: Static Power of Xilinx FPGAs

packets. This is the reason that the majority of network processor consists of multiple simple RISC processors. The main drawback of multi-processing is the increased programming effort that is introduced. The application must be divided into specific tasks for each processor and synchronization must be performed between the processors. The most common synchronization scheme is the locking scheme but using this scheme may cause several problems (e.g., deadlocks). Therefore, more efficient synchronization schemes are required that can improve the programming efficiency of multi-processor platforms.

One of the promising alternative to lock schemes is the use of transactional memories. Transactional memories can improve significantly the programming efficiency of parallel programming by using mechanisms similar to database transactions. Each processor/thread accesses the shared memory without locking in advance the memory region. If there are no conflicts, the processor completes the modifications. Otherwise, the processors/threads is aborted and and the transaction is re-executed. Transactional memories can be implemented either in software or in hardware. Usually, the hardware schemes are implemented by modifying multi-processors cache-coherence protocols. The use of caches in network processing platforms is usually avoided due to the low temporal locality of network processing applications. Therefore, new schemes must be developed that support transactional memories for network

processing platforms without the use of cache coherent schemes and without adding significant area overhead.

1.3.6 Reduced Time-to-Market

Emerging Internet applications (e.g., video streaming, e-commerce, etc.) create new processing requirements for the network devices. Therefore, these devices have to be flexible to support future requirements. The typical time frame of 18-months for the development of an ASIC can not be sustained in many network markets. Network processors and FPGAs offer the flexibility to upgrade the device on the field. Furthermore, the cost of the development of a new design has been increased significantly the last years, especially the cost of the lithography masks of the state-of-the-art process technologies like 65nm. The high development cost and the increased development time were the most important reasons that many vendors of network devices preferred the utilization of network processors and FPGAs, instead of the design of an IC for their specific application.

1.4 Reconfigurable Logic Platforms

Section 1.2 briefly described reconfigurable gate-arrays (FPGAs) as an alternative device in the implementation of network devices. This section gives a more detailed description of FPGAs and their deployment in network processing devices.

As described in Section 1.2, FPGAs are semiconductor devices that contain programmable units. The early-days FPGAs contained only programmable logic and programmable interconnect that were used to implement basic logic modules (FSMs, decoders, etc.). Therefore, the main use of the FPGAs was as glue logic or as prototyping of hardware designs. Current FPGAs contain not only programmable logic but also embedded memory units, processors, DSP units, transceivers, etc. Therefore, FPGAs have become complete System-On-a-Chip (SoC) platforms that can be used in diverse embedded applications. The main feature of the FPGAs is that they combine the flexibility of the software development with the performance of hardware modules. The flexibility is preserved through the programmability of the embedded processors and the reconfigurability of the basic units. The reconfigurability can be applied both at design time (static reconfiguration) and at run time (dynamic reconfiguration). Dynamic reconfiguration usually refers to an FPGA in which a part of the de-

sign is operating while another part is being reconfigured (partially dynamic reconfiguration). The reconfiguration time of older FPGAs could be hundreds of milliseconds. Currently, the reconfiguration time has been reduced significantly. As a figure of merit, the time to reconfigure one quarter of an FPGA containing 40,000 logic cells is around 1 ms [23]. During reconfiguration, a specific area can not be used hence there can be a decrease in the performance of the system. Therefore, even with reduced reconfiguration time, the main challenge remains to hide the reconfiguration latency.

FPGA are a promising solution and their features can be exploited to efficiently address the network processing requirements.

- Network traffic growth: The high computational power of the FPGAs due to the inherent parallelism can be used to address the increased network traffic.
- Network traffic complexity: The computational power of FPGAs can be used for the implementation of hardware accelerators capable of meeting the high processing requirements introduced by the increased network traffic complexity such as encryption, compression, traffic management, etc.
- Network traffic fluctuation: The dynamic reconfiguration of FPGAs can be exploited to efficiently address network traffic fluctuations. The organization of the FPGA can be reconfigured to meet the changing network processing requirements of the network traffic.
- Power consumption: The implementation of several network processing tasks into reconfigurable logic can provide a significant reduction of the power dissipation compared to the implementation in processors. Generally, the hardware implementation of applications is more efficient in terms of power consumption than a software implementation due to the lower switching activity of hardware modules compared to a processor.
- Parallel programming: Although there are several schemes that can be used to program efficiently multi-processor platforms, the majority of them address high complexity platforms and are implemented using several cache coherence schemes. In a network processing platform, usually simple processors are used that need a basic synchronization. FPGAs can be used to implement efficient synchronization schemes without much overhead that can be adapted to the application requirements.

- Reduced TTM: The reconfigurable nature of FPGAs makes them ideal platforms for rapid prototyping. Although the TTM of network processors can be less than that of the FPGAs, the latter provide a fair combination of reduced TTM and value-added characteristics. FPGA EDA tools can provide a rapid development framework and at the same time they provide the flexibility (offering programmability at the hardware and software layers) to support emerging applications and application specific requirements. Furthermore, the use of off-the-shelf Intellectual Property (IPs) units (delivered in hardware description languages) can further reduce the TTM and provide customizations for specific target applications.

1.5 Research Questions

The main goal of the research described in this thesis is to investigate how to exploit the programmability and the reconfigurability of reconfigurable platforms (FPGAs) to design efficient network processing platforms. The thesis explores both the static reconfiguration performed at design time and the dynamic reconfiguration that is performed at run-time. The main questions that this thesis is addressing are the following:

- *How to exploit reconfigurability of reconfigurable platforms to develop efficient application-specific network processing platforms at design time?*

  The main challenge is to determine how to exploit the flexibility of FPGAs to design network processing platforms that can meet the network application requirements. Each network application has different requirements in terms of header processing, payload processing, and traffic management. The goal is to exploit the flexibility of the FPGAs to develop the best suited architecture for a specific applications at design time.

- *How to exploit the dynamic reconfiguration of reconfigurable platforms to create network processing platforms that can adapt to the fluctuating network traffic processing requirements?*

  The main challenge is to create architectures that will best meet the network traffic requirements. In other words, the main question is how to match the adaptive nature of the FPGAs with the dynamic nature of

network traffic. Due to the network traffic fluctuations (in terms of bandwidth, applications, etc.) the processing requirements are constantly changing in a network device. Using the optimum architecture for each network traffic pattern, the maximum performance can be achieved with lower power consumption. Although the dynamic reconfiguration seems a very promising solution, a major topic of this thesis is to investigate under which circumstances the dynamic reconfiguration can be exploited efficiently in network processing applications. The challenge is to identify the parts of a network processing platform that can benefit the most by the dynamic reconfiguration.

Addressing these questions is an important step towards the design of network processing platforms in FPGAs that can face the current and future challenges of network processing. By exploiting the reconfigurability and programmability of the FPGAs, more efficient network processing platforms can be developed in terms of performance, power consumption, and programming efficiency.

1.6 Dissertation Overview

In this thesis an integrated approach to the use of reconfigurable platforms for network processing applications is presented. **Chapter 2** presents a background in the area of network processing. The network processing platforms are categorized based on the network topologies that they are targeting. Furthermore, this chapter presents some common network processing applications such as routing, switching and queue scheduling. Finally, the chapter presents the related work in the area of network processing and reconfigurable logic.

Chapter 3 presents the organization of the proposed reconfigurable network processing platform. Furthermore, the chapter presents an integrated design flow for the development of reconfigurable network processing platforms. The network processing requirements are formulated as an integer non-linear optimization problem and a design space exploration is performed to study the potential benefits of using reconfigurable platforms.

Chapter 4 introduces a new scheme for the efficient and robust development of multi-processing system in reconfigurable platforms in network processing applications. The chapter presents the configurable transaction memory controller that uses the notion of transactions to ease the development of multi-processing platforms. The proposed scheme is ideal for network processing

application in which the number of shared variables is usually small and the conflict probability is low. The transaction memory controller is configurable and can be tuned based on the design time information such as number of processors, number of shared variables, etc.

Chapter 5 describes the implementation of the reconfigurable network processing platform. The chapter presents the implementation of two common network applications (edge routers and web switches) in the proposed platform. Each hardware acceleration modules that has been developed is described and the implementation results are presented. Furthermore, it describes the proposed reconfigurable queue scheduler that can be used both is routers and switches. Two schedulers with different features are implemented in reconfigurable logic. The first one is more scalable and can support a large number of queues but it is less accurate in terms of bandwidth allocation and latency. The second scheduler is more accurate but it can support only a limited number of queues due to area constraints. The scheduler is reconfigured using one of the two schedulers based on the number of the active queues.

Chapter 6 presents the experimental results of the proposed architectures. A dynamically reconfigurable edge server is evaluated in terms of performance for several network traffic workloads. The speedup of the system in which both the processors and the hardware co-processors are reconfigured against a static system is evaluated. Furthermore, the web switch is evaluated in terms of performance and power dissipation. It is compared against a network processor-based implementation of a web switch in terms of latency and throughput. The chapter also presents the performance evaluation of a reconfigurable queue scheduler in terms of bandwidth allocation. The scheduler is compared against an ideal scheduler for several network traffics (traffic fluctuation in terms of number of active queues). Finally, the configurable transactional memory controller is compared against a traditional locking scheme in terms of area and performance. The system is evaluated using both micro-benchmarks and typical network processing applications.

Finally, **Chapter 7** presents the conclusions of this research. The chapter summarizes the dissertation and the main contributions of this dissertation and proposes future research directions.

Chapter 2

Network Processing Background

In the design of network processing devices or platforms, three separate domains must be studied (similar to the design of embedded systems). First, the requirements of the operational environment, in which the devices are going to be deployed, must be investigated. Second, the applications that are going to run on these devices must be identified and studied to identify possible implementations of and optimizations within the network processing devices. Third, design trade-offs must be investigated between the identified implementation and optimization possibilities. This chapter presents a background in the area of the network architectures, the network applications and the network processing platform architectures.

Section 2.1 presents a short introduction in the area of network processing. Section 2.2 presents a short introduction to the network hierarchy and the different processing requirements based on the position of a device in the network. Understanding the targeted network topology of the network device is extremely important to the design of the device. Section 2.3 presents some typical network processing applications that are further investigated in the following chapters. Section 2.4 presents a short introduction in the architectures that are currently used by many network processor vendors. Section 2.5 presents the related work in the area of network processing and reconfigurable logic. Finally, Section 2.6 presents the summary and the conclusion of this chapter.

2.1 Introduction

Network processing is an interdisciplinary field that combines the knowledge from many areas. To design efficient network processing systems a broad knowledge of computer architecture, hardware design, embedded systems, and network topologies is required. First of all, a basic knowledge of network topologies is required to better understand the workload that a network processing platform has to face. The processing requirements of network devices that are used by a customer are completely different from the processing requirements of the devices that are used by an Internet Service Provider (ISP). Furthermore, to better understand the workload of a network device, a basic knowledge of the network applications requirements is required. Applications like simple email programs have completely different processing requirements from a streaming video server application or from a peer-to-peer file sharing application. Besides the understanding of the network traffic and the workload processing requirements, an understating of computer architecture, hardware design and embedded systems is also required to design efficient devices that can face the network processing requirements. Therefore, the main challenge of designing a network processing device is to be able to combine the knowledge of many areas and at the same time to handle the trade-offs in each area.

2.2 Network Hierarchy

Nowadays, the Internet infrastructure comprises a multitude of interconnected networks that can largely be classified based on their scale:

- Local Area Networks (LANs) connect computers and other networked devices together over a relatively short distance such as offices and buildings.
- Metropolitan Area Networks (MANs) cover a much wider area, e.g., a city, by connecting many LANs together to form a larger network.
- Wide Area Networks (WANs) constitute an even larger network by interconnecting many LANs and MANs

Figure 2.1 illustrates the conceptual hierarchy of networks [3] [24]. Enterprise networks are usually the networks that are used in enterprises, university campuses, etc. Access networks are the networks that are used to connect multiple

customer networks and the WAN. Finally, WAN networks are the backbone networks that are used to connect Internet Service Providers and legacy voice-based traffic carriers (e.g., Internet). Each device that is used in these networks has specific requirements. The most common network device is the router. Routers are used to route packets in the network nodes. Although this feature is common to every router, each router has different requirements depending on the targeted network. Routers that operate in the heart of the networks are called core routers, and their processing requirements, protocol support, and port speeds are quite different from the edge routers. The routers that are situated at the periphery of the networks are called edge routers. Edge routers have to aggregate diverse metro and customer ports and they have to support several technologies such as ATM, frame relay, Gigabit Ethernet, and more. The routers that are used to aggregate traffic from several home networks, SOHO networks (Small Office, Home Office), and enterprise networks and to connect them with the Internet Service Providers (ISP) are called access routers. Access routers have to support several services for the customers such as traffic scheduling, traffic policing and billing.

Figure 2.2 illustrates the requirements for the several network routers. Core router have to sustain high bandwidth but do not need to offer feature-rich applications. On the other hand, the processing requirements per packet in access routers are much higher (traffic management or payload processing such as encryption, compression, intrusion detection, etc.) but they have to sustain limited bandwidth. The requirements for the edge routers lies between the core and the edge routers. Furthermore, edge and access routers have to be flexible to support emerging network applications (new applications, new service requirements, etc.), while the core routers perform standard functions. The core routers also have much larger routing tables than the edge and access routers.

Furthermore, in the enterprise networks, specialized network devices are used, besides routers. For example, ISPs and web hosting companies use load balancers. These devices are used to balance the traffic to several server clusters and must be able to switch traffic at layer 4 (transport layer) all the way up to layer 7 (application layer). Enterprises and campus use also firewall devices to prevent network attacks. Firewalls can be used to allow specific network traffic to flow between a private network and the access network.

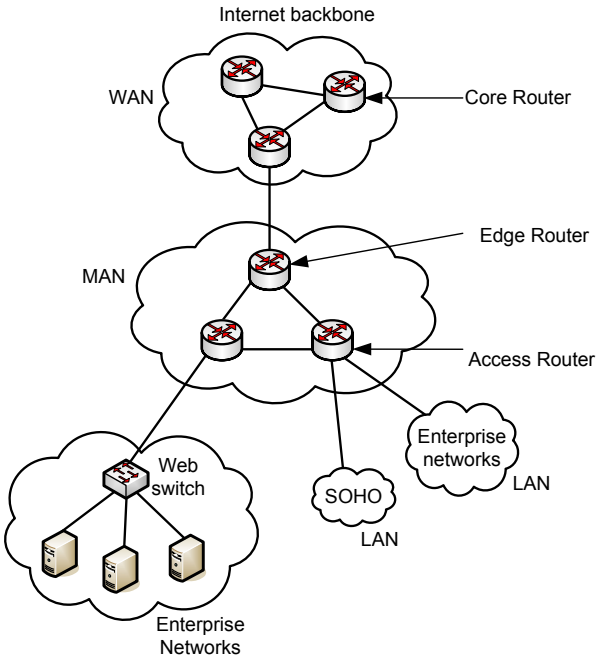

Figure 2.1: Network topologies

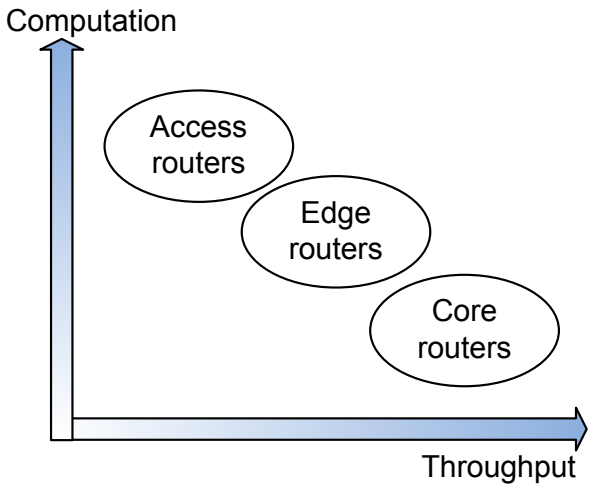

Figure 2.2: Requirements of network topologies

2.3 Network Processing Applications

The main application that is performed in every network node is the switching of the packets from the input ports to the output ports. The switching of the packets is based on one or more OSI layer header information. When the switching is performed using Layer 2 (e.g., Ethernet address) then the devices are called switches. If the switching is performed based on Layer 3 (IP header) then the devices are called routers. When the switching is performed using higher layers (e.g., Transport layer or Application layer) then the devices are called content-based switches. Furthermore, the majority of the routers and the content-based switches have to support traffic management functions such as queue scheduling. This section presents in more detail these applications and theirs requirements.

2.3.1 Edge/Access Routers

Edge routers are network devices that are used to route packets between one or more local area networks (LANs) and a wide area network (WAN). The main difference between a core router and an edge router is the complexity of the packet processing. The core routers have to sustain higher bandwidth than the edge routers but usually the processing is simple and it is performed only on Layer 3 (Internet layer) of the OSI layers. On the other hand, the edge and access routers are usually designed for lower bandwidth than the core routers but they need to process packets at higher layers (up to the application layer) and/or to perform packet modifications (e.g., payload processing). Furthermore, the processing requirements in the core routers are well defined and standardized while the edger outers must support continually evolving requirements [25]. Figure 2.3 depicts a block diagram of a typical edge router. The router consists of a number of line cards that communicate using a switch fabric. The line card consists of network interface modules (line interface, framers, etc.), memories (SRAMs, DRAMs, CAMs), and a network processing platform. The network processing platform of the edge routers typically include a number of processors, some hardware accelerators (co-processors), memory and network interfaces, schedulers, and usually a control processor. The pool of processors is used to process the majority of the packets (also called data plane processing) while the control processor is used to process control packets (e.g., network management packets). The co-processors are used to accelerate some common payload processing such as checksum computation, packet encryption, or packet compression. The major challenge in the development of network pro-

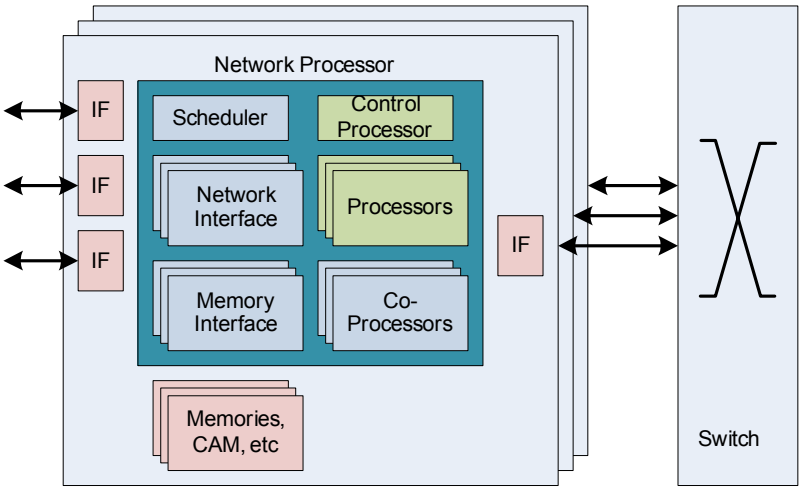

Figure 2.3: Typical edge router

cessors is to create a balanced system that will be able to meet the processing requirements of the network load. If the majority of the packets need header processing then the area of co-processors is wasted. If the majority of the packets need heavy payload processing then the processors may be under-utilized.

The typical data plane processing of an edge router is the following. First, the packets are classified into different flows. A flow is a class of packets that have common features. A feature can be the same destination address, the same source address, the same protocol type, etc. Each flow has its own processing requirements. For example, some flows need just header processing while other need also payload processing. The most common header processing functions in a router is the IP forwarding [26]. In this case, the header of the packet (IP destination address) is examined to determine the next hop address and the output network port of the packet using the IP routing table. The IP routing table contains the information to extract the next hop address based on the destination IP address of the packet using the longest prefix match. Longest prefix match means that instead of trying to find the exact match in the table, the lookup finds the entry for which the prefix match covers the most bits. To accelerate the lookup several algorithms have been proposed [27]. The most popular scheme for IP lookup uses a tree structure known a trie. A trie is an ordered tree data structure that is used to store data. A trie search uses either individual (binary trie) or multiple bits (multiple-trie) of the lookup key to navigate through the tree [6]. The search starts from the root and proceeds

towards the leaves until it find the maximum number of matched bits. Besides the IP lookup, the routers should also check and modify the packet's header (e.g., decrease the Time-to-Live (TTL) field) before the packet is sent to the output queue. Concisely, the major steps in routing a packet are:

- Remove the packet from the input queue
- Check the version of the packet (Version field)
- Verify checksum
- Lookup the destination IP address
- Update Time-To-Live (TTL) field
- Calculate the new checksum (incremental checksum)
- Insert the packet to the output queue

Some other flows may need also payload processing. For example, in case that the packets belong to Virtual Private Network (VPN) that supports the IPSec protocol [28] then the packets need to be encrypted before they are forwarded. IPSec provides security for transmission of sensitive information over unprotected networks such as the Internet. IPSec provide the following network security services:

- Data confidentiality: The sender can encrypt the packets before transmitting them across a network
- Data integrity: The receiver can authenticate packets sent by the IPSec sender to ensure that the packets have not been altered during transmission
- Data origin authentication: The receiver can authenticate the identity of the IPSec sender.

IPSec can be configured to run in two different modes; the transport mode or the tunnel mode. In the first case only the upper layers (TCP, UDP and the application layer) are encrypted while the IP header remains the same. In the second case, the IP header is also encrypted and it is replaced by a new IP header as it is illustrated in Figure 2.4.

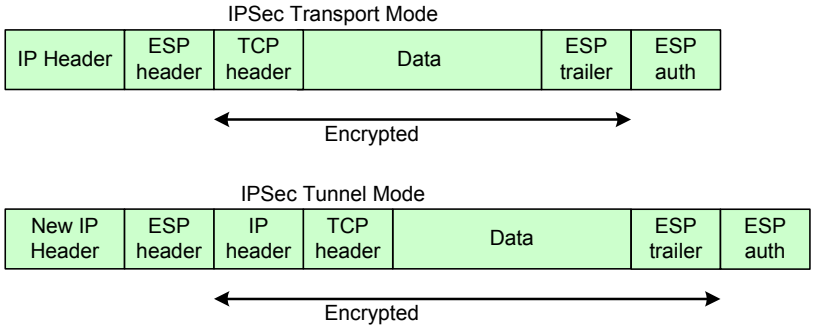

Figure 2.4: IPSec modes

There are several encryption algorithms that are used in IPSec, but the most common encryption algorithms are the Data Encryption Standard (DES) and the Triple-DES (3DES). DES is a block cipher algorithm that takes fixed-length data and transforms it into a cipher using a fixed-length key. The block size of DES is 64 bits and the key size is 56 bits. DES is symmetrical; the same key is used for the encryption and the decryption of the packet. Therefore, the same module can be used both for the encryption and the decryption. In case that higher security is required, 3DES can be used instead. In 3DES the original DES operation is performed 3 times with three different keys.

Another possible payload processing is the packet compression. There are several advantages of applying packet compression before the transmission of a packet [29]:

- Compressed packets consume less network equipment bandwidth
- Compression reduces fragmentation of packets, since payload length is decreased
- Line rate performance is significantly enhanced

Beside edge and access routers, the packet compression can be also applied to several other devices such as Storage Area Network (SAN) routers or wireless access point routers. In SAN network the use of packet compression can enhance significantly the performance of the storage cluster and can reduce the storage requirements [30]. Several protocols, such as Voice-over-IP (VoIP) or 802.11 (Wireless networks), need also packet compression. The compression can reduce the transmission time of the packet, which is extremely important in

wireless connections. In wireless devices the main constraint is the power consumption. Reducing the transmission-reception time of the network traffic can affect significantly the power consumption of the wireless devices. Although the additional processing power for the compression and the decompression may increase the power consumption there are several papers in the literature that show that the overall power consumption is reduced due to the reduced transmission time [31] [32].

There are two types of data compression algorithms, called lossy and lossless. Lossy data compression reduces the amount of data to a smaller size but it does not retain the integrity of the original data. This type of compression is usually applied to video and audio data in which some information can be eliminated without affecting significantly the overall data. On the other hand, lossless compression is applied in cases that the integrity of the original data must be preserved (e.g., text, databases, etc.). The most commonly used algorithm for lossless packet compression is the Lempel-Ziv (LZ) algorithm or any other variation of this algorithm [33] [34]. The LZ algorithm achieves compression by replacing portions of the data with references to matching data that has already passed through. The replaced data is encoded by a pair of numbers; the first one represent the length from the start of the compressed data and the second the number of matched characters. In the case of packet compression, the algorithm is applied from the Transport Layer (TCP or UDP) up to the Application Layer while the IP header remains intact [30].

2.3.2 Content-based Switches

As the network traffic keeps increasing there is the need for the ISPs and search engines to accommodate server clusters to face the increased demand. The most straightforward way to balance such a cluster is to use a switch that distribute the network workload based on layer 3 and layer 4 information (such as the IP address and the TCP or UDP port). However, the load balancing based on these layers is not efficient enough. First of all, the data must be replicated to each server; hence any change in the server content need to be updated on every server. Furthermore, the use of NAT proxies results to the same IP address for a large number of clients. Hence, the use of IP address to balance the system is not efficient. Moreover, the network balancing, using layer 3 and layer 4 information can not face more sophisticated requirements, such as content customization (e.g., content based on language or geographic region). The solution is the switching based on layer 7 (content) information. Content-based switches (also called web switches) use information of the application layer to

perform the distribution of the workload, such as HTTP cookies, URL strings, or Secure Socket Layer (SSL) session IDs. Figure 2.5 depicts the topology of a content-based switch in a server farm. The switch is placed between the server cluster and the Internet. Based on the type of application the content-based switch can forward the client's request to a specific server cluster (e.g., a secure server used for e-commerce, a streaming video server, or a simple web server). The web switch has a common virtual IP address that is seen by the external world, while internally each server has its unique IP address. Layer 7 load balancing provides 3 major advantages according to Cisco [35]:

- Scalability and acceleration of the application
- Persistent user sessions on a server
- Content customization based on user profile

The **scalability** is achieved since each web request is forwarded to a separate cluster. Therefore, in a case that a specific application (e.g., streaming video requests) has higher requirements, only the specific cluster has to be scaled to face the increased workload. Layer 7 load balancing based on the URL (Uniform Resource Locator) string can be performed either by distributing the data based on the directory domain or based on the file type (e.g., html, image, video, script). Load balancing based on the URL can also provide faster web response (download time) because of specialized application server. Furthermore, layer 7 load balancing provides **session persistence**. Session persistence can be used to forward the traffic of one client always to the same server in which previous data (transactions) are stored. Session persistence can be based on an HTTP cookie, a URL string or an SSL session. Finally, content-based switching can also provide **content customization** based on user profile, language or geographical region. For example, connections that come from a specific region can be forwarded to the server cluster that stores the content in the specific language for that region.

Web switches were initially implemented in software running on general purpose computers. The main drawback of using a software web switch for web balancing is the additional latency that is introduced. After a web switch has identified the corresponding server, the packets have to travel through the web switch to change the packet's header. The latency is mainly introduced because of the protocol stack processing. The packets have to travel until the upper layer where the switch module will process and forward the data as it is

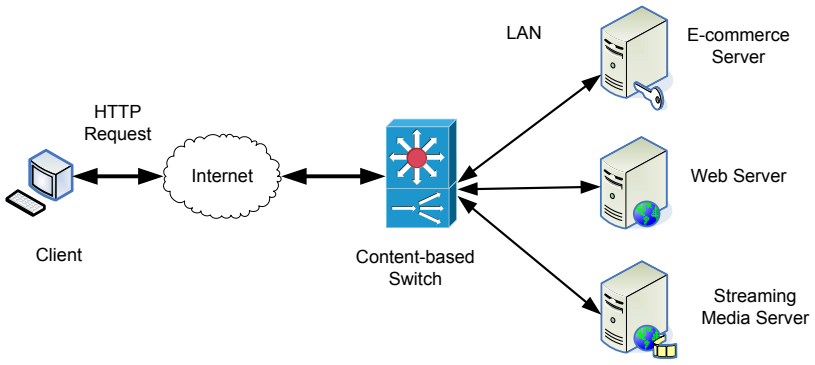

Figure 2.5: Content-Based Switch

depicted in Figure 2.6 (TCP Gateway). But after a connection has been established between a client and a server, the processing requirements are usually some header modifications. Hence, the TCP splicing scheme was introduced in 1998 [36]. The TCP splicing is used to forward the packets at the IP layer (layer 3) instead of the upper layers, after a connection has been established. Hence, the latency is reduced significantly and the sustained throughput is increased. Figure 2.7 depicts the TCP splicing in the network level. The client first establishes a connection with the web switch and then the web switch establishes a connection with the server. After the TCP splicing the packets are forwarded through the web switch with some header modifications.

Three schemes have been proposed in the literature for the implementation of a content based switch. A software approach (using a general purpose processor) provides great flexibility but has limited performance. A hardware approach (ASIC) has increased performance but lacks of flexibility. The most recent scheme is the use of network processor that combines the flexibility of processors with the throughput of the ASIC using basic hardware coprocessors. However, these coprocessors are usually modules that are used in general packet processing (CRC, header checksum, etc.) and can not be used for dedicated functions such as payload processing or URL parsing. Furthermore, the network processor platforms incorporate multiple instances of specialized processors that are programmed using vendor specific language (e.g., the microcode used by the micro-engines processors in the Intel's IXP network processors).

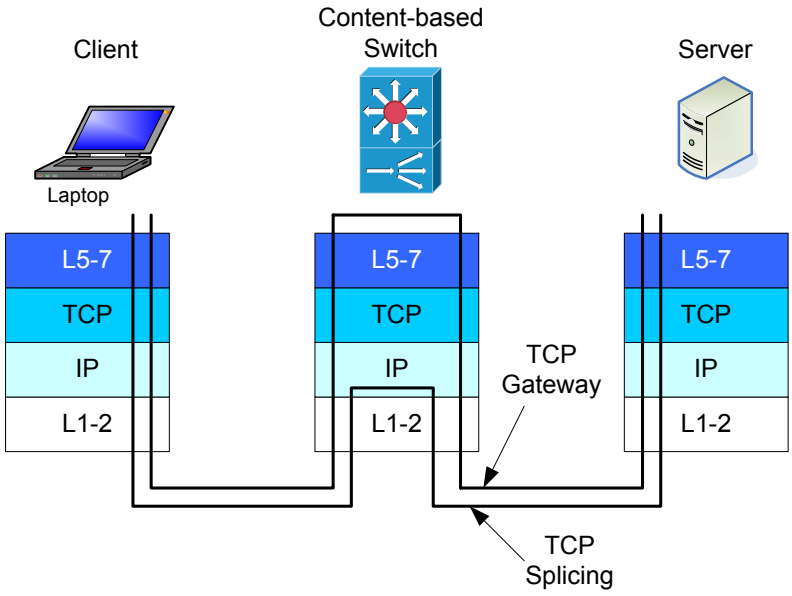

Figure 2.6: TCP Gateway and Splicing

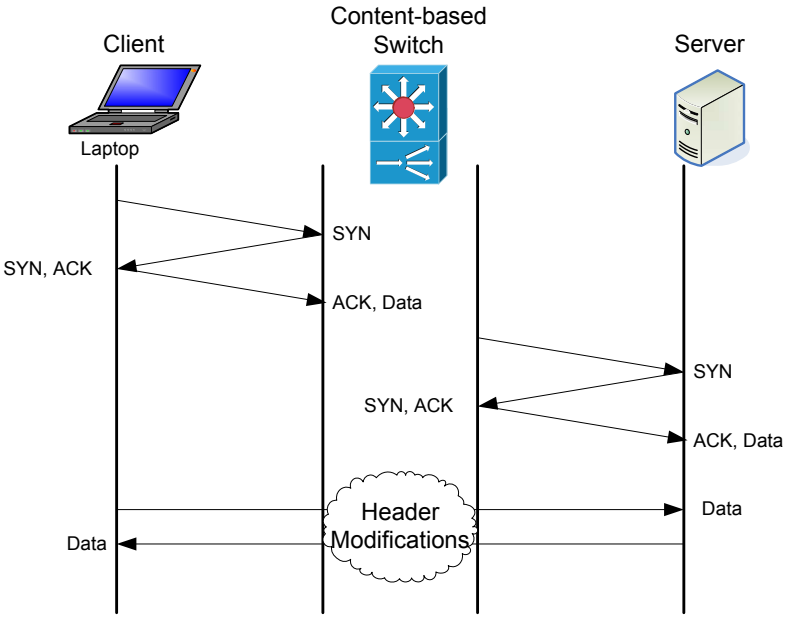

Figure 2.7: TCP Splicing

In [37], a content-based switch is presented, that has been implemented in hardware. The system consists of input and output controllers to process the data packets, while the parsing of HTTP headers and URL based routing is performed by a general purpose processor. This scheme can support up to 500 ops/sec with 50msec latency but it can not be extended or modified in the data plane path.

In [38] [39] a content based switch has been proposed targeting the Intel IXP2400 network processor. The IXP network processor consists of 8 multi-threaded micro-engines and each micro-engine is able to support up to 8 threads. The application has been partitioned into 4 micro-engines. Two of them are used for the input and output packet processing, one is used for the client's packet processing, and the last one for the server's packet processing. The proposed system supports also TCP splicing to accelerated the throughput and reduce the latency. Furthermore, a similar design implemented in software (Linux) has been used for comparison. According to this paper the network processor scheme can support up to 700Mbps throughput for 1 MByte of requested file size, while the Linux scheme can support up to 320 Mbps for the same requested file size. Using 1 Kbyte requested file size the throughput is 8.2 and 46.6 Mbps for the Linux-based and the network processor-based scheme, respectively.

In [40], a similar scheme is proposed based on Intel network processors but in this case the data processing is performed by the micro-engines while the control processing is performed by a host processor. Furthermore, the general purpose Strong ARM is used to control the micro-engines. This scheme is able to achieve 3.47 Mpps (Million packet per seconds) using the older IXP 1200 network processor.

In [36], the TCP splicing mechanism is proposed and a software implementation under the BSD operating system (BSDI BSD/OS 3.0) is performed targeting a general purpose processor. The proposed implementation is able to sustain up to 70Mbps (in 1998) with almost 85% CPU utilization. The mean forwarding latency was 102 ms while the latency of a simple forwarding scheme is 92 ms.

In [41], there is one more implementation of the TCP splicing as a Linux kernel module. In this case, the switching is performed based on the URL and then the connection is spliced between the clients and the servers. The system was based on Fast Ethernet Network Interfaces (100Mbps) and the URL-aware switch was based on a Pentium III 555MHz CPU. According to the paper, the system was able to issue 233K connection with 1KB file size with 61.32%

CPU utilization.

2.3.3 Queue Scheduling

The increase of the bandwidth in the Internet has created the need not only for more powerful header and payload packet processing, but also the need for efficient congestion management and congestion avoidance in the network devices. To face the congestion in the network, several scheduling schemes have been proposed. Packet scheduling can be used to provide Quality-of-Service (QoS) guarantees in terms of delay and bandwidth to the network flows. A flow is a stream of packets that belong to the same class and require the same grade of services. The requirements in terms of latency, jitter and bandwidth are different for each flow; hence, packet scheduling algorithms are required. The algorithms vary from simple algorithms (such as the round robin algorithm), to more advanced algorithms that can supports queues with different weights.

There are several algorithms that have been proposed for the scheduling of queues with different priorities [42] [43] [44] [45] [46]. The simplest algorithm for scheduling of queues with different priorities is the priority queuing (PQ). In this case a queue is selected only if all queues of higher priority are empty. The PQ scheduling allows routers to organize the packets to several classes of traffic. For example, the video and the voice traffic can get priority over non-time critical applications such as file transfers and web browsing traffic. The implementation of this algorithm is straightforward but this scheme has the disadvantage of starvation when the high priority queues are fully utilized. The Weighted Round Robin (WRR) can address the limitation of the Priority Queue scheduler by visiting and serving each queue a number of times depending on the weight of each queue. The main limitation of the WRR is that it provides the correct percentage of bandwidth to each queue only when the average size of the packets is the same for all queues. The limitation was addressed by using a new scheme; the Deficit Weighted Round Robin (DWRR). The DWRR can be easily implemented both in hardware and in software, it can support a large number of queues but it has the drawback that it does not provide end-to-end delay guaranteed as precise as other scheduling schemes.

A more sophisticated algorithm is the Weighted Fair Queuing (WFQ) in which a virtual time is used to schedule the queues. The WFQ is closer to the ideal scheduler, the Generalized Processor Sharing (GPS). GPS is an ideal scheduling policy that provides exact fair share allocation. The algorithm is based on an idealized fluid-flow model. However, in real systems, the packets can not be

Table 2.1: Comparison of Schedulers [3]

Scheduler	Latency	WFI	Complexity
GPS	0	0	-
DRR	$(3F - 2\phi_i)/r$	$O(N)$	$O(1)$
WF2Q+	$(L_i/r_i) + (L_{max}/r)$	$O(L_i/r_i)$	$O(logN)$

N is the number of queues, F is the frame size, ϕ_i is the weighting factor of session i in bandwidth allocation, L is the packet length and r is the minimum allocated rate for session i

divided into smaller units therefore only approximations of this algorithm can be implemented. The WFQ algorithm [47] is very close to the GPS, however the implementation complexity of the original scheme was high since it had to keep information for every packet in the queues. Therefore, a more accurate and simple algorithms was presented in [48], called Worst-case Fair Weighted Fair Queuing (WF2Q+). In this case the system has to keep information only for the head packet of each queue. As, a result the WF2Q+ can achieve better results in terms of quality but it does not scale well for large number of queues [6].

Table 2.1 shows the comparison of the ideal GPS, the DRR and the WF2Q+ scheduler in terms of latency, fairness (WFI: Worst-case Fair Index), and complexity. As it is shown, the DRR has the lowest complexity ($O(1)$) while the complexity of the WF2Q+ is $O(log(N))$. On the other hand, WF2Q+ is more fair and can guarantee lower latency compared to the DRR scheduler [3] [49].

2.4 Network Processing Architectures

The diversity of the network traffic requirements based on the network that it belongs has lead to a great diversity of architectures that are being deployed by network processor vendors [7]. One of the main categorizations of network processors is performed based on the interconnection scheme of the embedded processors.

Interconnection. The processors can be connected in a parallel, a pipeline or a hybrid scheme. The parallel scheme means that all processors are connected in parallel, have all access to the same resources (memory, accelerators) and usually all processors run the same application (Run-to-Completion). The main advantage of this scheme is that the programmer does not have to partition the application is several steps which provide faster development time. The main

drawback of the parallel scheme is that synchronization must be preserved between the processors. Although that the network processing applications are inherently parallel (each processor can handle a different packet) a basic synchronization is required in many applications (e.g., traffic metering, management, resource sharing, etc.). Another drawback of the parallel schemes is that all of the processors may compete for the same resources which may lead to unpredictable performance. But this drawback is easy to solve by providing enough bandwidth between the processors and the resources. On the other hand, in the pipelined scheme, the processors are connected serially and each processor is assigned to handle specific parts of the application. The pipelined scheme can be either homogeneous or heterogeneous. Homogenous pipelined processors mean that all the processors are the same, while in the latter case specialized processors are used for each pipeline stage. For example, a pipeline stage can be used for classification, another one for packet modification and a separate one for packet scheduling. The main drawback of this scheme is that if the pipeline stages are not well balanced, the performance can be compromised. For example, if the packet modification on same packets takes much more time than the previous stage then the processing time of the other packets can be affected. Finally, the hybrid scheme assumes a pipeline scheme of parallel connected processors. This scheme inherits all the advantages and drawbacks of the other two systems.

Instruction set. Another categorization of network processors is performed based on the architecture and the instruction set that is used. The main characteristic of network processing applications is the simplicity and the low usage of floating point operations, even in the higher layer functions. Therefore, the most common architecture used by the network processors are the Reduced Instruction Set Processors (RISC). The RISC processors operate at higher clock frequencies and consume less power compared to a Complex Instruction Set Processors (CISC). CISC processors are mainly used for general purpose processors and have specialized instructions for complex operations. Therefore, the CISC processors are mainly used for the control-plane processing. The protocol processing applications are generally simple and can not take advantage of the extra instructions of a CISC processor, therefore RISC are the dominant architecture in the area of network processing (in the data-plane processing). In many cases, the RISC processors are optimized for network processing application by adding barrel shifters, bit-level manipulation instructions, etc. Nevertheless, there are some network processor vendors that prefer other architectures. Some vendors use Very Large Instructions Word (VLIW) architectures (e.g., Agere NP [50]) while some others use dataflow architectures (e.g.,

EZChip [51]). In the case of VLIW, the inherent parallelism of packet processing is exploited by issuing many instructions in several functional units at the same time. The main drawback of this scheme is the additional complexity on programming and compiling these processors. The dataflow architectures try to exploit the inherent sequential task flow of the network processing applications. The majority of these applications can be divided in specific sequential tasks (e.g., classification, modification, and scheduling). The dataflow architectures use specialized processors with optimized instruction sets for each of these tasks to accelerate the packet processing (e.g., a specialized processor for classifications, a second one for modifications, and a third processor for scheduling). The specialized processors can achieve higher throughput and deterministic performance. However, the programming of these processors is extremely difficult, since if the tasks are not balanced equally, the over-loaded stage will become the performance bottleneck.

2.5 Related Work

The first step in the design of a network processor is the design space exploration. The exploration is used to expose the effect of several design trade-offs to the features of the network processor, such as the performance, the area, and the power dissipation. In this thesis, a design space exploration has been developed and has been used for the proposed reconfigurable platform. Section 2.5.1 presents several design space exploration frameworks for network processing platforms that have been proposed by the research community and the industry. Section 2.5.2 presents the related work in the area of using reconfigurable logic platforms for network processing applications. Furthermore, the section discusses the difference of the proposed design space exploration scheme and the proposed reconfigurable platform with the other approaches.

2.5.1 Design Space Exploration

The use of design space exploration tools is one of the first steps when designing the architecture of the network processor. The design space exploration can be performed using several methods [52]. The fastest method is achieved using analytical approaches. In these cases, the system is described in analytical equations therefore the time to perform the exploration is significantly small. The major drawback of the analytical approaches is that they usually provide less accuracy than simulation-based exploration tools. On the other

hand, the design space exploration using simulation models provide higher accuracy but the simulation time is much longer than the analytical exploration. The simulation models that are used for the design space exploration can cover several levels of the system, such as the instruction-set models, cycle-accurate models, or RTL models. Another scheme for the design space exploration is the combination of the simulation-based and the analytical methods. In this case, the system is described in analytical equations but the characteristics of the system are extracted from the simulation models. This scheme provides fast design space exploration and high accuracy.

In the past, several tools have been presented that can be used for the design space exploration (DSE) of network processing platforms [53]. EXPO [54] is a DSE tool that uses the theory of the arrival and service curves to model the operation of a network processor. The computation complexity in this case is too expensive, thus they use a piecewise linear approximation of all arrival and service curves. The network processors can be modeled in a task graph and given the mapping of tasks to available resources it can estimate the Pareto-optimal solution for access and backbone networks. The tool is restricted to model a system with a common bus that every resource is attached to this bus.

In [55], an automated exploration framework is developed that is used in a soft multi-processors platform. The application is modeled as a task graph and each task is allocated in one of the processors. The tool is used to find the optimum partitioning of the IPv4 packet forward application into an array of processors, but this tool does not incorporate co-processors which are essential parts in network processors.

In [56], a design space exploration is performed using several parameters of a general-purpose processor such as the processor clock rate, the instruction and data cache size, the area and the memory access time. The CommBench benchmark [44] is used to illustrate the difference of the optimum configuration using packets that only need header processing versus packet that need also payload processing. The model is applied both to a single processor and multiple processors.

In [57], a design space exploration is performed for network programs on different architectures. The compared architectures are a speculative super-scalar processor, a fine-grained multithreaded processor, a single chip multiprocessor and a simultaneous multithreaded processor (SMT). The benchmark that it was used includes IP forward and MD5 and DES encryption processing. In [58] a design space exploration of the System-On-a-Chip (SoC) communication of the components is performed. The number of busses and bridges are investi-

gated in order to find the optimum configuration for a given graph of connected modules.

Finally, STMicroelectronics has presented a system-level exploration platform for Network processors called StepNP in [59] [60]. In that case the platform contains multi-threaded processors connected with a custom network-on-a-chip. The system is modeled at the functional and transaction levels and not at a cycle-accurate level.

The proposed design space exploration tool, which will be presented in Chapter 3, is based on the formulation of the system as an optimization problem. The system is described in equations as a single-objective optimization problem with several constraints. The constraints and the parameters of the system are extracted from the implemented components of the system. Therefore, the proposed scheme achieves the speed of the analytical design space exploration tools while the accuracy remains close to the accuracy of the cycle-accurate implementation models.

2.5.2 Reconfigurable-based Network Processing Platforms

The use of dynamically reconfigurable systems to improve the performance of the system in many applications has increased over the last years. This section presents the research in dynamically reconfigurable systems in the domain of network applications. In [61] [62], a reconfigurable programmable router has been introduced that is mainly used in active networks. Active networks are networks in which the packets are processed by emerging protocols that are either included into the packet or can be downloaded dynamically into the router. The system consists of general-purpose CPUs and hardware plug-ins. Each plug-in has an SRAM and an SDRAM interface to communicate with the memory and a custom interface to a 32-bit wide ring in order to communicate with the CPUs and the other plug-ins. These plug-ins are dynamically configured kernel modules used to process the active packets, that can be downloaded by a trusted server. The system has been implemented onto two FPGAs, one used as the network interface device and one used as the host of the hardware plug-ins.

In [63], a reconfigurable system called Programmable Protocol Processing Pipeline (P4) has been introduced. In this case, a set of FPGAs is used in a pipeline manner in order to accelerate packet processing. Each device has a FIFO buffer associated with it that is used to load and store the processed packets. The devices are connected using a switching array that can include or

exclude processing elements. As an example, Forward Error Correction (FEC) is used as a protocol processing function. Although FPGAs can be reconfigured dynamically, in this paper only the performance evaluation of a static design (and not of a dynamically reconfigurable device) is presented.

In [64], an architectural concept for network processors called FlexPath is introduced. The FlexPath scheme use both processors and hardware coprocessors to accelerate network processing functions. Special modules for basic network processing functions have been implemented such as a preprocessing module, post-processor, and path dispatcher. Packet with standard processing requirements may bypass completely the central processing complex and can be fully processed by the hardware units. Although that the proposed scheme is targeting FPGAs, there is no exploitation of dynamic reconfiguration.

In [65], a reconfigurable network coprocessor platform is presented called DynaCore. In this paper an FPGA-based platform is presented that can accommodate hardware accelerator units. The platform includes a dispatcher that is used to forward the incoming packets to the hardware acceleration units. The system consists only of hardware acceleration units without presenting a connection of the hardware units with the general-purpose processing elements used for the remaining header processing.

In [66] [67], a reconfigurable platform is introduced targeting mainly active networks. This system consists of software and hardware parts. The software part is a set of kernel and user space modules running on a Linux PC. The hardware part consists of an FPGA device that is used to load the required processing modules. When a packet is received it is checked whether a passive or an active packet is. For active packets the system checks whether the required hardware for this application is already present in the device. Otherwise, it can request the bitstream (the hardware configuration file) for the specific active packet. When a new bitstream is received for an active packet, the bitstream is authenticated, decrypted and checked for integrity and then is used for the configuration of the device.

In [68], the design and analysis of a network processor using accelerators in reconfigurable logic is presented. In this paper, two different approaches are presented. In the first case, each task is mapped to a general-purpose accelerator. In the second case, different accelerators are used for different tasks that can be dynamically reconfigured on the device. The paper showed that the use of reconfigurable modules can improve the execution time by about 20 times. The system has been evaluated in three applications; tree lookup,

pattern matching, and network intrusion detection.

In [69], the PLATO platform is presented. PLATO is a reconfigurable active network platform which provides four physical connections for ATM networks. Two applications were ported to this platform; an Active 4x4 ATM Switch and Wormhole IP over ATM routing filter.

In [70], a programmable protocol processor for network terminals has been presented in Linkoping university. In this case, the network processing requirements for terminals has been identified and a platform has been developed that use several dedicated hardware blocks to accelerate the common processing tasks of network terminals. In [71], a data-flow based architecture has been proposed for network protocol processing also in Linkoping university. In the case of the data-flow approach, the processor does not operate on data stored in the memory. Instead, the processor operates in data-flow fashion. This scheme reduces the number of load and store instructions, therefore reducing significantly the processing time of the packets.

2.6 Conclusions

In this chapter a closer look of the network processing domain has been presented. One of the most important issues in designing a network processing platform is studying the targeted applications and the targeted position of the platform in the network. For example, edge routers have different processing requirements than a core router. Therefore, the targeted application has a major impact on the selection of the network processing platform. Furthermore, a short presentation of the available architectures for network processing platform has been presented. Finally, the related work in the area of routing, content-based switching, and queue scheduling has been presented. The related work in the area of design space exploration tools for network processors and network processing using reconfigurable logic has also been presented.

Chapter 3

Reconfigurable Network Processing Platform

The previous chapters presented the requirements of the network processing applications, especially in the edge and enterprise networks. The devices used in these networks have to be both flexible (programmable) and powerful (able to sustain high throughput and high processing complexity). Furthermore, it has shown that the FPGAs can provide both flexibility (through programmability and reconfigurability) and performance (through the inherent hardware nature). Therefore, the reconfigurable platforms seem to be ideal platforms for edge and enterprise network processing applications.

This chapter presents the proposed reconfigurable network processing platform and the design flow that is used to implement it. Section 3.1 presents the organization of the platform and a short description of the reconfigurable modules. Section 3.2 presents the integrated design flow that is used to design and configure the platform. Section 3.3 presents the formulation of the platform as an optimization problem and the design space exploration that is performed for this platform. Finally, Section 3.5 presents the conclusion and the summary of this chapter.

3.1 Proposed Reconfigurable Network Processing Platform

To design an efficient network processing platform for the edge and enterprise networks several issues have to be considered. The platform should include processors that are specialized for network applications but also be programmed using widely used programming languages and not vendor specific languages. Furthermore, the platform should also incorporate specialized co-processors that can be used to accelerate the payload processing. The platform should also be flexible in order to support future protocols and applications. Due to the parallelism of network processing, the platform should also support the easy development of multi-processing programming.

Figure 3.1 depicts the organization of the proposed reconfigurable network processing platform. It consists of a pool of RISC processors with separate instruction and data memories (Harvard architecture), a shared bus, a Direct Memory Access (DMA) engine, a queue scheduler, a transactional memory controller and a number of co-processors. The processors can be either hard-core (embedded in the device by the manufacturer) or soft-core (delivered in a hardware description language that can be incorporated to the device by programming/configuring the device). The co-processors can be connected either directly to the processor with specialized interfaces or to the shared bus. The first option provides faster communication (lower latency) between the processor and the co-processor. On the other hand, the co-processors that are attached to the shared bus can have higher bandwidth interacting with the memory (either on-chip or off-chip memory) using the DMA engine without the utilization of the processor. Furthermore, the co-processors that are attached to the bus can be shared by many processors contrary to the co-processors attached directly to the processor. The shared bus that is used can sustain up to 4Gbps bandwidth [72] which is sufficient for the class of network applications and devices that are examined in this thesis (edge routers, content-based switches, etc.). The proposed platform could be further scaled to support higher bandwidth by the replacement of the shared bus with a Network-on-Chip (NoC) interconnection scheme.

The co-processors are used either for header processing (e.g., checksum modules) or for payload processing (e.g., encryption, compression, media (either video or image) transcoding (change of the frame rate, resolution, etc.), and CRC calculation). Furthermore, the co-processors can be either static or dynamically reconfigurable. In the first case, the co-processors do not change

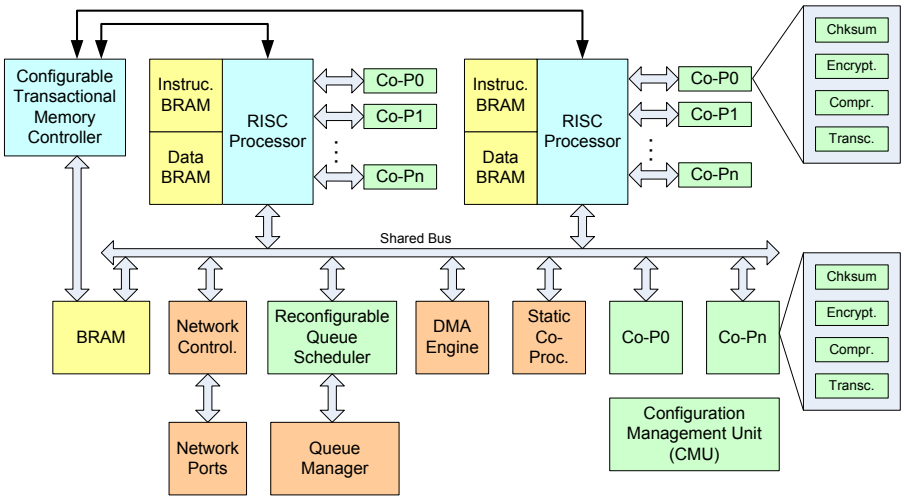

Figure 3.1: Reconfigurable Network Processing Platform

over time. In the second case, the function of the co-processors can be reconfigured at run-time from a library of co-processors. A special unit is used to change the co-processors attached either to the bus or the processors (Configuration Management Unit, CMU). The CMU monitors the network traffic and based on the network features (e.g., distribution of flows) reconfigures the co-processors to perform another task. The co-processors that have been used for the performance evaluation of the proposed platforms are described in Chapter 5.

The platform includes also a reconfigurable queue scheduler. As it was mentioned in Section 2.4.3, there are several algorithms for queue scheduling, each with its own features. Some of them can be scaled easily to support multiple queues without significant area overhead but with average latency results. On the other hand, some other algorithms are better in terms of latency and bandwidth allocation but they add significant area overhead when they are scaled to support large number of queues. Therefore, a reconfigurable queue scheduler has been developed in which the operational algorithm depends on the number of active queues.

In this platform, the use of parallel interconnection of the processors has been chosen. As it was mentioned in Section 2.3, the parallel interconnection provides an easier programming framework for the programmer. The applications do not have to be divided into separate tasks for each processor with equally

processing time; a time consuming process. Therefore, all of the processors execute the same application (Run-to-Completion). Furthermore, it has been shown that for forwarding applications, which is the typical application in edge and access routers, the pool topology performs better than the pipelined topology [73].

A major challenge in the programming of parallel processors is the synchronization. The most common scheme for synchronization is the locking mechanisms. Using this scheme, the processor should first acquire a lock before it access shared regions of a memory and after the processing the lock is released. However, there are several drawbacks using this mechanism. The most important drawback is that the system can end to a deadlock if the order of the locks is not taken care of. Consequently, the programming of parallel programs is usually difficult and error-prone. To facilitate the programming framework, the transactional mechanism has been proposed as an alternative method. In the proposed reconfigurable platform, a configurable transactional memory controller has been developed. The transactional memory controller is configured based on the number of processors and the applications requirements (order of shared variables, etc.).

Furthermore, the DMA unit has been modified to support multiple processors without wasting time for the locking of the unit and the polling of the state of the DMA unit. The modified DMA unit is illustrated in Figure 3.2. In the original DMA unit, each processor had first to check the status (idle or busy) of the DMA unit and then access the registers of the unit. Furthermore, before accessing the registers, the processors had to lock them and after the access to release them. The modified DMA unit provides faster access of the processors to the DMA unit without requiring any locking schemes. Each processor uses a special interface to forward the DMA commands (source address, destination address and transfer size) to a specific FIFO. A round robin scheduler is used to arbitrate the FIFOs and forward them to the original DMA unit. In case that the FIFOs are full, the processor is blocked until there is empty space.

The following section presents the design flow to find the best configuration for each application and workload for the platform. Since the implementation and simulation of every configuration is time consuming, an analytical method must be used to easily explore the different configurations. This problem has been formulated into an integer non-linear programming model to perform an analytical design space exploration.

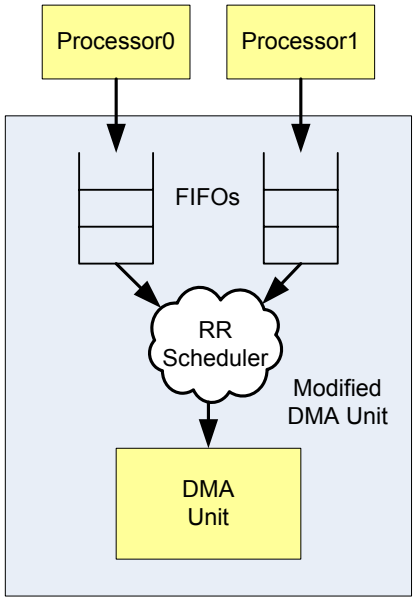

Figure 3.2: The modified DMA unit

3.2 General Design flow

Figure 3.3 depicts the high level design flow of the reconfigurable network processing platform. It consists of three steps; the first two are performed at design time and the third one is performed at run-time:

- Traffic characterization
- Design Space Exploration
- Run-time Adaptation

The first step is to examine the network traffic for the specific application and estimate whether there are network traffic patterns with different processing requirements. In [19], there is a study about how the network traffic changes during the day. For example, during the working hours the majority of the packets are http packets while during the night the majority of the packets are from P2P applications (file sharing applications). The second step is to perform a design space exploration to find the optimum configuration for each

network traffic pattern. After the design space exploration, a bounded number of configurations should be selected that can be used in specific network distribution flows taking into account the reconfiguration overhead. Finally, the configuration management unit (CMU) selects the best possible configuration depending on the configuration overhead, the network characteristics (flow distribution, average packet size, etc.), and the network variation (how often the network characteristics change).

The design space exploration and the performance evaluation can be performed using either real network traffic traces or simulated models of the real traffic. Using models of the network traffic provides faster performance evaluation but it lacks of accuracy. On the other hand, the use of real traffic traces provides a more realistic workload but the performance evaluation can be slower compared to simulated models. In the case of the simulated model network traffic there are many proposals by the network simulation community. In [54], the network traffic is being formalized by the use of arrival curves which provide deterministic bounds on the workload. The arrival curves represent the number of packets that might arrive from a specific flow within any interval of time. In [74], the network traffic has been generated using the Holt-Winter equations as applied to networking. In this case the network traffic is decomposed to three variants: the baseline traffic, the linear trend traffic and the seasonal trend traffic. In [18] and [17], there is a detailed analysis of how the network traffic changes during the day or the week. The changes are in terms of the application, the protocol and the size of the packets. The main features of the network traffic as have been characterized by [16] are the following:

- Average utilization may be very low
- Peak utilization can be very high
- Packet traffic is bursty

The network traffic fluctuations in terms of application illustrate that the use of an adaptive network processor could improve the performance of a router. For example, the network processing platform can be reconfigured to handle high traffic of VPN connections by increasing the number of encryption modules, while in the case of high number of wireless connections the platform can be reconfigured by increasing the number of hardware accelerators for compression.

The modeling of the network traffic and the design space exploration is performed at design time. The results from the design space exploration are used

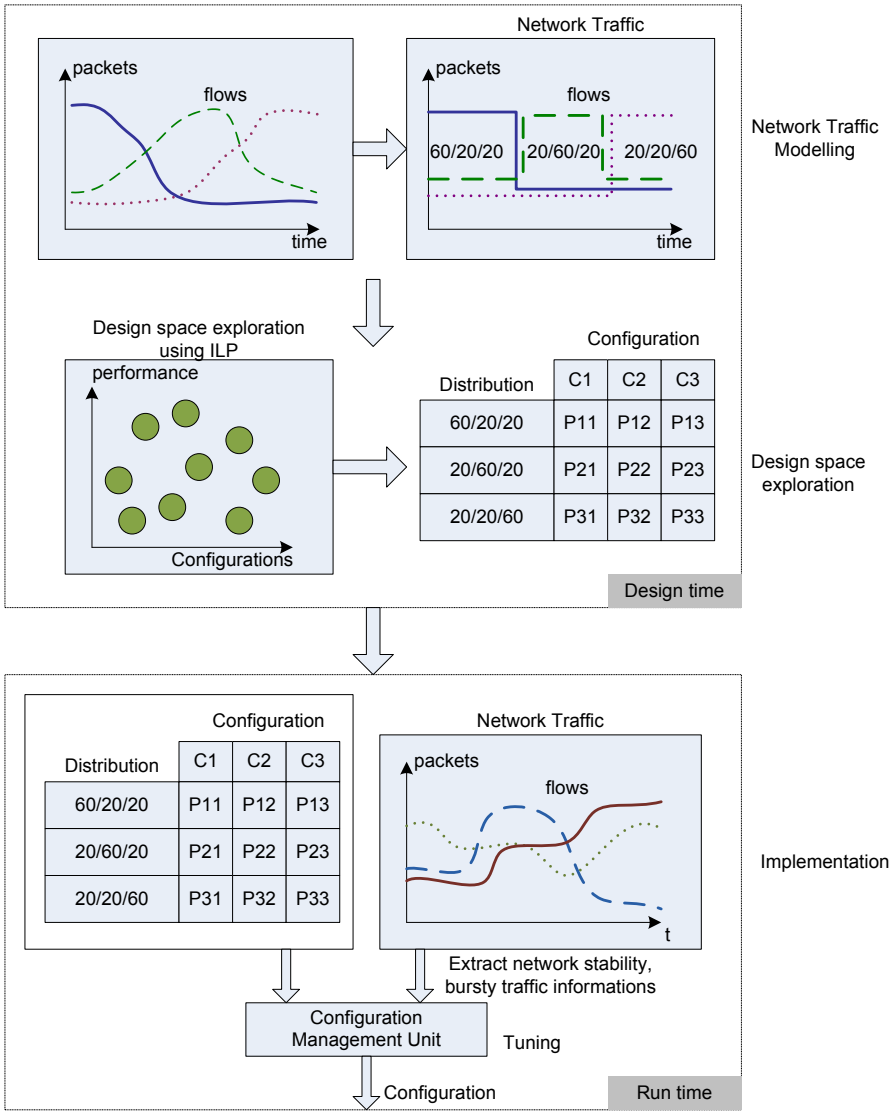

Figure 3.3: Design flow

for the development and the selection of several configuration of the reconfigurable platform. During run-time, the platform selects the configurations based on the measurements of the real network traffic using the Configuration Management Unit.

3.3 Design Space Exploration

After the characterization of the network traffic for the target application, a design space exploration of the reconfigurable platform must be performed at design time. The goal is to find the configuration of the platform with the minimum processing time of the network traffic workload in a given constraint area. As an example, the current section presents the formulation for an edge router in which the packets are classified in three different flows as it is shown in Figure 3.4. In the first flow, the packets are simply forwarded (plain traffic). In this case the packet is only processed by the processor. In the second flow, the packets belong to VPN connections supporting the IPSec protocol, therefore, the packets have to be encrypted/ decrypted depending on the direction before they are forwarded. The payload processing is performed by hardware co-processors while the header processing (IP forward) is performed by the processor. There are several algorithms that can be used for the encryption of the packets in the VPN connections such as the DES, and the AES for confidentiality and the SHA1 for integrity protection [75]. In this case, the DES encryption algorithm has been selected for the performance evaluation. In the third flow, the packets have to be compressed to according to the IPComp [76] protocol using the Lempel-Ziv compression algorithm. Again, the payload processing is performed by hardware co-processors and the header processing is performed by the processor. The variable that should be optimized (minimized) is the processing time of the network traffic as it is shown in Equation 3.1. Since the hardware processing can be performed in parallel with the header processing, the time that should be minimized is the maximum of the three execution times: the execution time of the processor (Equation 3.2), the execution time of the encryption unit (Equation 3.3) and the execution time of the compression unit (Equation 3.4). The constraints of the system are:

- The number of processors
- The number of co-processors directly connected to the processor
- The number of co-processors attached to the bus

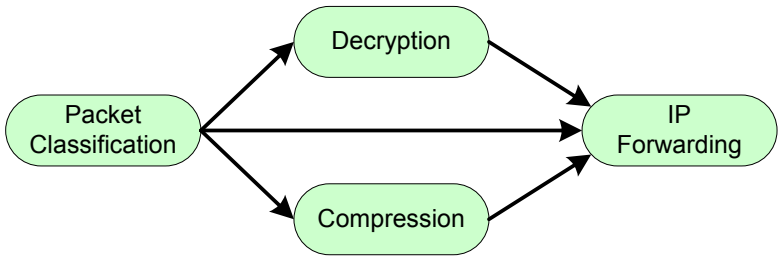

Figure 3.4: Processing flows for an edge router

- The bandwidth of the bus
- The available processing power of the processor
- The available area

$$ProcTime = Max(ProcT_{processor}, ProcT_{DES}, ProcT_{LZ}) \tag{3.1}$$

$$ProcT_{Processor} = \frac{\%Plain \cdot T_{plain} + \%Encr \cdot T_{encr} + \%Compr \cdot T_{compr}}{N_{processors}} \tag{3.2}$$

$$ProcT_{DES} = \frac{\%Encrypt \cdot AvgPacketSize}{N_{DES} \cdot Throughput_{DES}} \tag{3.3}$$

$$ProcT_{LZ} = \frac{\%Compress \cdot AvgPacketSize}{N_{LZ} \cdot Throughput_{LZ}} \tag{3.4}$$

The constraints are formulated in the following equations. Equation 3.7 states that the total throughput of the co-processors must not exceed the theoretical upper bandwidth of the bus. Equation 3.8 states that the headers of the packets that can be processed by the hardware co-processors must also be able to be processed by the total number of processors. Otherwise, the co-processors will be idle waiting for the processors to finish.

$$Area_{processor} \cdot Nprocessor + Area_{DES} \cdot N_{DES} + Area_{LZ} \cdot N_{LZ} \leq TotalArea \tag{3.5}$$

$$N_{processors} + N_{DES} + N_{LZ} \leq MaxN_{BusSlaves} \tag{3.6}$$

$$Throughput_{DES} \cdot N_{DES} + Throughput_{LZ} \cdot N_{LZ} \leq BusBandwidth \tag{3.7}$$

$$(\frac{Thr_{DES}N_{DES}}{AvgPacketSize} + \frac{Thr_{LZ}N_{LZ}}{AvgPacketSize})N_{CCpPacket} \leq N_{processor} \cdot ProcPower \tag{3.8}$$

where,

$ProcT$	Maximum Processing Time
N	Number of processors,DES,LZ modules
T	Processing time per packet
$Area$	Area in Slices
Thr	Throughput in Mbytes/sec
$\%(Plain, Encrypt, Compress)$	Percentage of Network traffic
$ProcPower$	Processing power (Available Cycles)
$N_{ccpPacket}$	Number of Clock cycles per packet

The co-processors can be attached either directly to the processor using specific interfaces (point-to-point) or can be attached to the shared bus. In the first case the data have to be transferred to co-processor through the processor, hence a lot of processor cycles are wasted. In the second case the DMA unit has to be initialized before the transfer, hence there is latency before the transfer. For example, in the case of the DES module connected in the shared bus the time to transfer the data using DMA and the time to transfer the data to the module using the direct bus is given by Equations 3.9 and 3.10, respectively. In networks that the packet size is small (e.g., 48 bytes for ATM networks) the units that are attached to the processor may have better performance than attached to the bus. In networks that the packet size is larger (e.g., in Ethernet networks the packet size varies from 64 bytes to 1500 bytes) the co-processors that are attached to the bus have better performance than those attached to the processor.

$$t_{Bus} = (nc_{DMAInit} + words \cdot nc_{perWordtransfer}) \cdot t_{cycle} \tag{3.9}$$

Table 3.1: Performance of the modules

$Module$	$Performance$
Microblaze	184 DMIPS
DES Unit	38MBytes/sec
LZ Unit	30MBytes/sec

$$t_{Direct} = words \cdot nc_{perWordTransfer} \cdot t_{cycle} \tag{3.10}$$

where,

nc	number of cycles
$words$	4 bytes
t_{cycle}	the period of the clock

The equations can be solved using any optimization tool such as the Excel Solver or the MPL Solver. A case study using the Xilinx MicroBlaze soft-core processor is performed using off-the-shelf modules for the encryption and the compression. The case study is performed using the parameters shown in Table 3.1:

Figure 3.5 depicts the optimized configuration for several workloads distribution for an average packet size of 512bytes for the MicroBlaze framework. The platform has been evaluated for 3 different flow distributions. In the first one, 60% of the traffic is packets that need only to be forwarded, 20% belong to Virtual Private Networks (VPNs) that need either encryption or decryption, and while 20% of the packets belong to flows that need compression. The MicroBlaze and the co-processors are clocked at 100MHz. All of the payload co-processors have been attached to the bus, since the average packet size of Ethernet traffic does not justify the use of these modules connected directly to the processor. In this case, the communication overhead to transfer the packet to the modules through the processor would result in many wasted processor cycles. On the other hand, the header checksum module has been connected directly to the processors using the specific interface. As depicted in Figure 3.5, the optimum configuration is achieved using 3 processors, 1 encryption unit and 2 compression units when the majority of the packets need plain forwarding. When the majority of the packets need encryption, the optimum configuration is achieved using 2 processors, 3 DES modules and 1 LZ modules. In the last case (20/20/60 distribution), the optimum configuration is achieved using 2 processors, 1 DES module and 3 LZ modules. The throughput of the

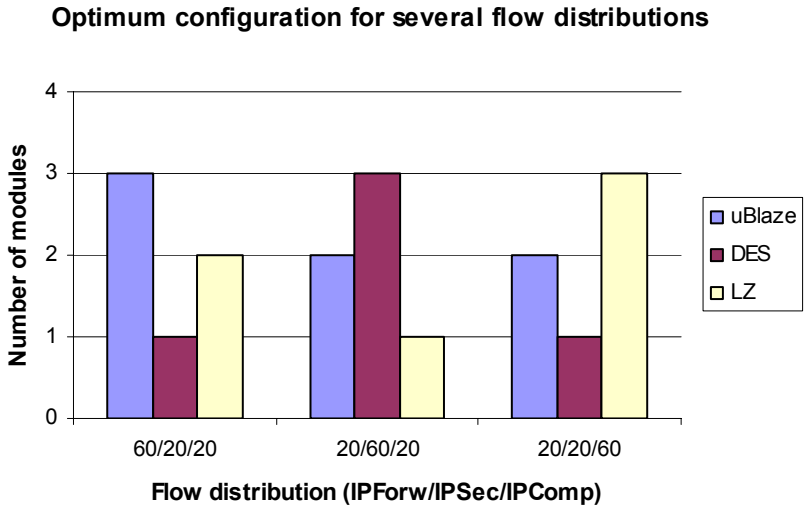

Figure 3.5: Optimum configuration for the MicroBlaze for several workloads

LZ modules is less than that of the DES modules; therefore more co-processors for compression are used in the case of compression-optimized configuration than corresponding for the encryption-optimized configuration.

In order to present the turning points for several configurations, a complete Design Space Exploration using the optimization equations (3.1)-(3.8) has been performed for each variable (number of processors, DES modules, and LZ modules). Figure 3.6 depicts the optimum configuration for every combination of traffic flow mix (in steps of 10%) using 512 bytes average packet size. Each point has 3 circles representing the optimum number of processors (uBlaze), DES modules, and LZ modules. The X-axis is the percentage of traffic that need encryption and the Y-axis is the percentage of traffic that need compression. The percentage of traffic that need plain forwarding is omitted since is the remaining of the other two traffics. As we move towards the lower left corner of the figure, the percentage of traffic that need compression and encryption is reduced, therefore the number of processors is increased. As the traffic that belong to VPN connections requiring IPSec support is increasing, the number of DES modules is also increasing (lower right corner). Finally, when the traffic that need compression increases, the number of LZ modules also increases (upper left corner).

Figure 3.7 depicts the distribution of the modules, for the MicroBlaze in the case of the 20/20/60 distribution for several packet sizes. When the average

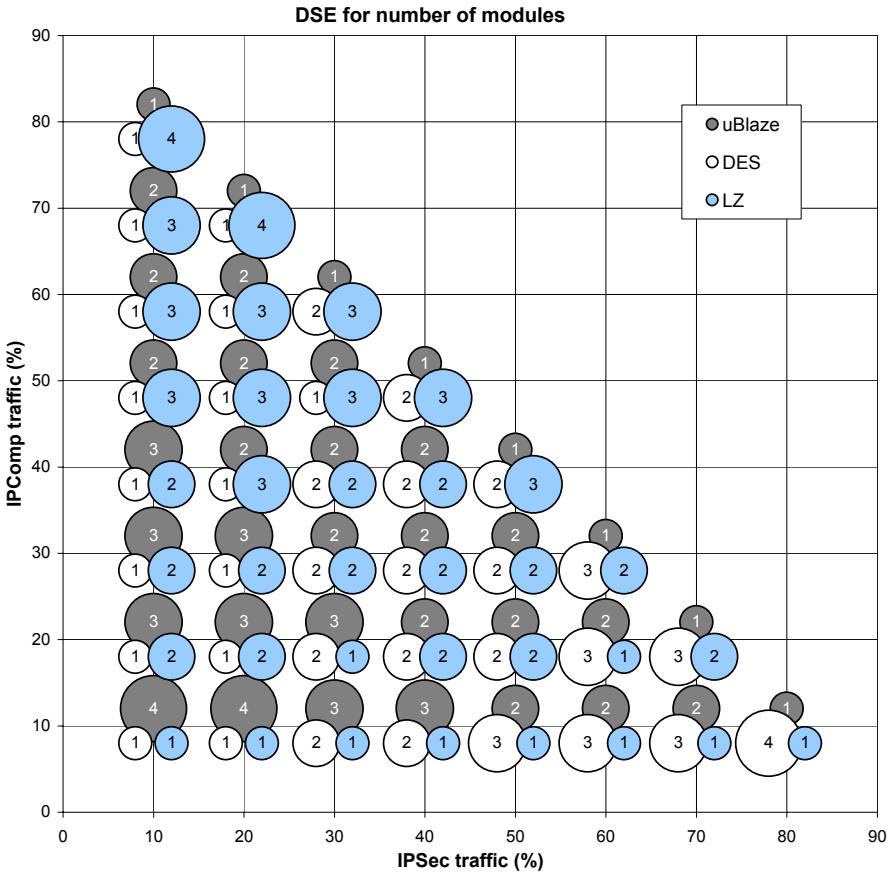

Figure 3.6: DSE of optimum configuration

packet size is small, the optimum configuration is achieved using 2 processors and 3 LZ modules. On the other hand, if the packet size is large (1024 bytes), the optimum configuration is achieved using 1 processor and 4 LZ modules since the system is constrained by the processing time of the compression.

Figure 3.8 depicts the upper theoretical processing time of 100 packets for several flow distributions for 512 bytes average packet size. This figure depicts that the use of a reconfigurable framework that adapts the number of processors and co-processors to the processing requirements of the network traffic and the average packet size can improve significantly the overall performance of the system.

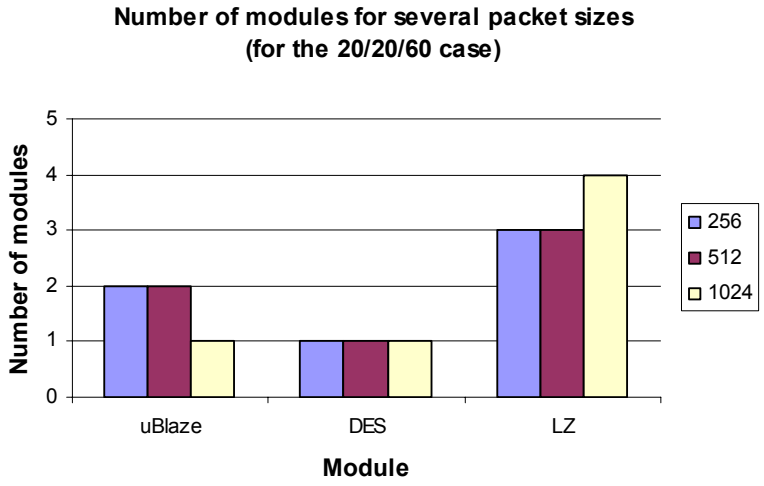

Figure 3.7: Co-processor distribution for several packet sizes

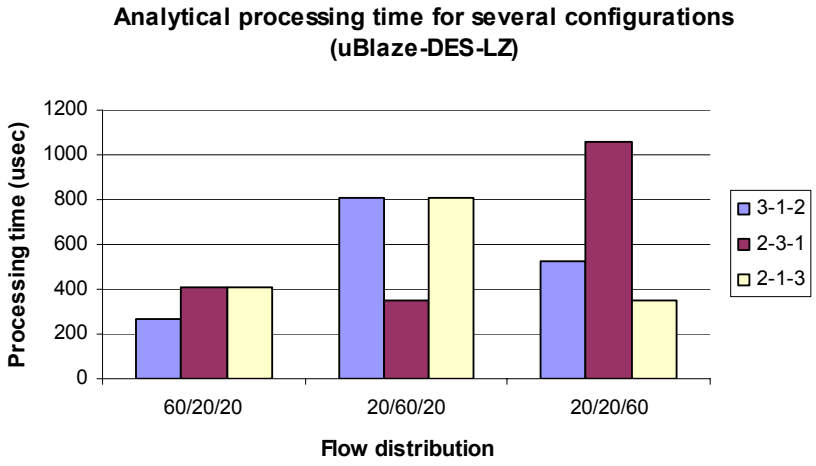

Figure 3.8: Processing time for several configurations

Although the use of a more powerful processor (e.g., PowerPC) can replace the use of two or three simple processors there are many cases that the use of simple multi-processors can provide better results is terms of performance and power consumption. As it is shown in [77], a platform using multiple cores at lower frequency and lower power supply than a simple core can provide higher throughput consuming the same power. The product of power and performance can be especially high in the case of multi-cores especially when the application is highly parallel. Network processing applications are inherently parallel applications since each processor can process a packet independently. Therefore, in network processing platforms, simple processors are used to perform the packet processing while more complex processors (e.g., PowerPC) are used for the network management (for example to process exception packets or management network packets). The same principle can also be applied in the hardware accelerators, in which a pool of simple units clocked at lower frequency can provide better results in terms of performance and power consumption than a complex unit clocked at higher frequency. For example in [78], there are several implementation of the DES algorithm in which it is shown that the use of three area-optimized units can sustain the same throughput as a performance-optimized unit consuming less than 1/4 of the power of the performance-optimized unit. After the design space exploration of the system, the system architect can select the best configuration for each workload and proceed to the implementation of the configurations.

3.4 Conclusions

In this chapter the reconfigurable network processing platform has been introduced. The basic modules of this platform have been shortly described and an integrated design flow has been presented. The integrated design flow describes the flow from the design space exploration of the optimum configuration for each network traffic pattern to the implementation and the management of the configurations. The design space exploration and the correlation of the configuration with the network traffic patterns are performed at design time. At run-time a configuration manager is used to select the configuration of the platform based on the real-time traffic.

Finally, an example is presented in which the design space exploration is applied to the design of a reconfigurable multi-service edge router. The edge router has been described as an optimization problem using non-linear integer programming equations. The constraints and the features of the platform have

also been described in equations. The description of the system in analytical equations using the characteristics (area, speed) of the implementation modules has provided a fast design space exploration of the system. The results justify the proposition that the use of different configuration based on the network traffic fluctuations can provide a better throughput than a static system.

Chapter 4

Configurable Transactional Memory

This chapter presents the configurable transactional memory controller that is used in the proposed reconfigurable network processing platform and can provide an easier programming paradigm and higher throughput compared to the traditional locking schemes. Section 4.1 presents a short introduction about the need for more efficient programming models and an introduction to the transactional memories. Section 4.2 presents the overall system architecture of the configurable transactional memory controller. Section 4.3 presents the implementation of the controller in the reconfigurable transactional memory controller. Finally, Section 4.4 presents a summary and the conclusions of this chapter.

4.1 Introduction to Transactional Memories

The increasing power consumption of the processors was one of the most important reasons that many vendors shifted to multi-processor platforms, instead of increasing the clock frequency and the sophistication of the processors. Multi-processor platforms can achieve higher throughput and lower power consumption compared to a processor synchronized to a higher frequency [77]. The main drawback of the multi-processing platforms is the programming efficiency. Programming concurrent heterogeneous platforms remains an art due to many challenges. One such challenge is the synchronization among concurrent threads or processors.

Network processor vendors were the one of the first ones that adopted the use of simpler multi-processor platforms into the same device, mainly because of the fact that network processing applications are inherently parallel. Each packet can be assigned to be processed by a specific processor independently of the other packets. Although the network processing applications are inherently parallel, a basic synchronization is required, especially for applications such as traffic metering, traffic billing and accounting, performance analysis and security management. In these applications, many counters need to be updated for each packet that is processed [79]. In the multi-processor context, the synchronization of the processors is usually based on locks that have significant performance limitations and increased programming complexity. Before accessing a shared memory region, the processor should first acquire the lock of this region. If another processor is using this region, then the processor should wait until the region is free. After acquiring the lock, the processor can access the memory and then it releases the lock. The current shared memory abstractions based on locks and mutual exclusions are difficult to use, scale, and generally result in a tedious and error-prone design process. One of the most important drawbacks of the locks is the deadlocks that can be created. A deadlock occurs when two or more processors are each waiting for another to release a resource. Another problem of using lock-based synchronization is to find the right granularity for each application. Fine-grained locks have better performance but increased programming complexity and are error-prone. On the other hand, the programming complexity can be reduced using coarse-grained locks (for example, one lock for the whole data structure) but with a major impact on the performance of the system.

A much simpler abstraction for synchronization of multi-processors is the use of transactions. Transactions are programming operations that are executed atomically as seen by the processors; therefore, they provide the consistency of the shared memory. The main idea of the transactions is that each processor can access the shared memory assuming that all processors access a different region of the shared memory. In case the processors access the same region, then a conflict is detected. In this case, one of the processors must stop and restart the execution of the transaction. The main issue is that the effect of the processor until the conflict has been detected must be canceled (rollback). Programming using transactions can provide all of the benefits of the databases transactions such as:

- Atomicity
- Consistency

- Isolation

Using transactions, the programming of a parallel platform is much simpler. There is no need to lock the variables before modifying them or to unlock after the update. Therefore, transactional schemes can be used to develop more robust code. The two main drawbacks of transactional memories are the interrupt handling and the I/O operations. However, in network processing applications the interrupt handling and the I/O operations are limited. Moreover, the tasks that are performed on each packet are quite distinct (parsing, modification, scheduling, etc.). Therefore the tasks can be easily mapped to transactions. The transactional support of a processor can be implemented either in software or in hardware.

Although lock-based synchronization schemes are most widely used method of synchronization of multi-threaded and multi-processors systems, the performance limitations and the increased programming complexity has created the need for research on hardware assisted transactional memories. The majority of the proposed schemes are implemented by modifying or upgrading standard multi-processor cache coherence protocols. One of the first implementations of transactional memories is presented in [80]. In this paper, a transactional memory is implemented by modifying the cache coherence protocol. The main idea of the proposed architecture is that the same protocol that is used for cache coherency can be used for transactional conflict management with no extra cost. The proposed architecture introduces new instructions such as load-transactional, store-transactional, commit and abort and can be implemented for both bus-based (snoopy cache) and network-based (directory) architectures. In the proposed implementation, each processor maintains two caches: a regular cache for regular operations and a transactional cache for transactional operations. In the second cache, each line contains information about the transaction such as invalid, valid, dirty or reserved.

A transactional memory with the same semantics has been proposed in [81] [82]. In these papers, a unified model is proposed in which the conventional coherence and consistency technique has been completely replaced instead of being duplicated for regular and transactional accesses. In these works, only one cache per processor is used to store the data and the transaction information. Furthermore, a write-buffer is used to store the data that have been modified. In case that a commit is requested, all of the data of the write-buffer are transferred to the main memory without being interrupted and the addresses are also broadcasted to the other processor to preserve the consistency. The main drawback of the proposed scheme is that it increases the communication

bandwidth of the multi-processor platform.

In [83] [84] [85], a new scheme is introduced called Transactional Lock Removal (TLR). The proposed scheme uses the lock semantics to identify and create the boundaries of the transactions. The main feature of the proposed scheme is that it tries to serialize the code only on conflicts while using speculative lock elision for the remaining part of the critical code. When there is a conflict it uses timestamps to eliminate livelocks and to serialize the transactions. The main advantage of the proposed scheme is that it does not require either instruction set changes or compiler changes; only the transactional hardware unit must be incorporated to the current processors to support transactional memories. The main bottleneck of the transactional memories is that they are often bounded by system constraints such as cache size. In [86], a new scheme is introduced to support unbounded transactional memories. The proposed scheme explains how to use the main memory to store the data of the transactional cache, hence to eliminate the bounding of the transactional memories by sacrificing the performance of the system.

Until now FPGAs have been used only for prototyping of the proposed transactional schemes ([87] [88]). But as FPGAs have evolved from simple logic devices to complete SoC platforms incorporating multiple processors (either hard or soft core) there is the need for efficient and robust concurrent programming abstraction without significant area overhead. In this case the programming of the platform using locks is error prone and the adoption of complex transactional schemes with cache coherency adds significant area overhead. Hence, a simple transactional memory scheme is required.

The proposed Configurable Transactional Memory Controller (CTMC) is mainly targeted at embedded applications in FPGAs and can be used on multiprocessors with or without cache. Hence, the proposed scheme does not depend on cache coherence protocols. Furthermore, the proposed scheme is targeting reconfigurable platforms hence it can be reconfigured to meet the application demands. The size of the cache and the buffers that are used for the memory consistency can be extracted by profiling or static analysis of the application code. The main feature of the proposed architecture is that it is centralized. It uses only one cache to store the shared data for all of the processors. This feature keeps the communication bandwidth of the platform low since the data of each processor does not have to be broadcasted to the other processors. The main limitation of this feature is that it is not scalable to a large number of processors. Using the current implementation the proposed scheme can be used for platforms incorporating up to 8 processors.

The remaining chapter presents a configurable transactional memory controller that is light weight and integrates in to existing tool chains and soft and hard processors easily. Furthermore, in applications such as network processing, where the number of data conflicts and lock contention is low, the proposed scheme can also be used to significantly improve the performance of the system. The main features of the proposed CTMC are:

- A configurable transactional memory controller for FPGAs that fits to the application
- Small area overhead, thus ideal for embedded applications targeting FPGAs
- No need for cache coherence protocol
- No need for changes in the ISA, or the compiler
- Major speedup over locks in applications with low presence of data conflicts (e.g. network processing)

4.2 System Architecture

The top level organization of the proposed scheme is illustrated in Figure 4.1. This figure depicts an example with 4 processors that have access to a shared memory. The processors can access the shared memory either conventionally, using the shared bus, or transactionally, using a direct interface with the transaction controller. The access to the shared memory using the bus can be used when the processor accesses the private sections of the memory. In case that the processors want to access shared portion of the memory, the transaction controller must be used. The division of the memory into private and shared regions is performed by the programmer; therefore, the programmer should use the libraries for the common bus when accessing the private region and the libraries for the transactional controller when accessing the shared regions. The transactional controller uses a small fully associative cache to store the shared data and a FIFOs (one for each processor) in which the addresses, of each shared variable that each processor has accessed, are stored. Note that we have multiple ports on the memory controller as opposed to a bus-based single port memory controller found in conventional processor based systems.

The organization of the cache is depicted in Figure 4.2. For each entry there are 8 bits that are used to store what kind of access each processor had (one

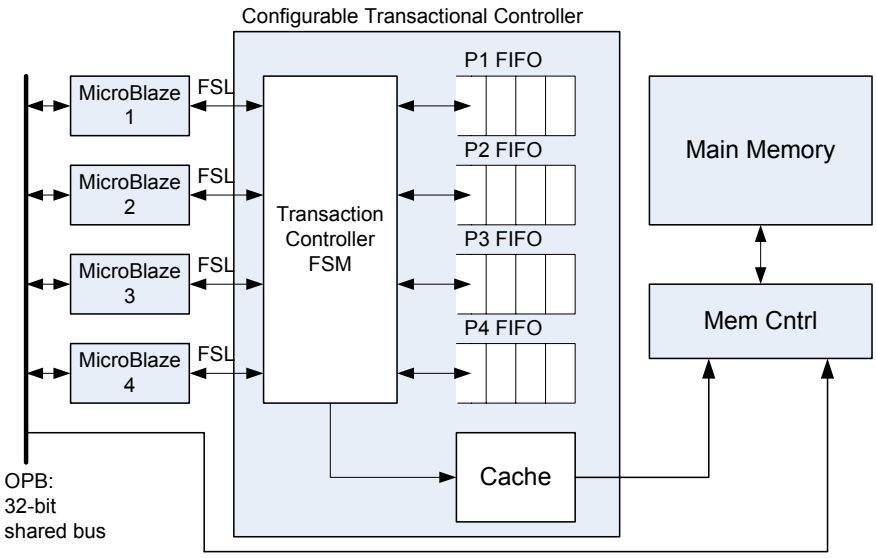

Figure 4.1: Configurable transactional controller

Read and one Write flag for each processor). The FSM of the transactional controller is shown in Figure 4.3. When a processor issues a load, the FSM checks whether the data are already in the cache. If the data is not stored in cache then the transaction controller accesses the main memory, stores the data to the cache, returns the value to the processor and asserts the corresponding Read flag in the cache. When the processor issues a store the transaction controller checks if there are any collisions. A collision occurs if any of the other processors R or W flags is already asserted. If there is no collision the data are stored to the cache and the corresponding W flag is asserted.

In case there is a collision, the controller retrieves all the accessed addresses from the corresponding FIFO and modifies the cache. If the address is used only by this processor, this entry is deleted. If the address is used also by other processors, only the flags (Read and Write flags) of the specific processors are deleted. Furthermore, a Rollback Flag is asserted for the specific processor. When the processor tries to commit the transaction, the controller notifies the processor that there was a collision and reset the Rollback Flag for the specific processor. Therefore, the processor does not have to poll or to be interrupted in case that there is a collision. The drawback of this scheme is that in case there is a collision the processor will keep executing the instructions and it will be notified only when it tries to commit. But if the number of instructions of the

Tag	Data	R1	R2	R3	R4	W1	W2	W3	W4
0x1000	0x1	✓							
0x2000	0x2		✓	✓					
0x3000	0x3	✓	✓		✓	✓			
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 4.2: Cache organization

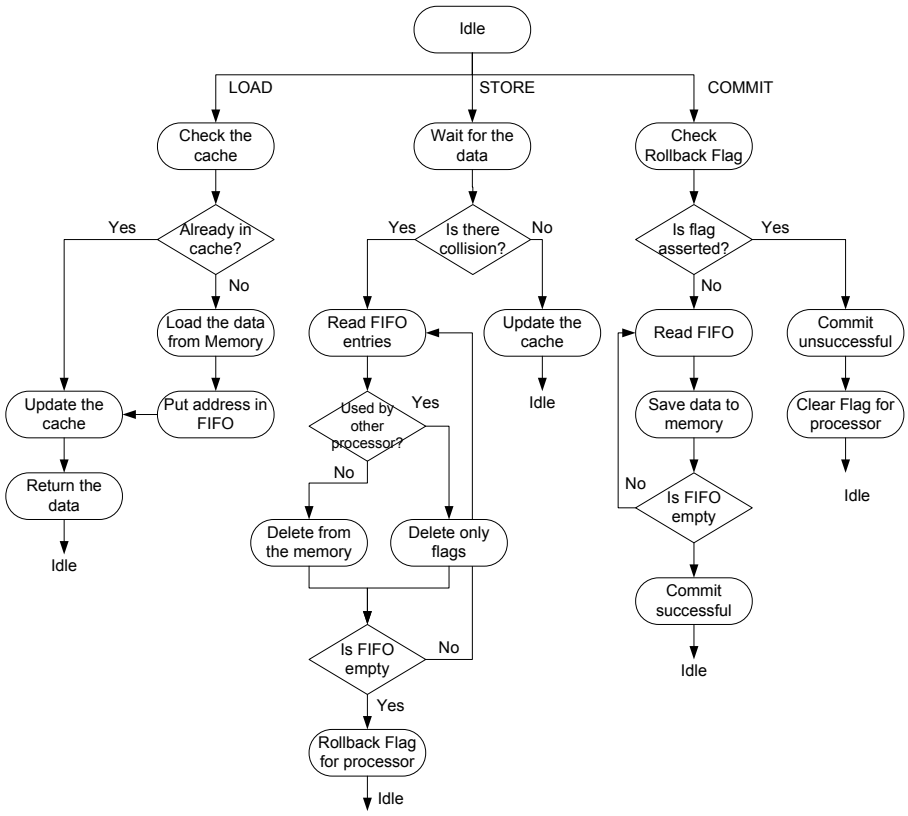

Figure 4.3: FSM of the configurable transactional controller

transaction is small (which is typical in network processing applications) the effect in the performance is very small.

When a processor issues a commit and the Rollback Flag for this processor is 0, then the transaction controller retrieves all the accessed addresses from the corresponding FIFO and copies the entries from the cache to the main memory. In case that an entry has been accessed only by this processor then the whole entry is cleared. In case that the entry has been also accessed by other processors then only the Read and Write flags of this processor are cleared and the entry remains valid.

The current scheme uses the Cuckoo hashing to store and retrieve the data from the cache. The Cuckoo hashing [89] uses two hash functions instead of only one. When a new entry is inserted then it is stored in the location of the first hash key. If the entry is occupied the old entry is moved to its second hash address and the procedure is repeated until an empty slot is found. This algorithm provides constant lookup time O(1) (lookup requires just inspection of two locations in the hash table) while the insert time depend on the cache size (O(n)). In case that the procedure enters an infinite loop the hash table is rebuild. But even in this case it can be shown that the performance of the system in the insertion is efficient as long as the number of keys is kept below half of the capacity of the hash table.

The main advantage of the proposed scheme is that the organization can be configured to meet the application requirements. Table 4.1 shows the parameters of the design that can be easily changed to meet the target application. The number of FIFOs depends on the number of the processors that are used to access the shared memory. The depth of the FIFOs depends on the application, e.g., the maximum number of shared variables in the transactions. The size of the cache depends not only on the number of processor and the number of shared variables but also on the application requirements (e.g., collision probability). In summary, CTMC is based on two basic hypothesis; first, for applications that require small number of operations per transaction (less than 10 instructions), CTMC will have a small area overhead but will have higher performance and significantly lower design effort; second, for applications that perform lot of operations in each transaction (greater than 20-25 instructions), the lock based scheme might have marginally better performance and lower area overhead but the design effort required will be significantly larger than that based on CTMC.

Figure 4.4 depicts an example source code for MicroBlaze implementation of the transactional and the lock schemes. Here the transaction begins at the first

Table 4.1: Parameters and Dependency

Parameter	Dependency
Number of FIFOs	Depends on the # of processors
Depth of FIFOs	Depends on the application (shared variables)
Cache Size	Depends on the # of processors and the application and the constraints

transactional instruction (trans_load()) and ends at the last commit operation. This approach is similar to conventional transactional begin and end semantics. However, in our approach these key words are not explicitly required. As shown, in the case of the transactional scheme, the programmer does not have to worry about issuing lock requests, but instead issues transactional loads and stores. Note the difference compared to other efforts where in there is burden on the compiler to take code that are marked with transactional sections (using transaction begin and end constructs) and generate appropriate code. In our approach we used a library based scheme to work around the requirement of modifying the MicroBlaze compiler. Similarly, for the lock-based scheme, the user now has to issue lock requests and unlock requests to particular sections of the memory. Therefore programming with transactional semantics entails less burden on the programmer, who is not required to worry about acquiring and releasing various sections of the memory, a function that is now performed by the hardware.

Furthermore, the proposed scheme can be used to provide ordering between the transactions to maintain a specific program execution similar to the scheme used in [81]. The programmer can assign an identification number (common to a group of transactions) and a sequence number to each transaction. These two numbers can be forwarded to the transactional control with the commit command request (32-bit wide). The transactional controller checks the identification and the sequence number of the transaction to ensure consistent execution. If the sequence number for a specific transaction group is higher than the current sequence number then the transaction has to rollback. Otherwise, the transaction is committed. Figure 4.5 depicts an example in which three transactions with common identification number have to be synchronized. The transaction with ID X, and Y do not have to be synchronized while the transac-

Network Metering Application (Update of two counters)

Locking:

```
while (temp = lock_req(x_addr));
while (temp = lock_req(y_addr));

x_counter = lock_load(x_addr);
y_counter = lock_load(y_addr);
x_counter++;
y_counter++;
lock_store(x_addr, x_counter);
lock_store(y_addr, y_counter);

unlock_req(x_addr);
unlock_req(y_addr);
```

Transactional:

```
do {
    x_counter = trans_load(x_addr);
    y_counter = trans_load(y_addr);
    x_counter++;
    y_counter++;
    trans_store(x_addr, x_counter);
    trans_store(y_addr, y_counter);

    temp = trans_commit();
} while (temp!=0);
```

Figure 4.4: Transactional vs. Locking programming

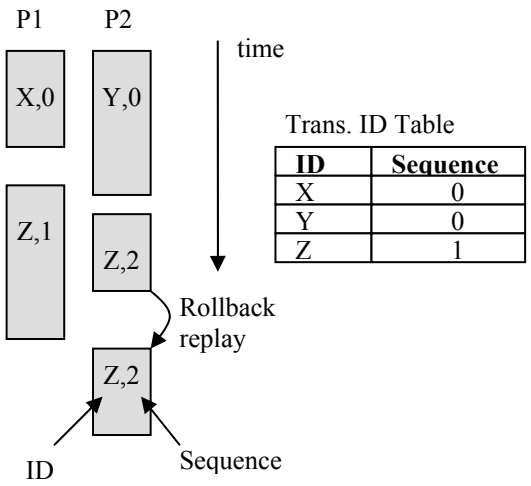

Figure 4.5: Transaction ordering

tion with ID Z has to be synchronized. A small table is used in the configurable transactional controller to check if the sequence number of the current transaction is smaller than the last one stored in the table. In that case the transaction is rolled back and is executed again as it is shown for the transaction that is executed in processor 2.

4.3 Implementation

The CTMC has been implemented in the reconfigurable network processing platform and it has been mapped into the Xilinx Virtex4 LX FPGA platform using the soft-core 32-bit Microblaze processors. Each Microblaze can be configured with up to 8 32-bit dedicated interfaces called Fast Simplex Link (FSL). This interface has been used for the communication of the processor with the transaction controller. Each time a processor issues a transactional load, it sends the address to the FSL link and waits until it receives the value of the data. When the processor issues a store, the address and the data are sent to the FSL link. Furthermore, the FSL interface uses a dedicated signal for control information. When a processor tries to commit a transaction this control bit is asserted. The transaction controller uses the control signals to convey either acceptance or rejection of a particular transaction.

The key advantage of the above scheme is that at design time, based on profil-

ing the code and static analysis the user can configure a particular transactional memory configuration. In our experiments this process was done manually. However this can be easily automated and is part of our ongoing research. In this study, we used a 64 entries cache, and the FIFOs are 32-bit wide with 16 entries deep. The main memory is 32x16K internal Block RAM (BRAM). Note that for larger memory size requirements we can use an external memory by connecting the transaction controller to a DRAM memory controller. In such a scenario, a multi-port memory controller becomes even more critical to provide efficient data path from the memory controller to the processor sub-system.

In order to compare the efficiency of the proposed transactional memory with the conventional locking scheme, a fine-grain locking memory controller has also been developed. The locking memory controller uses a similar cache organization as the one used in the transactional controller. The cache has been modified and, only one bit is used for each processor to indicate which processor has locked the address. Each entry of the cache uses a lock bit for each processor (instead of using Read and Write bits). Thus, the processor has first to lock the address before it access the data. The data are read from the memory and the lock bit of the specific processor is asserted. After the update of the data, the processor must issue a store and an unlock command. In case that the processor tries to access a region that is locked, the processor is blocked until the region is unlocked by the processor that locked the memory. Although the lock controller consumes less area, the programmability of this scheme is much harder and error prone than using the transactions. The programmer has to perform detailed analysis of their implementation to avoid deadlocks.

The area and the maximum clock frequency of the transactional controller and the lock controller for several processors are shown in Table 4.2 and Table 4.3 respectively. As it is shown the current implementation of the transactional controller can support up to 8 processors using the same clock frequency as the processors and the bus (100MHz). The area overhead is quite small (less than 530 slices for 8 processors, which requires a total area of about 8000 slices). This is clearly an important benefit of the configurability of our scheme that tailors the transactional controller to the application needs. The number of occupied internal Block RAMs (BRAMs) that are used for the FIFOs and the cache depends on the number of processors and the application. The depth of the FIFOs and the cache is proportional to the number of shared variables among the processors. These are parameters that can be configured or customized for each application. As it is shown the transactional controller is larger than the lock controller and the maximum clock frequency is less than

Table 4.2: Area comparison (in slices)

Processors	Transactional	Lock
2	277	137
4	370	247
8	530	353

Table 4.3: Clock Frequency comparison (in MHz)

Processors	Transactional	Lock
2	151	182
4	137	156
8	107	149

the lock controller. But the reduced program execution (as it will be shown in Chapter 6) and the easy of programming that the transactional memory offers compensates for the area overhead. Besides, the main problem in the current development of multi-processor platforms is the programming model for the efficient use of the available resources, while the area is usually abundant. Furthermore, the frequency that is reported in Table 4.3 has been retrieved from the Xilinx EDA tools without any optimization, since the main constraint was to support the clock frequency of the processor (100MHz).

4.4 Conclusions

The ability to configure a transactional memory to the needs of a particular application is an important aspect that is explored in this paper. The proposed configurable transactional memory controller helps in increasing the programming efficiency of a multi-processor system on FPGA. In addition, for some applications it can also help in increasing the system performance, where the synchronization among processes is the performance bottleneck.

Finally, the proposed scheme could be extended to support not only processors but also hardware accelerators modules. Hence, in this case the configurable transactional controller could be used to provide consistency between the processors and the hardware acceleration modules as it is depicted in Figure 4.6.

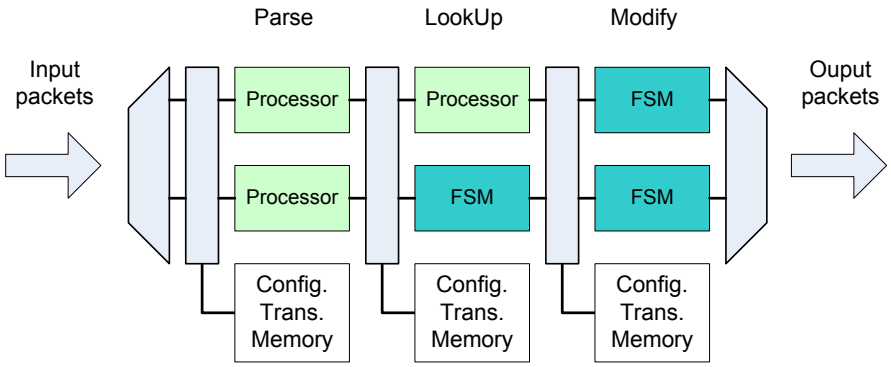

Figure 4.6: Configurable transactional memories in platforms

This figure shows a multi-processor platform in which the processing of the packets is performed in several stages (e.g., parse, lookup, modify) that include both processors and hardware acceleration modules and consistency needs to be preserved. The CTMC can be used to provide synchronization between these modules in each pipeline stage.

Chapter 5

Hardware Acceleration Modules

Several typical network applications for access, edge and enterprise networks were presented in Chapter 2. Chapter 3 presented the reconfigurable network processing platform and described shortly the major modules that are used. Furthermore, it presented the integrated flow to design the platforms. The design space exploration has indicated that the use of reconfigurable platforms can be used to accelerate network processing applications.

This chapter presents the design and implementation of several hardware acceleration modules of the reconfigurable network processing platform that are used to evaluate the platform in some typical network applications. Section 5.1 presents the design of two modules that are used to accelerate the content-based switching applications. Section 5.2 presents the proposed reconfigurable queue scheduler that is reconfigured based on the number of active queues. Section 5.3 presents the hardware accelerators that are used for the performance evaluation of the platform in edge and access routers. Finally, Section 5.4 presents the integrated implementation of the reconfigurable network processing platform in FPGAs. Section 5.7 presents the conclusions and the summary of this chapter.

5.1 Content-based Switching Accelerators

Section 2.3.2 presented a short introduction into the content-based switches. As it was described, content-based switches (also called web switches) are

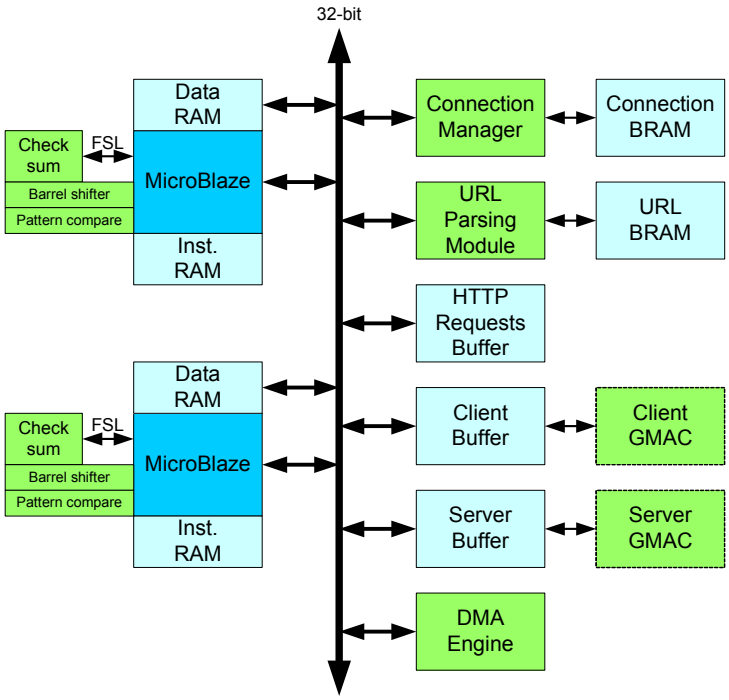

Figure 5.1: System Architecture

used in server clusters to balance the network traffic based on the application's layer header and the payload of the packets. To evaluate the efficiency of the proposed reconfigurable network processing platform, a web switch has been implemented in the platform. The main goal of the proposed scheme is to provide the flexibility of the network processors combined with the increased performance of an ASIC keeping the power consumption low. Hence, a careful software-hardware partitioning has been performed. The organization of the web switch targeting the reconfigurable network processing platform is depicted in Figure 5.1. The system consists of two 32-bits RISC soft-core MicroBlaze processors (each one with a 16KB separate instruction and data memory) and two accelerators to speedup the performance of the system.

The processors, the co-processors and the memory units are all connected using a 32-bit common bus (OPB) which supports burst transactions. Two Dual-Port Block RAM modules (16KB BRAMs) have been used for the buffering of the client's and server's packets attached to the OPB bus. The other port of the memories is connected to the Xilinx's Gigabit Media Access Controller

(GMAC). The first GMAC is connected to the client link while the second one is connected to the server farm. Furthermore, a 32KB BRAM is used to store the HTTP requests.

The most time-consuming functions of the web switching application are the management of the spliced connections and the URL parsing. To accelerate the performance of the system, these two functions have been implemented in hardware and are connected to the processors using the shared OPB bus. The Connection Manager Module is used to control the spliced connection table (the RAM in which the connections are stored). The URL Parsing Module is used as a search engine. The input is the URL string and the output is the corresponding IP address of the server. Every block RAM (BRAM) is on-chip (internal) for faster access but the system can be easily extended to use external RAM to support more connections, causing some reduction in performance.

The processors that have been used are the Xilinx's MicroBlaze processors. These processors have been customized to include a barrel shifter and a pattern compare unit. The barrel shifter instructions takes 2 cycles while the pattern compare instructions (such as pattern compare equal) takes 1 cycle. The use of these additional units provides improved performance since the majority of the network processing application use bit-wise instructions, while the area overhead is minimal. Furthermore, the processors have been extended using a checksum module for the packet's header processing. The checksum module is attached to the processor using a point-to-point interface called FSL (Fast Simplex Link) that provide a fast interface between the processor and the co-processor.

The application that is hosted in the processors is a simplified version of the TCP splicing used in Linux servers that supports the most important features of the TCP splicing. The simplified version of this application has the same characteristics as in [39] for the TCP splicing targeting the Intel network processor, as it is shown in Table 5.1 and Table 5.2. Each processor polls the Client and the Server buffer for new packets. If there is a new packet, only the headers of this packet (up to the HTTP header) are forwarded to the processor. The FSM of the content-based switch is depicted in Figure 5.2. The processor firstly verifies the IP header using the checksum module. If a packet arrives with the SYN flag on, the processor sends a command to the Connection Module to add the new connection. After the processing of the packet, if there is a new packet created, then the packet is forwarded to the corresponding buffer (client or server). When a new HTTP request arrives, the processor stores the packet in the HTTP Request BRAM. After a connection has been established

Table 5.1: Processing of SYN packets

Step	Task	Supported
1	De-queue packet	Y
2	IP header verification	Y
3	IP option processing	N
4	TCP header verification	Y
5	TCP option processing	N
6	Send ACK packets	Y
7	Update state	Y

Table 5.2: Processing of SYN/ACK packets

Step	Task	Supported
1-5	Same as above	Y
6	Verify ACK number and flags	Y
7	Initialize receive sequence number	Y
8	Update state	Y
9	Send ACK packet	Y

between the content switch and the server, the HTTP Request packet is retrieved from this BRAM and is forwarded to the server buffer. Moreover, after the connection has been established, the packets are forwarded from the client buffer to the server buffer and vice versa using DMA burst transfers. Hence, in this case the processors are used only to check the established connections and change the packet's header (such as the IP, the TCP Sequence number and the header's checksum).

A list of free indexes is stored also in the shared memory that stores the HTTP request packets. Hence, every time a processor wants to store a request, firstly it requests an available index from the free list table and then stores the request to the provided index. Since the indexes are stored in the shared memory, the transactional memory controller has been used to preserve the consistency between the two processors. However, this task is only used once per connection (just to store the client's HTTP request packet) therefore the performance amelioration compared to a locking scheme is negligible.

Moreover, in order to hide the communication overhead between the processor

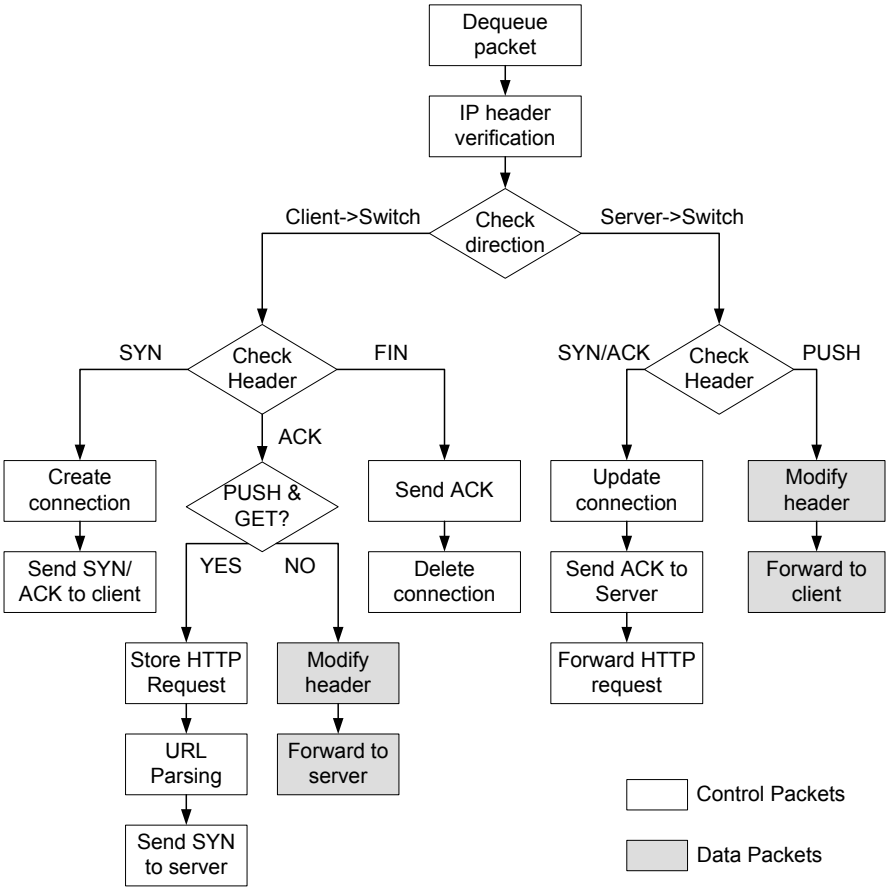

Figure 5.2: The FSM of the content-based switch

and the buffers, the following scheme is used. The Data BRAM is a dual port RAM, in which one port is connected directly to the processor while the other port in connected to the OPB bus using a bus controller, which supports DMA burst transfers. Hence, while a packet is processed by the MicroBlaze the next packet is loaded to a reserved space in the Data BRAM using DMA transfers. Thus, the communication overhead is hidden by the packet processing.

5.1.1 Connection Management Module

The Connection Table that is used to store the connection's information is organized as it is depicted in Figure 5.3. Each entry consists of the Client's IP address, the Client's TCP port, the Server's IP (SIP) in which the connection has been spliced, the state of the connection (idle, connected, spliced, etc.) and the initial TCP sequence number (SEQ field) of the client and the server. The Server's IP is 8 bits (the last 8-bits of the 32-bit IP address, since the first 24 bits of the IP address are usually the same for a server cluster), thus it can support a cluster server of up to 256 servers, which is adequate in most of the cases. The current implementation can support up to 4K connections while it can be easily extended to support more connections. The Connection Management Module (CMM) supports three commands:

- **Write** (IP, TCP, SIP, state, Client SEQ, Server SEQ) return 0; Write a new entry to the table using all the information.
- **Search** (IP, TCP) return SIP, state, Client SEQ, Server SEQ; Search the entry using the IP and the TCP number of the client and return the Server's IP, the state and the sequence number
- **Delete** (IP, TCP) return 0; Delete the entry using the IP and the TCP information.

The organization of the CMM is illustrated in Figure 5.4. A hash unit is used to create a 10-bit index out of the 32-bits IP and the 16-bits TCP input. When a new entry command is sent, the Connection Management Module tries to store the entry into one of the Block RAMs (BRAMs). In case that all of the BRAMs

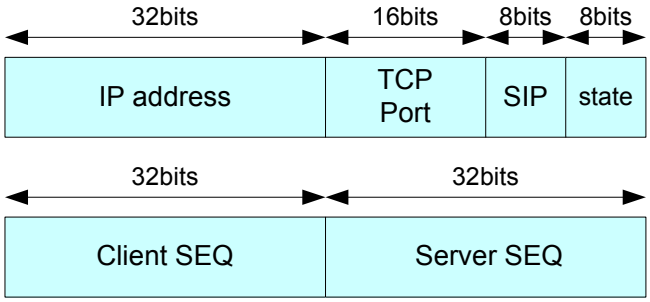

Figure 5.3: The connection entries

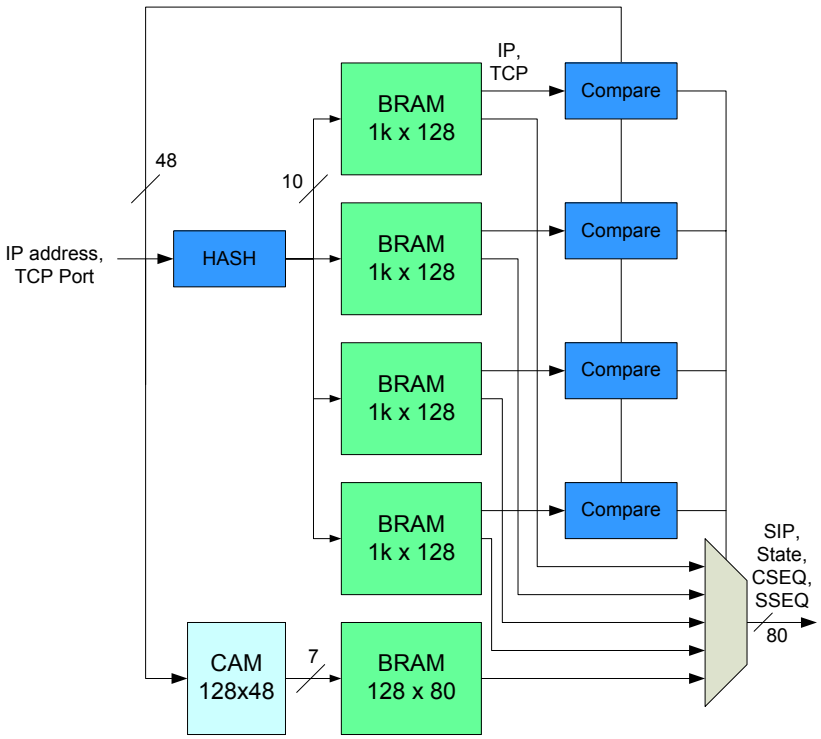

Figure 5.4: The Connection Management Unit

are occupied (collision) a small CAM (Content Addressable Memory) is used. The CAM stores the IP and the TCP field (48-bits) and outputs an index to a RAM in which the remaining data are stored (Server's IP, Connection's state, etc.). When a search is performed, the CMM searches each of the BRAMs and if there is a match in any of them it outputs the result. In case that the connection data has been stored to the CAM then the multiplexer selects the CAM output. The CAM that we used takes 1 clock cycle for read and 16 clock cycles to write [90] . As is it shown in [91], using four block RAMs we can reduce significantly the probability of collision. This scheme can provide a result of a search only in 4 cycles.

To measure the efficiency of the Connection Management Module we used the UC Berkeley Home IP Web Traces [92]. After extracting the IP address and the Port number of the clients, we measured the number of BRAM collision for 4096 connections using a simple XOR hash function. The number of collisions (that hash to the same address over four times) was 295. The CAM

Table 5.3: Collision Probability

Number of BRAMs	128CAM	256CAM
2	18%	15%
3	11%	8%
4	4%	1%
5	0%	0%

can accommodate 128 entries, hence the probability of a connection collision (in which the connection can not be stored neither at the BRAM nor at the CAM) is 4%. Table 5.3 shows the probability of a connection collision for several numbers of BRAMs and for 128 and 256 entries CAM. This scheme can be easily extended using larger FPGA device. In case that we use four 16Kx128 BRAMs and a 512 entries CAM then the probability of connection collision is 5% (there were 1383 BRAM collisions) and is able to support 64K connections.

5.1.2 URL Parsing Module

One of the most computationally demanding functions, is the web switching based on the URL string. For example content distribution can be performed based on the URL (e.g. the www.foo.edu/publications requests could be distributed to a separate server than those to www.foo.edu/people). Another option is the switching based on the application. For example the HTTP request for an image could be forwarded to a separate server than the server for the HTML or PHP pages. Hence, an efficient URL parsing module is necessary to perform fast web switching based on the URL. In our implementation an efficient form of a compressed trie has been used. A trie is a tree for storing strings in which there is one node for every common prefix. A compressed trie is a trie in which non-branching subtrees leading to leaf nodes are cut off [93] [94]. Three separate block RAMs are incorporated; a 256x32 bits Char RAM, a 256x32 bits Address RAM and a 256x12 bits Control RAM as it is illustrated in Figure 5.5.

- The Char Table stores the characters of the URL string
- The Address Table stores the Server's IP and an index to the Char Table

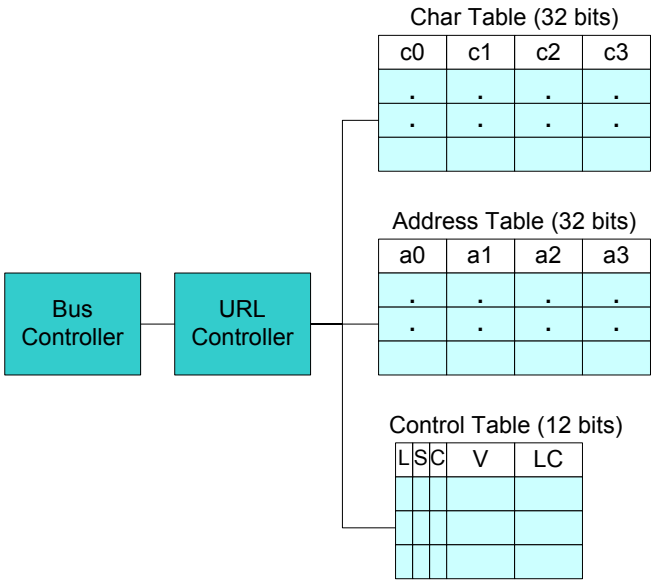

Figure 5.5: The URL module

Table 5.4: Directory-based URL Table

URL	Server IP
/pub/*	16
/pub/phd/*	17
/pub/msc/*	18
/people/*	19

- The Control Table stores several control bits

After a search has been performed, the Server's IP (the last 8-bits) is stored to the Connection Table (Figure 5.3). To explain the function of the URL module an indicative example is used, as it is shown in Table 5.4, for URL directory-based switching. This table shows the URL and the last byte of the IP address (the first three bytes are the same for a server cluster) of the corresponding server. As it is shown, we need a co-processor that will be able to perform longer prefix matching of the URL string to the URL table.

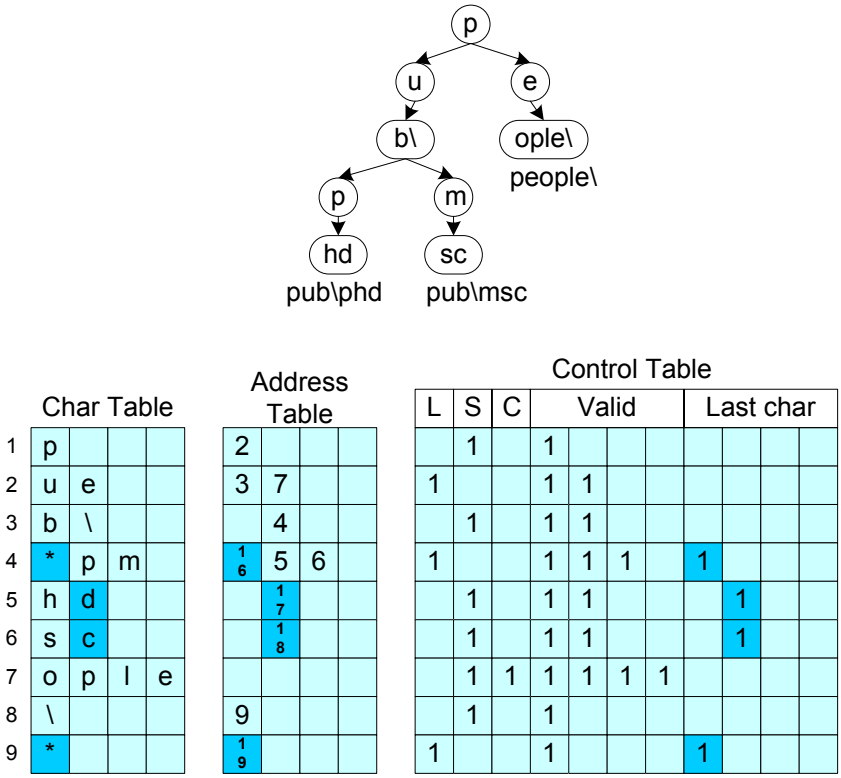

Figure 5.6: The URL graph and the URL Tables

The characters are stored in the Char Table using the following algorithm. As long as the char are unique compared to the other characters in the same position, the characters are stored in the Char Table sequentially. When there are different characters in the same position (e.g. p**u**b and p**e**ople) these characters are stored in a separate entry (e.g. address 2, Figure 5.6) with an index to the Address Table for the next characters (e.g. address 3 and address 7 for *pub* and *people*, respectively). The L and the S columns of the Control Table are used to indicate if the corresponding entry has characters that split to new sub-string or belong to the same string, respectively. For example, entry #7 stores characters that belong to the same string (hence the corresponding S entry is '1') while entry #4 stores characters that belong to different sub-strings (misc, "phd", and "msc"; hence the L entry is '1'). The C column in the Control Table shows if the string is continued to the next sequentially entry. For example, the "people" string is continued to entry #8.

The Control Table holds two more 4-bit columns. The first column represent if a char is valid in the Char Table, while the second column (Last Char) represent if this is the last characters, hence there is a valid Server's IP address in the Address Table. For example, in the entries in which the Last Char flag is '1' (the shadow boxes), there is the Server's IP address as it shown in the corresponding URL Table (Table 5.4).

The URL search algorithm which has been implemented in the URL controller is depicted in Figure 5.7. The controller first checks the same level and the same string flags to identify if the row that is processing holds sequential characters or characters that point to new sub string. In the case that the characters are at the same level (used to point new sub-strings) each character is compared against the current URL char. If it is the same then the next address is retrieved from the Address Table. In the case that the same-string flag is on (second half of the algorithm) the module compares the valid characters of this row with the corresponding URL string.

Using this scheme we can achieve a throughput of 1-4 matched characters per cycle. In order to compute the number of characters that should be stored in each entry of a LookUp table such as the one in Figure 5.6 we used the HTTP request traces from the San Diego Super Computer Center [95]. According to these traces the maximum number of characters that are used to describe the first two sub-string of a URL string (e.g. \pub\phd) is 27 characters. Taking into account, that the majority of the web-switching is performed using the first two sub-string we set 28 characters the maximum number of characters that can be searched by the URL module. Thus, the maximum number of cycles to output a valid IP address is 28 cycles. The number of entries is expandable and can accommodate a large number of URL strings. In the current implementation we used a 1024 entries table; hence the maximum number of stored characters is 4096. However, in cases that the URL entries have higher demands, the URL module can be used as a cache in which the most current entries are stored, while the remaining entries are used by a software function in a separate memory. In addition, the most important feature of this accelerator is that in order to maintain and to update the URL table there is no need for a heavyweight reconfiguration. The URL table is stored in BRAM that can be updated using any external interface (such as UART) and exploiting the second spare port of the inherent Dual Port BRAMs.

Furthermore, the URL co-processor can be used for URL application-based switching. In this case, the server is selected based on the extension of the requested file, as it is presented in Table 5.5. In this example, the requested

```
% Same Level
if Same_level='1' then
  if char(0)=url(j) and valid='1'
    if Last_Char='1'
      return ServerIP
    else
       address = url(address);
       j = j + 1;
    end if;
  elseif char(1)=url(j) and valid='1'
    …
  elseif char(2)=url(j) and valid='1'
    …
  elseif char(3)=url(j) and valid='1'
    …
  else
    if url_continued='1'
      address = address + 1;
    end if;
```

```
% Same String
elseif Same_string='1'
  if c(0)=url(j) & c(1)=url(j+1) &
     c(2)=url(j+2) & c(3)=url(j+3)
    if url_continued='1'
      address = address + 1;
      j = j + 4;
    else if Last_Char='1'
      return ServerIP;
    else
      address = url(address);
      j = j + 4;
    end if;
  elsif c(0)=url(j) & c(1)=url(j+1) &
        c(2)=url(j+2)
    …
  elsif c(0)=url(j) & c(1)=url(j+1)
    …
  elsif c(0)=url(j)
    …
  end if;
end if;
```

Figure 5.7: The URL Search Algorithm

Table 5.5: Application-based URL Table

URL	Server IP
.gif	16
.mpg	17
.htm	18
.cgi	19

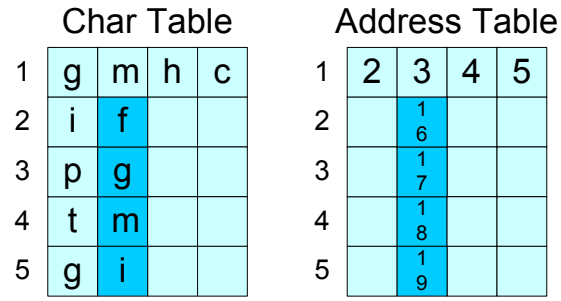

Figure 5.8: The application-based Tables

images (.gif), the videos (.mpg), the HMTL and the CGI files are all forwarded to a separate server. In this case, the processor does not send the first 28 characters. The processor reads the URL characters until a space is found, which means that the URL string is finished. Hence, the last three characters are sent to the URL module for parsing. The corresponding tables for the URL module are shown in Figure 5.8. The first letter of each application extension is shown in the first row, while the remaining characters of each row are shown in row 2-5. Using this scheme, we can find the corresponding server in N+2 cycles where N is the length of the URL string divided by 4 (the cycles to find the last 4 characters).

5.2 Queue Scheduler

Every network processing platform needs a queue scheduler to provide priority-based scheduling of the packets. The proposed reconfigurable network processing platform includes a scheduler that is configured based on the number of active queues. The block diagram of the system is depicted in Figure 5.9.

The system consists of a Queue Table Controller, a Queue Table that holds the queue information and a Queue Scheduler that can be partially reconfigured either as a DWRR scheduler or as a WF2Q+ scheduler. Furthermore, a FIFO is used that contains the QueueID of the active queues and it is used by the DWRR scheme. The scheduler can be configured either as a DWRR scheduler or as a WF2Q+ scheduler depending on the number of active queues.

The Queue Table Controller receives the information from a Queue engine such as the Queue Manager Reference Design form Xilinx [96]. This Queue Manager can support up to 64K queues and eight traffic classes. The current implementation of the dynamic queue scheduler is targeting mainly network edge and access routers and not high-performance core routers (that usually support thousands of queues) and it is designed to explore the efficiency of dynamic reconfiguration in the schedulers, hence it can support up to 2K queues (the QueueID is 11 bits). The Queue Table (a 2-port 2Kx64bits RAM) holds the information for each queue: the size of the packet in the head of the queue (16 bits), the number of packets there are in each queue (16 bits), the DWRR weight (16 bits), and the WF2Q+ weight (16 bits). When the queue scheduler selects a queue, the QueueID is sent to the Queue Table Controller which updates the Queue Table. In the case of the DWRR the scheduler reads the next active queue from the FIFO. If the queue is eligible then the Table Controller checks if it is the last one. Otherwise, it writes the queue back to the FIFO of the active queues.

5.2.1 Deficit Weighted Round Robin Scheduling

The Deficit Weighted Round Robin (DWRR) was proposed by M. Shreedhar and G. Vargeshe in 1995 [97] and it was initially designed to address the limitations of the Weighted Round Robin and the Weighted Fair Queuing models. In DWRR queuing, each queue is configured with three parameters:

- A weight that defines the percentage of the output port bandwidth allocated to the queue
- A Quantum Size that is proportional to the weight of the queue and specify the number of bytes that will be added to the deficit counter in each turn
- A Deficit Counter that specifies the total number of bytes that the queue is permitted to transmit (available credits) each time that it is visited by the scheduler

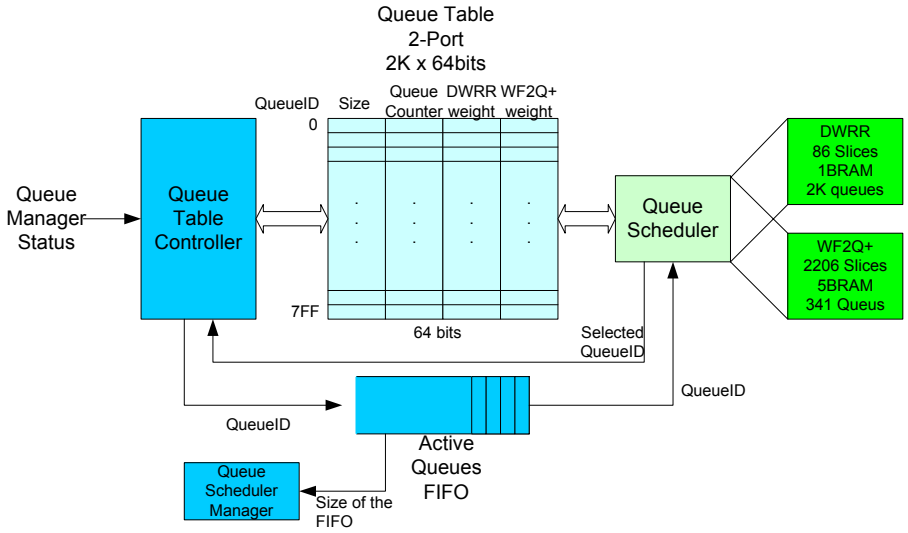

Figure 5.9: Block diagram of the queue scheduler

A Round Robin Pointer is checking every queue that is not empty. Each time a queue is visited by the scheduler, the deficit counter of the corresponding queue is accumulated with the quantum size of the specific queue. If the size of the first packet in the queue is less than the deficit counter (available credits) for that queue then the packet is transmitted and the packet size is subtracted from the deficit counter. If the size of the packet is more than the deficit counter then the packet is not transmitted and the Round Robin Pointer is shifted to the next active queue.

Figure 5.10 depicts an example with 3 queues using the DWRR scheduler. The first queue can consume half of the bandwidth (50%, Quantum Size: 500bytes) while the other two queues consume the other half of the bandwidth (25%, Quantum Size: 250 bytes). In the first and the second queue, the size of the first packet is less than the available credits (Deficit Counter), therefore the packets are transmitted (eligible queue). In the third queue, the packet size is more than the available credits. Thus, the packet will not be transmitted, the available credits will be increased by 250 bytes (the quantum size) and the packet will be transmitted in the next iteration.

The implementation of this scheduler is straight forward. It only needs an adder and a comparator and a table that holds the Quantum Size and the Deficit Counter for each queue (in our case a 2Kx16bits RAM). The drawback of this

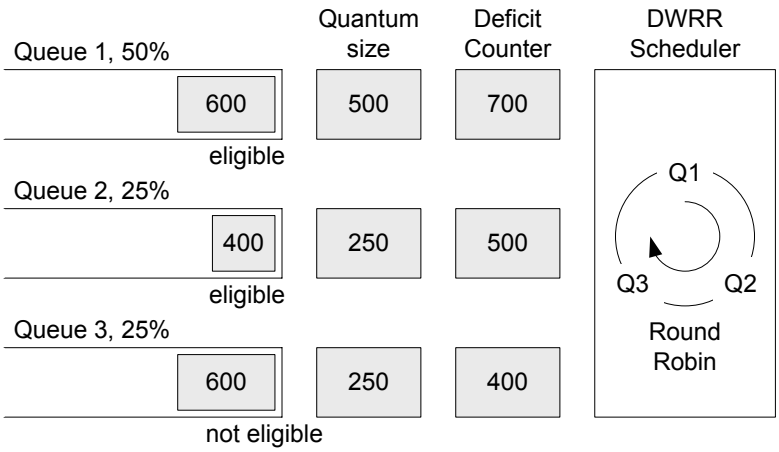

Figure 5.10: The DWRR Scheduler

algorithm is that many queues can be visited that are not eligible to be transmitted if the counter is less than the size of the packet. Hence, the time it takes for the scheduler to decide which queue will be transmitted is unknown when the weight of the queue is smaller than the size of the packets. To overcome, this drawback, a number of queues can be examined at the same time and then use a priority encoder to decide which queue will be transmitted.

5.2.2 Worst-case Fair Weighted Fair Queuing+ Scheduling

The Worst-case Fair Weighted Fair Queuing was proposed by J. Bennett and H Zhang in 1996 [46]. The weighted Fair Scheduling algorithm tries to allocate services more accurately than the other algorithms and it is closer to the ideal case of Generalized Processor Sharing (GPS). In the case of the GPS all the queues are sliced into bits and each queue is allocated exactly the corresponding bandwidth. Since, the slicing of packets into bits can not been implemented efficiently, the WFQ algorithm assigns to each packet a finish time equal to the virtual time that the packet should be finished transmitted if the algorithm was the GPS. In this case the finish time of each packet arriving at the Queue Manager should be stored. Instead, in the Worst-case Fair Weighted Fair Queuing+ algorithm, the scheduler assigns a start time and a finish time only to the packet that it is head of the queue. Hence, for each queue the following parameters are used:

- The weight of the queue
- The start time of the head packet of the queue
- The finish time of the head packet of the queue
- The virtual time

The WF2Q+ algorithm works in the following way. Each time a new packet is assigned as a head packet of a queue, the start time of this packet is configured to be either the current virtual time if the scheduler is idle or the time at which the transmission of the current packet will finish. The finish time of the packet is assigned as the start time plus the time to send this packet using the weight of this queue, as it is shown in the following equations (Equation 5.1, Equation 5.2).

$$queue(i).start\_t = \begin{cases} virtual\_t, scheduler = idle \\ finish\_t, scheduler \neq idle \end{cases} \tag{5.1}$$

$$queue(i).finish\_t = queue(i).start\_t + \frac{size(i)}{queue.weight} \tag{5.2}$$

$$virtual\_time(t) = \max\{virtual\_time(t+\tau), min\{start\_t\}\} \tag{5.3}$$

The scheduler selects the queue with the minimum finish time and then updates the virtual time using either the minimum start time of the queues or the current virtual time plus the time to transmit the selected packet as it is shown in Equation 5.3. The main advantage of this scheme is that the system operated without a priori knowledge of the traffic or assumptions about the packet size. The main disadvantage is the large number of required computations and the state information that should be stored in a memory [6]. The algorithm requires finding the minimum start and finishing time for the active queues. If the number of active queues is large then the minimum functions can slow down the performance of the scheduler if we examine each queue separately (O(N)). On the other hand, if we use a tree structure, we would waste too much hardware area. Hence, we use the heap sort scheme which is a more efficient implementation to find the minimum start and finish time from a number of active queues.

The heap sort scheme that is has been designed can compare 4 different values at the same time. To map efficiently the heap sort algorithm in the current

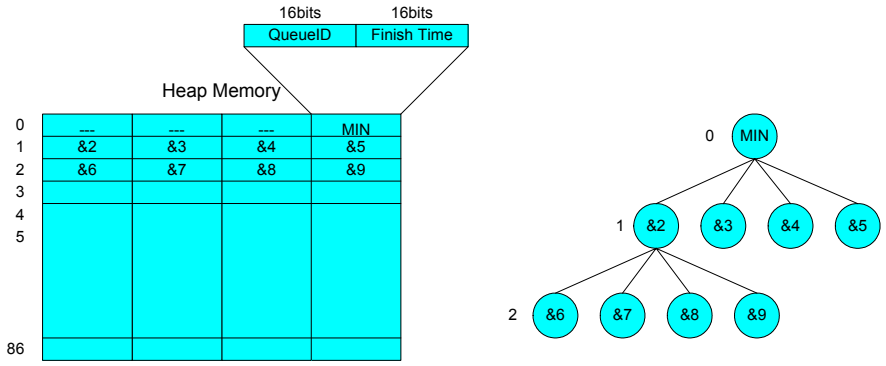

Figure 5.11: Heap Organization

design, the following scheme was used. A 256x128bits RAM is used to store 4 words as it is depicted in Figure 5.11. Each word contains the QueueID (16 bits) and the finish time (16 bits). The minimum finish time is stored at address 0x0. At address 0x1 is stored the next level of the heap sort (4 values), at address x2 is stored the 3rd level of the most significant word from the 2nd level, etc. Using this scheme we need 1 cycle to find the queue with the minimum finish time and 5 clock cycles to update the heap sort table.

Each time a new entry has to be inserted into the heap size the following scheme is used as depicted in Figure 5.12. The new finish time is inserted into the level 16-bits register and the memory data from the same level is loaded. If any of the four entries is not valid (V bit in the scheme) then this space is selected to store the new data. Otherwise, it compares the value of the new entry with the other 4 finish times and if any of them is smaller then the new entry is inserted into this space and the previous one is forwarded to the next-level register. If the new entry is bigger than all of the values in the memory then the new entry is forwarded to the next-level register. The current design can support up to 341 queues (1 + 4 + 16 + 64 + 256 = 341 elements). As it was mentioned before, the scheduler selects the queue with the minimum finish time but it is also use the minimum start time to update the virtual time. Hence, a similar heap sort has been used to sort the start time of the queues. But in this case, the minimum start time does not have to be extracted from the table. The scheduler only reads the minimum start time but it does not extract it from the table. Each time a queue is selected from the Finish Time Table, the corresponding Start time entry must be deleted from the Start time table. Hence, a small Look-Up Table is used to map the QueueID to the address of

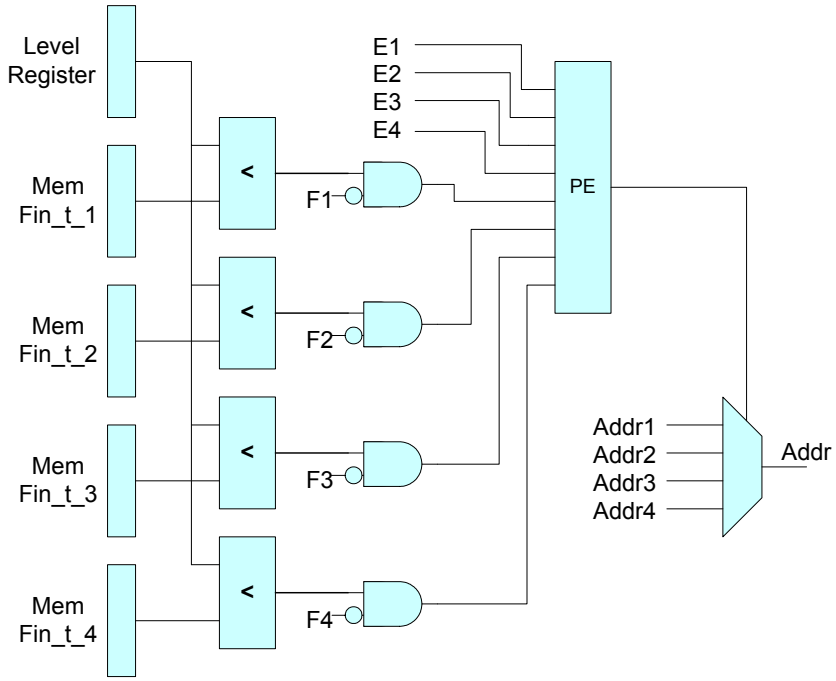

Figure 5.12: The WF2Q+ algorithm

the Start time Table as it is shown in Figure 5.13. Thus, the time to update the Start Time Table takes 1 to 5 clock cycles.

5.2.3 Queue Scheduler Implementation

The current design has been implemented into a Xilinx Virtex-4 FPGA. The design methodology for partially dynamically reconfigurable designs was used to floorplan the system [98]. The system was portioned into static and reconfigurable areas. The signals that are used for the communication of the static and the reconfigurable area have to be placed using specific tri-state buffers (bus macros). The signals that are used in this design are the signals for the second port of the Queue Table, the signals to read the active queue FIFO and the selected QueueID (12 bits). Table 5.6 illustrates the resources that each unit occupies. As it is shown in the table, the WF2Q+ scheduler occupies many more slices than the DWRR scheduler. Therefore, the DWRR is used as the standard scheduler that is always placed in the network processing platform. On the other hand, the WF2Q+ scheduler is used as a page-able module. When

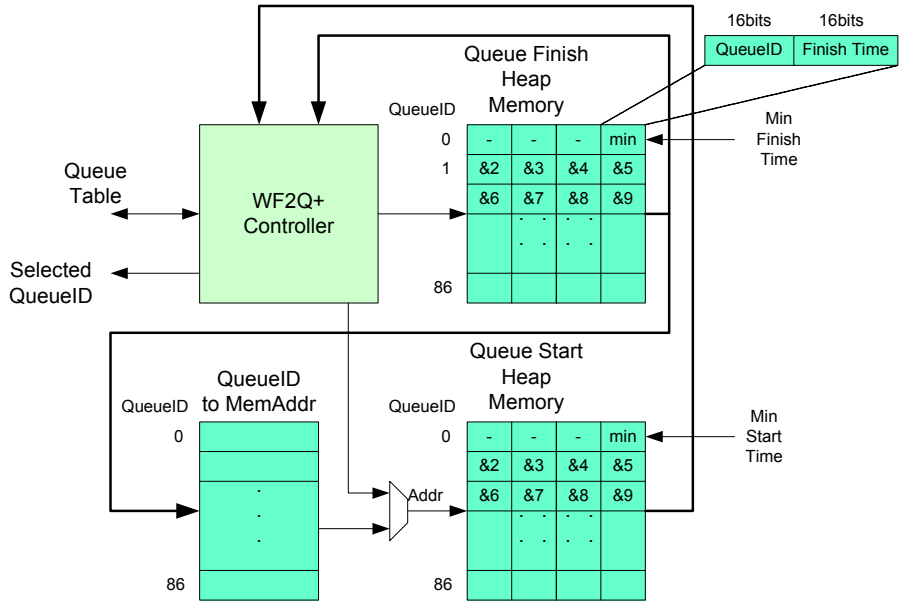

Figure 5.13: The WF2Q+ scheduler

Table 5.6: Queue Scheduler Resource Allocation

Unit	Slices	BRAMs
Table Controller	127	4
DWRR	86	1
WF2Q+	2206	5

the number of active queues is small then the more accurate WF2Q+ scheduler is inserted, while in cases where the DWRR scheduler is used, the freed area can be used to add other hardware acceleration modules such as payload processing accelerators. The effect of the switching of the schedulers in terms of latency, performance and bandwidth allocation is discussed in Chapter 6.

5.3 Payload Processing Accelerators

Section 2.4.1 has presented a short introduction to the requirements of the edge and access routers. These routers must sustain moderate network traf-

fic (in contrast with the core routers that have to sustain higher traffic) but the processing requirements per packet are more complex. As it was mentioned in Section 2.4.1, the more common payload processing functions that are required in edge routers are the encryption and the compression. To evaluate the proposed reconfigurable network processing platform, one encryption module and one compression unit have been used.

The DES encryption module has been used as a typical encryption algorithm. The DES module is a modified version of the DES unit from OpenCores [99]. The unit has been configured as a standard OPB co-processor. The OPB interface has been added, in which it consists of the OPB controller and the user specified registers. The DES module can sustain up to 47MB/sec (376 Mbps) and the whole unit occupies 757 slices.

For the compression requirements, the Lempel-Ziv compression algorithm (also known as LZ77) has been selected. The Lempel-Ziv compression methods are among the most popular algorithm for lossless compression. The LZ77 algorithm achieves compression by replacing portions of the data with references to matching data that has already passed through. The replaced data is encoded by a pair of numbers; the first one represent the length from the start of the compressed data and the second the number of matched characters. To evaluate the performance of the network processing platform a modified version of an open source module [100] has been developed. The unit examines just 32 bits and if there are equal bits, they are replaced by the distance from the start. The basic unit can sustain throughput up to 28Mbytes/sec (224 Mbps) and the whole design occupies 518 slices.

Figure 5.14 depicts the organization of the edge router targeting the reconfigurable network processing platform. The MicroBlaze uses one of the FSL links to communicate with a packet dispatcher. The MicroBlaze sends a command to the dispatcher and the dispatcher sends the first 40 bytes of the packet. These 40 bytes contain the Internet and the Transport's layer header when these headers do not have any option fields (the most common case). In case that the processor needs more data of the packet it sends one more command, and receives the next 40 bytes. The MicroBlaze processes these headers and depending on the network flow that it belongs, it can either simply forward it, or send it for encryption or send it for compression. Attached to the OPB bus there is 64Kbyte memory block that is used as an IP LookUp that stores the information for the forwarding and the classification. The algorithm that is used for searching for the longest prefix match in the LookUp is the Patricia-trie algorithm used in the MiBench benchmark [101].

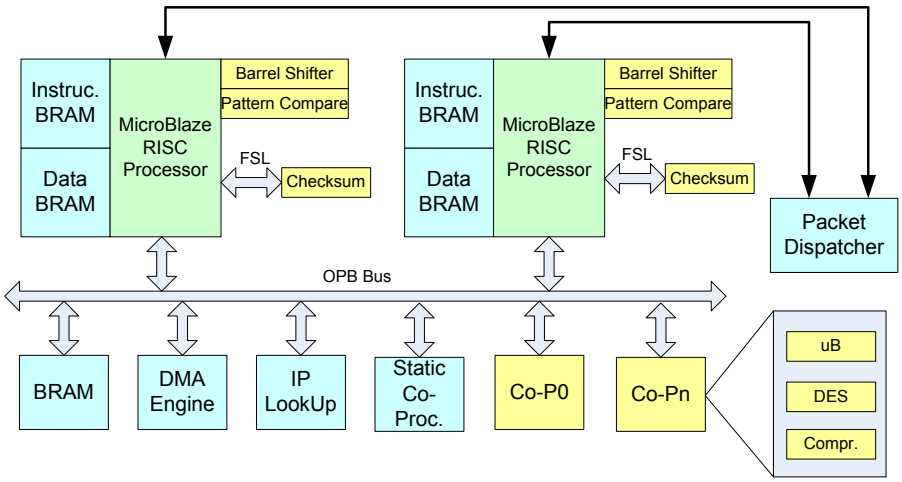

Figure 5.14: Edge router organization

The system is divided into two parts; the static part and the reconfigurable part. The static part contains the MicroBlaze processors, the network interface units, the packet dispatcher, the block RAMs, one Direct Memory Access (DMA) unit, one compression unit and one encryption unit. The reconfigurable part contains two spare hardware units attached to the OPB bus. Each spare unit can be configured to contain either the encryption/decryption unit or the compression/de-compression unit or a processor, depending on the network workload. Each MicroBlaze process the header of the network packet and depending on the network flow that it belongs, it tries to allocate a resource to process the payload of the packet. The status of the configuration (which spare area contains what co-processor) is stored in a special address in a shared memory attached to the OPB bus; hence it is accessible from the MicroBlazes. Each hardware unit has a specific register that is used to store the status of this unit. When the core is busy the resister is set and when the core is free it is zero. When a processor tries to allocate the unit, it first reads this register and if it is clear then it sets the register on the same access (indivisible) in order to prevent deadlocks (test and set). Then, using the DMA unit, it sends the payload of the data from the Block RAM buffer to the hardware unit. The DMA unit needs 4 registers to be set for every transaction; the source address, the destination address, the length of the transfer and the control register which also initializes the transfer.

5.3.1 Configuration Management Unit

The partial reconfiguration of the device can be controlled either by the processor or by a configuration manager. In case that a processor is used the following procedure is followed. Each time a packet header is processed, the processor updates a counter that is used for each network flow. When the total number of processed packets reaches a certain threshold, then the processor checks the network traffic distribution (percentage of packets that belong to each flow) and reset the packet's counters for each flow. If the current configuration (number and type of co-processors) is not the optimum for the specific network traffic distribution then it triggers a reconfiguration. The optimum configuration for the specific network traffic is selected and the old co-processors are replaced by the new co-processors.

In the case that a hardware configuration manager is used the processor can be offloaded from configuring the system. The complexity of the Configuration Management Unit (CMU) depends on the requirements and the fluctuation of the network traffic. A simple algorithm would be to check in which flow the majority of the packets belong and then select the configuration that is optimized for the specific flow. For example, if the percentage of packets that belong to VPN connections is over 50% then the configuration with the maximum number of encryption units is selected. A more advance algorithm can also be used as it is depicted in Figure 5.14. In this case a Look-Up table is used in which the performance of each configuration is stored for several network distributions (the best configuration is shown in the circle). The network traffic distribution can be divided in steps (e.g. of 20%) and the look-up operation can be performed by truncating the last digit for the first two distributions (for example if the network traffic distribution is 23/62/15 the look-up table will return the configuration for the 20/60/20 distribution). The table also stores the performance of the system during the reconfiguration in which some of the co-processors are not used. At the time of the sampling, the configuration manager examines the distribution of the packets and checks if the optimum configuration for this distribution is used. In case that the network distribution has changed (hence, another configuration must be used), the configuration manager should decide if it will perform a new configuration depending on the network stability. The network stability shows for how long the distribution remains the same (within some fluctuation, e.g. +-10%). The configuration manager should perform the reconfiguration only if the new configuration will perform better than the current configuration taking into account the reconfiguration overhead, as it is shown in Equation 5.4. Hence, this configuration

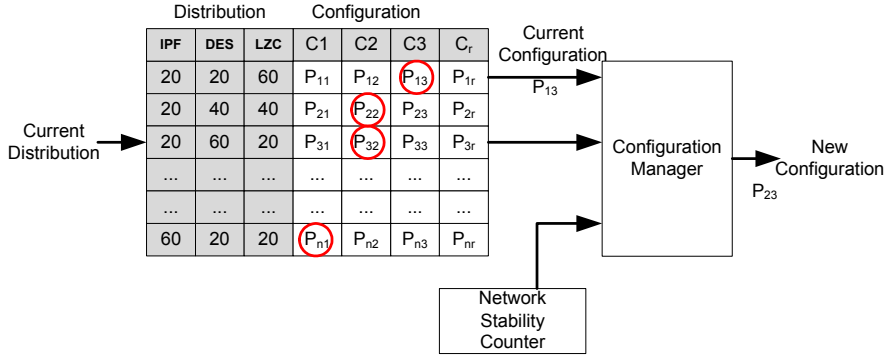

Figure 5.15: Configuration Manager

manager can schedule a new configuration only if it will increase the overall speedup of the system. For example, Figure 5.15 depicts a case in which the current configuration is C3 that is optimized for the 20/20/60 distribution (IPForward/ IPEncryption/ IP Compress). As the network traffic change to 20/60/20 the configuration manager calculates whether the new configuration (C2) will speedup the system taking into account the performance degradation during the reconfiguration and the network stability.

The CMU can also be used to perform the reconfiguration based on queue thresholds for each traffic flow instead of the traffic distribution. Each packet arriving in the platform is classified depending on the network flow that it belong (plain forward, encryption or compression) and it is stored to the corresponding FIFO. The configuration manager can perform the reconfiguration based on the FIFO occupancies of the flows. In this case a similar structure is used in the configuration manager. The only difference is that instead of the distribution, the Look-up table stores the FIFO thresholds. This set-up can be more robust in cases that the network traffic is bursty. For example, if a short burst of encrypted traffic appears in the network, while the system is configured to be optimized for compression, then the system can absorb this bursty traffic thus the system will be not reconfigured. As it is shown in the performance evaluation, in these cases the system achieves better performance than a configuration manager using the packet distribution.

$$P_{new\_config}T_{net\_stab-reconfig} + P_{reconfig}T_{reconfig} \geqslant P_{cur\_config}T_{net\_stab} \tag{5.4}$$

where,

new_config	new configuration
cur_config	current configuration
net_stab	time in which the network traffic is stable
$reconfig$	reconfiguration time

5.4 Conclusions

This chapter presented two typical applications targeting the reconfigurable network processing platform. The first application, content-based switching, has been implemented in the proposed platform. The second application is the IP forwarding for multi-service edge routers. The proposed implementation can reconfigure the number of processor and co-processors based on the processing requirements. Furthermore, a reconfigurable queue scheduler has been presented that use the most suitable algorithm based on the number of active queues. The next chapter presents the performance evaluation of the implemented applications.

Chapter 6

Performance evaluation

This chapter presents the experimental results for the reconfigurable architectures that were proposed. Section 6.1 presents the performance evaluation of the configurable transactional memory controller. The system is evaluated using both micro-benchmarks and typical network processor applications. Section 6.2 presents the performance evaluation of the dynamically reconfigurable edge router. This system is evaluated when the reconfiguration is used to adapt the system to the processing requirements of the network traffic. Section 6.3 presents the performance evaluation for the reconfigurable-based web switch. The system is evaluated in terms of performance, power and area and it is compared with a network-processor based implementation. Finally, Section 6.4 presents the performance evaluation of the reconfigurable queue scheduler. The scheduler is compared in terms of bandwidth allocations against the ideal scheduling and the dynamic reconfiguration overhead is presented. Section 6.5 presents a summary and the conclusion of this chapter.

6.1 Configurable Transactional Memory Controller

This section presents the performance evaluation of the Configurable Transactional Memory Controller (CTMC). To evaluate the proposed scheme a typical application for network processing was developed. Since there is a lack of widely accepted benchmarks for transactional memories, a networking application has been chosen that is widely used for per-flow metering and statistics in network processing equipments (e.g. edge routers). Similar applications

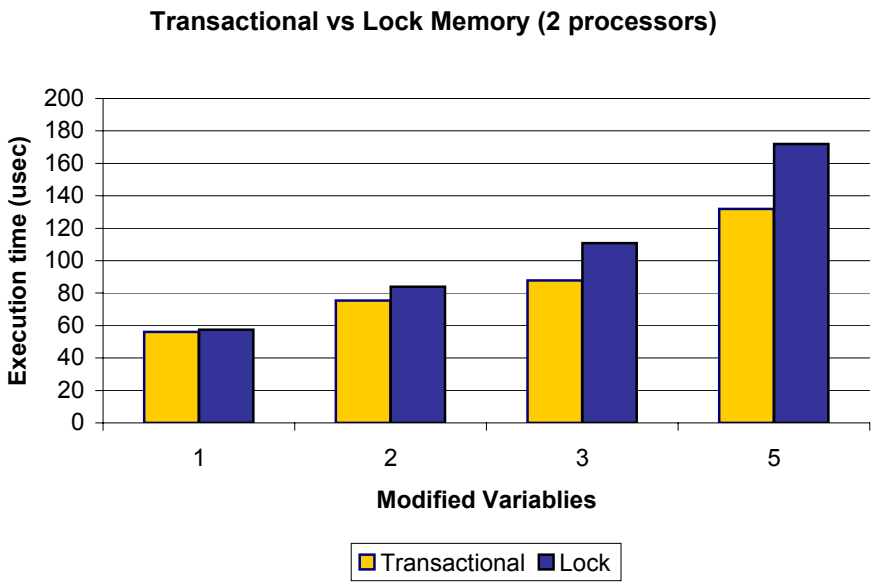

Figure 6.1: Performance comparison for 2 processors

are used in programs for network billing and accounting, performance analysis and security management such as the Argus open source framework [102]. In this application, each time a packet arrives in the system, the header is extracted and forwarded to the next available processor. Each processor extracts some information from the header (Source IP, TCP Port, etc.) and uses this information to update several counters. These counters can be used for billing and Quality of Service purposes.

Figure 6.1 and Figure 6.2 show the performance evaluation of the transactional and lock scheme for 2 and for 4 processors respectively, depending on the number of modified shared variables for the packet metering application. We constructed the configurable transactional memory with various parameters using VHDL and manually instantiated the different parameterized versions of the CTMC in our experiments. The cache that was used was a fully associative cache with 64 entries. The results are obtained from cycle accurate simulation of the RTL code (using the Modelsim simulator) and the timing results are obtained using the Xilinx design platform.

Figure 6.1 and Figure 6.2 depict the execution time to process 50 network packets (each processor). In the case of two processors the speedup of the transactional scheme is from marginal (2% using only 1 shared variable) to

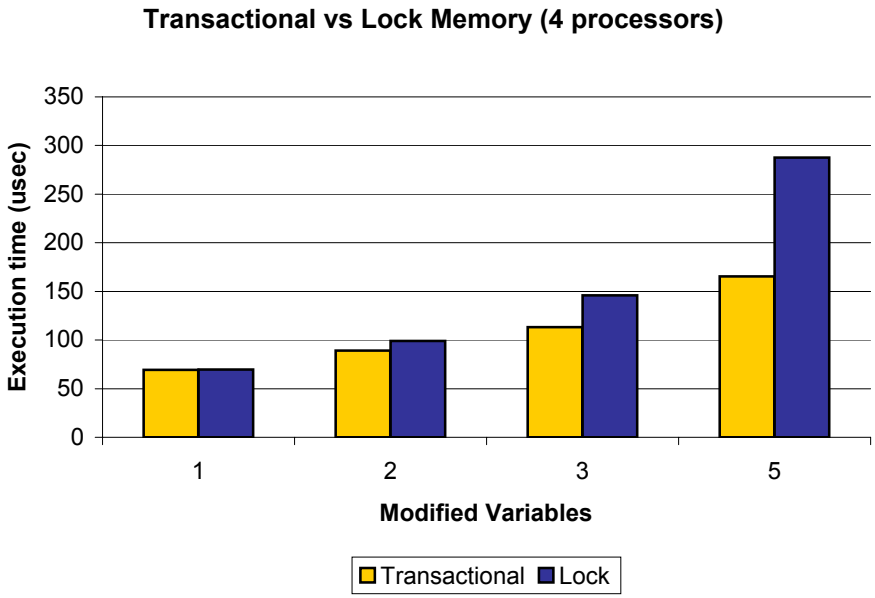

Figure 6.2: Performance comparison for 4 processors

30% using 5 shared variables. In the case of four processors the speedup is from 0% using one shared counter, to 73% using five shared counters. When five shared counters are used, a lot of time is wasted in the instructions that are used to lock and unlock the variables. On the other hand, the transactional scheme uses only one instruction to commit all of the shared variables, hence the execution time is much lower. Therefore, the proposed scheme not only introduces a much simpler programming model for multiprocessor platform but can also be used to improve the execution time of such applications. In the current measurement, the range of the counters that are updated is very wide (1024 different counters that represent 1024 different network flows) hence the probability of collision between 2 processors is very small. In case that the shared variables refer to the same address the probability of collision will be higher thus the execution time will increase.

Figure 6.3 depicts how the performance of the transactional memory depends on the number of rollbacks. The execution time in the figure is the time to process 200 network packets using four processors using the transactional and the locking scheme. The range of counters that had to be updated for each packet was artificially changed to create data dependencies and then the percentage of the rollbacks to the total number of completed transactions was measured. As

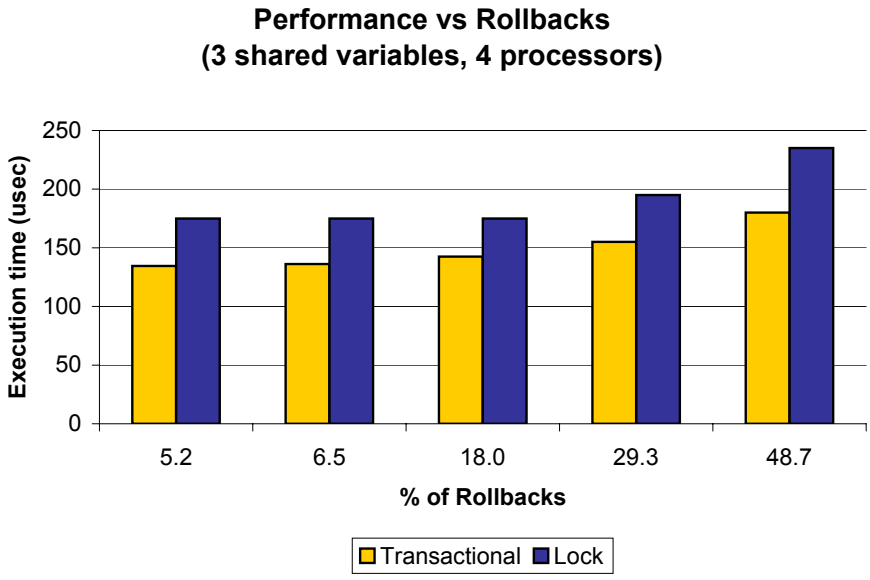

Figure 6.3: Performance over rollbacks

depicted in the figure, the processing time ranges from 20% to 30% less using the transactional memory than the lock-based scheme even when the percentage of rollbacks (percentage of data conflicts) is 50% (one rollback for every commit transaction request). Note that the function that was used to update the counters was consuming almost one third of the total time to process a packet. Hence, the probability of collision has a smaller impact on the total execution time.

To further illustrate how the memory conflicts affect the performance of the system we implemented the micro-benchmarks from [85]. These benchmarks represent two extreme cases for the update of the counters. In the first benchmarks all of the processors update the same counter (same address) creating a high number of conflicts. In the second benchmark all of the processors update a different counter which means that there are no conflicts at all. The performance of the system for these two cases is depicted in Figure 6.4 for a system with four processors. As it shown in the case of the single counter (high conflicts), the processing time to update 400 counters (100 for each processor) is almost 40% less than the processing time using the locking mechanism. The transactional scheme only rollbacks when there is a conflict, while in the case of the lock mechanism the processor is waiting for the other processors to un-

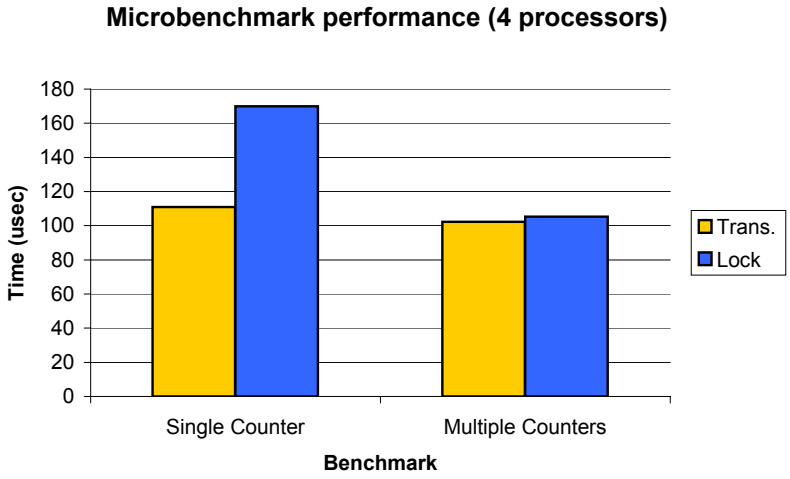

Figure 6.4: Microbenchmark results for 4 processors

lock the specific memory. On the other hand, when there are no conflicts, then the performance of system is independent of the scheme.

6.2 Reconfigurable Edge Router

This section presents the performance evaluation of the dynamically reconfigurable multi-service edge router. Chapter 3 showed that for each flow distribution there is a specific optimum configuration. Each configuration can have different number of processors and co-processors. In case that the reconfiguration is applied only to the co-processors then the reconfiguration overhead is only the reconfiguration time. In case that the reconfiguration is applied also to the number of processors then the reconfiguration overhead is the time to perform the partial reconfiguration, plus the time to initialize the instruction memory, and the time to initialize the processor. The network processing applications (especially in the data plane) are usually simple enough; hence no operating system is used. Therefore the initialization time of the processor is quite small compared to the reconfiguration time.

In case that the task requirements by the processor are not uniform in every flow then the number of processors must also change depending on the network traffic distribution. For example, in case that the tasks performed by the processor for the packets that belong to plain traffic are more complicated than

Table 6.1: Processor performance for several applications

Processor Performance	Cycles
IP Forward	303
DRR (for 20 queues)	1105

Table 6.2: Accelerator's performance

Module	Performance
DES encrypt	304Mbits/sec
LZ compress	240Mbits/sec

the tasks for the other flows, then the performance can be limited by the total throughput of the processors. Therefore the system must reconfigure not only the number of co-processors but also the number of processors. In order to evaluate the system an initial implementation of the system using the Xilinx EDK platform has been used. Using the post place and route simulation model we measured the performance of the system for several configurations. Table 6.1 shows the number of cycles that are required by the MicroBlaze processor for the Deficit Round Robin (DRR) scheduling (from the CommBench benchmark [57]) and the IP forward. The DRR is only applied to the packets that belong to the plain traffic. Table 6.2 shows the performance of the hardware accelerators.

Figure 6.5 depicts the execution time to process 100 packets (the packet size is set to 512 bytes) for several network flow distributions (Forward/Encrypt/Compress) by the 3 configurations (Table 6.3). The figure shows that the processing time change significantly using different configurations. The maximum diversity is noticed in the first three cases (60/20/20, 20/60/20, and 20/20,60) in which one of the flows has the maximum percentage of packets. The last configuration is the one in which the system is performing the reconfiguration. In this configuration the reconfigurable part cannot be used hence the system consists of 2 processors, 1 co-processor for encryption and 1 co-processor for the compression. However, the processing time during the reconfiguration is almost the same with the un-optimized configuration for the flow distribution. For example, in the 20/60/20 flow distribution the processing time using 3 processors, 1 DES, and 2 LZ units is almost the same as the

Table 6.3: Edge Router Configurations

Configuration	# of MicroBlaze	# of DES	# of LZC
Config_A	3	1	2
Config_B	2	3	1
Config_C	2	1	3
Reconfiguration	2	1	1

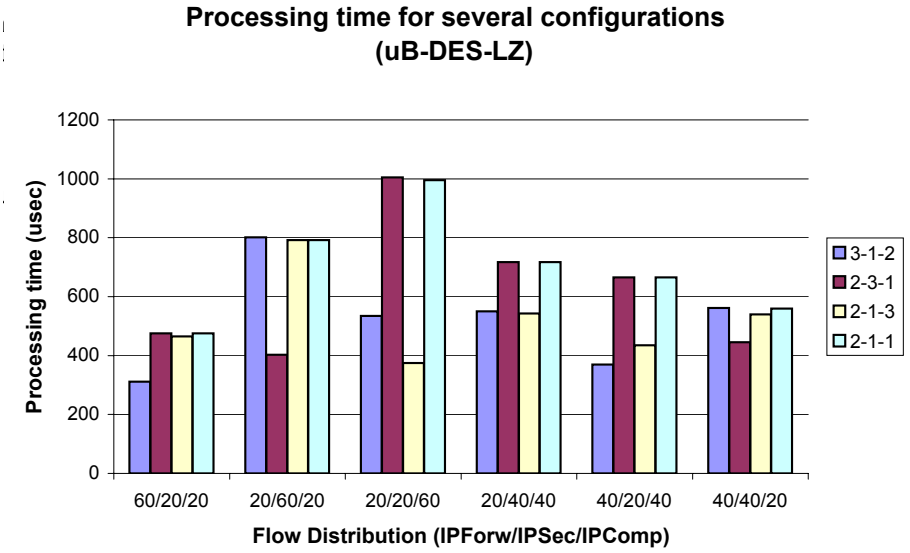

Figure 6.5: Processing time for several configurations

processing time during the reconfiguration (2 processors, 1 DES unit, and 1 LZ unit). This is due to the fact that the bottleneck is the one DES module in these two configurations, therefore there in no any performance overhead during the reconfiguration.

Table 6.4 shows the error between the analytical processing times that was performed in Chapter 3 and the experimental results from the cycle-accurate simulation of the system. The error ranges from 0.7% to 14.7%. The error is mainly caused by the synchronization of the processors and the contention of the resources (co-processors) at the same time that have not been taken into account in the analytical equations. Nevertheless, the table shows that using the analytical formulation of the system, quite an accurate estimation of the

Table 6.4: Error between analytical and experimental results

Configuration	60/20/20	20/60/20	20/20/60
3-1-2	13.1%	0.7%	1.1%
2-3-1	14.7%	12.2%	5.3%
2-1-3	12.9%	1.9%	5.8%

processing time is performed, mainly due to the inherent parallel nature of the network applications.

To measure the speedup of the system compared to a static system, we created 2 types of network traffic. In the first case, the network flow distribution remained the same for a certain amount of time (network stability duration). In the second case, we study the effect of bursty Internet traffic [103] by inserting small quantity of packets in which the flow distribution was different from the current distribution. That way we could measure the performance of the system under bursty traffic. In all of the cases the packets were sent back-to-back to measure the maximum throughput of the system. Based on the characteristics of the Xilinx Virtex4 FPGA [23], the reconfiguration time for 4K slices was estimated as 2ms. Figure 6.6 and Figure 6.7 depict the speedup of the system compared to a static system. The execution time of the static system was measured as the average execution time of the three configurations for each network flow distribution. The speedup is calculated using the following equation.

$$Speedup = \frac{ExecutionTime_{Static}}{ExecutionTime_{Dynamic}}, \tag{6.1}$$

Figure 6.6 depicts the speedup when the network traffic is smooth, using either the flow distribution or the flow's FIFOs thresholds, as it was discuss in Section 5.3.1. During the smooth traffic, the network flow distribution remains the same for a certain time (network stability) within some small fluctuations $\pm 5\%$. As shown in the figure, the Configuration Management Unit (CMU) that reconfigures the system using the flow distribution of the packets is always better than the CMU that used the FIFOs occupancy to perform the reconfiguration. This is due to the fact that the distribution scheme has better response time, since it can recognize the network distribution at processing time. When the network traffic is unstable, (e.g., the network stability is 5 ms) then the FIFO-based CMU has a similar execution time to the average execution time

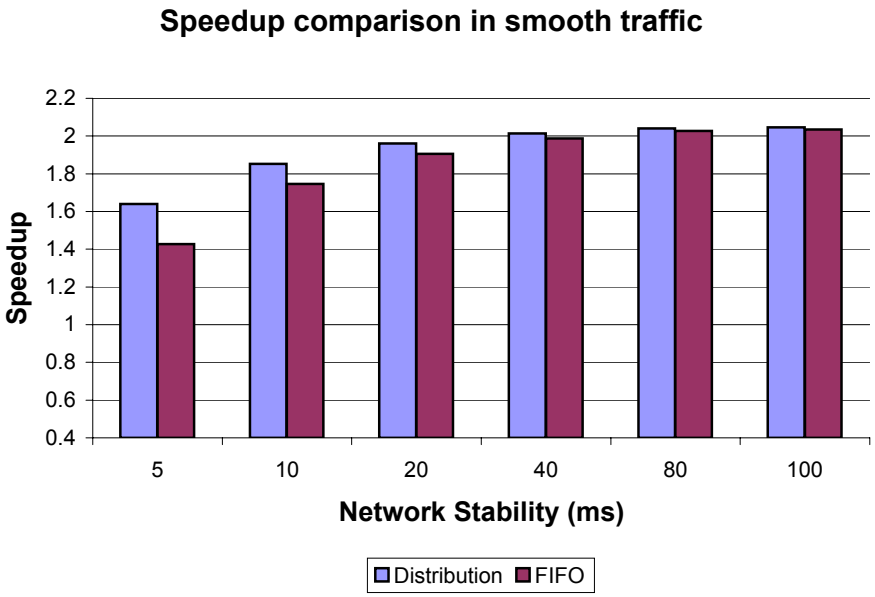

Figure 6.6: Speedup comparison in smooth traffic

of a static system, while the distribution-based CMU is significantly better than static system. The worse execution time of the FIFO-based CMU is due to the fact that the system remains in the reconfiguration state the majority of the time, therefore the overall performance is almost the same as the static system. When the network traffic becomes more stable (network stability is 20ms) then both of the CMUs perform a significant speedup compared with the static system but still the Distribution-based CMU outperforms the FIFO-based CMU. Finally, when the network becomes more stable (100ms) then the two schemes have almost the same speedup.

On the other hand, when the network traffic becomes unstable then the FIFO CMU is better than the distribution CMU. Figure 6.7 depicts the performance of the system when the network traffic is stable for a specific time (network stability) but it is interrupted by fragments of time in which the distribution change for a while and then returns to the previous distribution. The figure depicts the speedup when 2 spikes are inserted into the network traffic with 1ms duration.

Figure 6.8 depicts the comparison of the distribution-based CMU and the FIFO-based CMU for several numbers of burst spikes. As it is shown in the case of one spike in every network stability period the speedup is the same

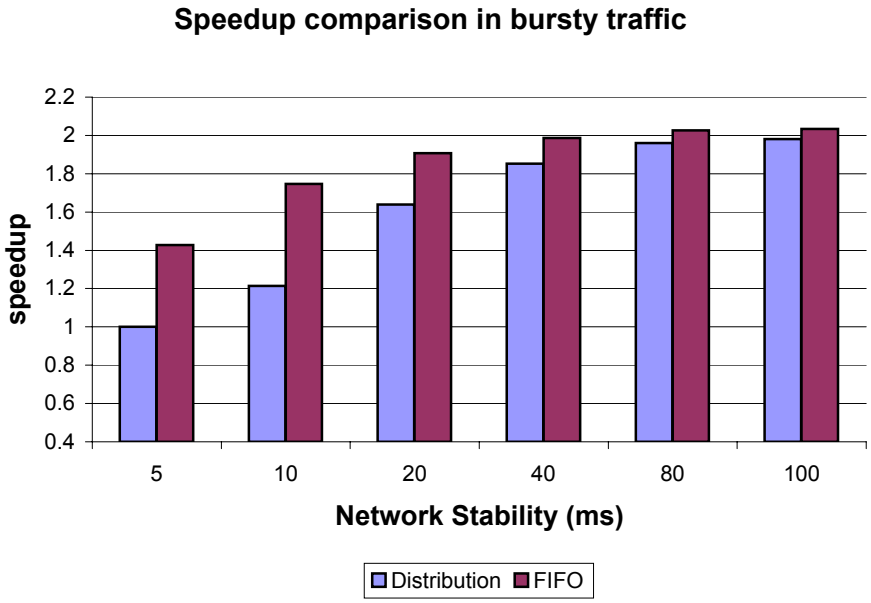

Figure 6.7: Speedup comparison in bursty traffic

for both schemes. As the number of spikes increase the FIFO-based CMU performs better, since it can absorb the traffic spikes. Therefore, a dynamic system could be used in which a module could capture the network stability and activate the appropriate CMU unit. In case of high traffic fluctuations with bursty traffic the FIFO-based CMU would be selected, while in case of smooth traffic fluctuations the Distribution-based CMU would be preferred.

6.3 Reconfigurable Web Switch

This section presents the performance and power consumption evaluation of the reconfigurable-based web switch. The web switch is also compared against a network processor-based implementation in terms of latency and throughput. The system has been implemented into the Xilinx Virtex 4 XCV4LX60 FPGA. The FPGA have usually higher power consumption than the ASICs and lower maximum frequency. On the other hand, the flexibility of the design that they offer is the main advantage. In this section we analyze the performance, the area and the power dissipation of the proposed scheme in order to compare it against a network processor-based implementation. The main advantage of

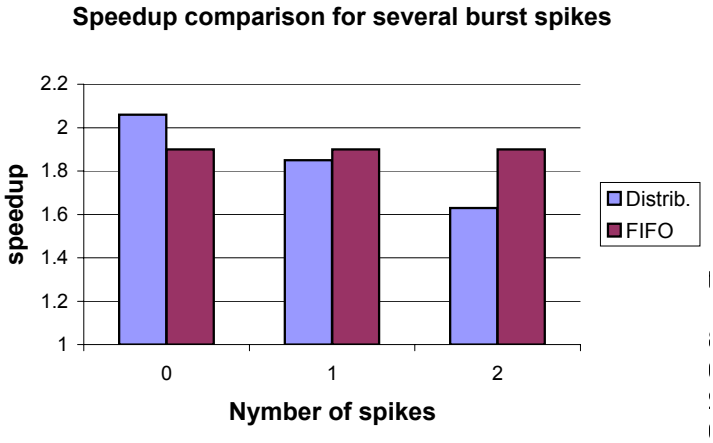

Figure 6.8: Speedup comparison in bursty network for 20 ms

the current network processors is that they offer a multi-threaded multi-core platform that operates at a high frequency. For example, the Intel IXP 2400 processor provides eight 8-way multithreaded micro-engines that operate at 600MHz. The main drawback is that these micro-engines are programmed into a special assembly language; therefore, it does not provide the flexibility of the common processors. On the other hand, the proposed scheme uses the MicroBlaze processors (running at 100MHZ) that can be programmed in C, while the most demanding functions (URL parsing, connection Lookup tables) are implemented in hardware that can be reconfigured.

6.3.1 Performance

Both the MicroBlaze's and the miscellaneous modules operate at 100MHz. Two designs have been used for performance evaluation. The first one uses only one processor, hence the overhead of consistency on multi-processors is eliminated (synchronization, etc). In the second case, two processors are used performing the same tasks. The application that it is hosted in the processors is the spliced web switching using the traces from [92] for the IP address and some of the traces from [95] for the URL matching. We measured the performance of the system with several HTTP file sizes in order to measure how the performance depends on the average file size. The request file size (the HTTP requested file) is a crucial factor. After a connection has been established, the packets are just forwarded after some of the header fields have been changed.

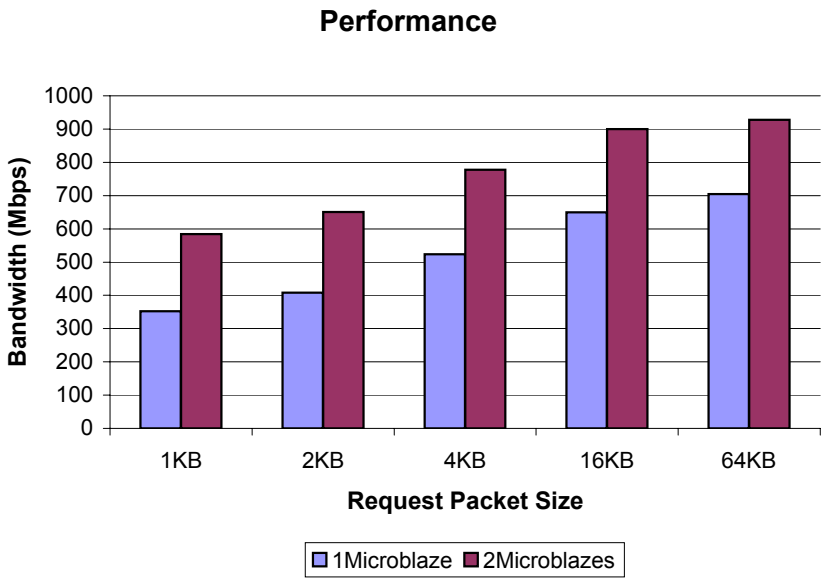

Figure 6.9: Performance of the content-based switch

Therefore, when the file size is small there is a significant overhead for the establishment of the connection between the web switch and the server cluster. Figure 6.9 depicts the performance of the system in both cases for several requested file sizes. The performance is measured in traffic bandwidth (sustained incoming traffic processing). All of the packets were sent back-to-back in order to measure the maximum sustained bandwidth of the system.

Figure 6.10 depicts the utilization of the common bus in both configurations. The OPB common bus is a 32-bit bus that operates at 100MHz. Therefore, the maximum sustained throughput is 3.2Gbps, but the typical data rate is 1.3Gbps. As it is depicted in these figures, in the first case the bottleneck of the design is the processing power of the processor. On the other hand, in the second case, the bottleneck of the design is the bandwidth of the common bus. The network-processor based scheme using the Intel IXP2400 network processor, published in [39], sustain almost 610Mbps for a 64KB packet size while in the case that the requested packet size is 1Kb the sustained throughput is lower than 100Mbps. Hence, the proposed reconfigurable platform scheme offers a significantly higher throughput when the requested file size is small while in the case that the requested file size is larger (16KB) the sustained throughput is almost 55% percent higher. Since the main bottleneck of the design is the bandwidth of the common bus, for larger requested file sizes (1024KB)

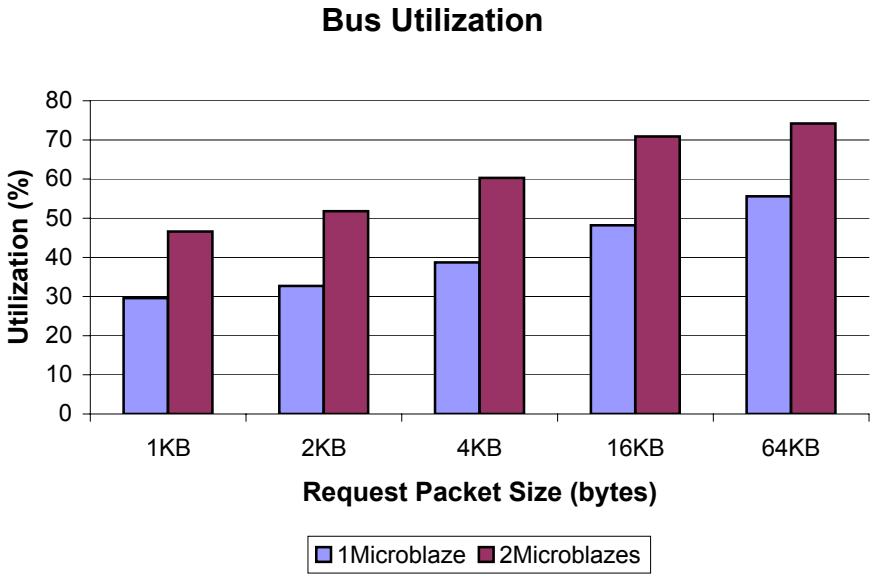

Figure 6.10: Bus utilization

the throughput of the proposed scheme drops to almost 20% higher than the network-processor based scheme (IXP2400).

Table 6.5 shows the average processing latency for various packet types compared to the network processor-based and the software-based scheme presented in [39]. The first three rows of the table show the latency for the control packets. The control packets are used to initialize the connection between the client and the server. The data packets are used after the connection has been established as it was illustrated in Figure 2.6. As it is shown in the table, the latency is very close to the scheme based on the Intel IXP network processor. However, this is the latency for each processor. Since two of these processors are used the performance of the system in terms of throughput is almost 55% higher. Furthermore, this table shows that the proposed architecture have similar performance in terms of latency with the one based on the network processor but in our case the design can be extended with other payload processing modules such as encryption, compression or intrusion detection without wasting processor's cycles.

Table 6.5: Web Switch Processing Latency for packets in us

	Packet Type	FPGA	IXP2400	Linux
Control Packets	SYN	5.5	7.2	48
	ACK/Req.	8.8	8.8	52
	SYN/ACK	8.5	8.5	42
Data Packets	Data	6.9	6.5	13.6
	ACK	6.6	6.5	13.6

Table 6.6: Web Switch Area Allocation

Module	Number	Utilization
Slices	9847	32%
DSP48	6	9%
BRAM16	96	71%
Equivalent gates	301,948	-

6.3.2 Area

Table 6.6 shows the area distribution of the system. As it is shown, the main constraint is the number of Block RAMs that have been used. This table shows also the number of equivalent gates. As it is shown the number of equivalent gates is much smaller than a typical network processor. As a figure of merit, the Agere's PayloadPlus Network Processor occupies 210 million transistors, thus about 35 million gates (assuming 1 gate = 6 transistors). Each MicroBlaze occupies 1088 Slices including the memory controllers and each Gigabit MAC occupies 790 slices. The main area constraint is the use of block RAMs which is 71% of the total block RAMs. The main advantage of the proposed scheme is that the remaining logic elements (slices) could be exploited for additional payload-processing modules (such as encryption, compression, or intrusion detection). Alternatively, the spare logic elements could be used to add SRAM or DRAM controllers that could increase the number of simultaneous connections or the size of the buffers. For example, the network processor-scheme ([39]) which uses the Intel IXP2400 is connected to a 8MB SRAM and a 128MB DRAM hence it can support more connections than our current scheme.

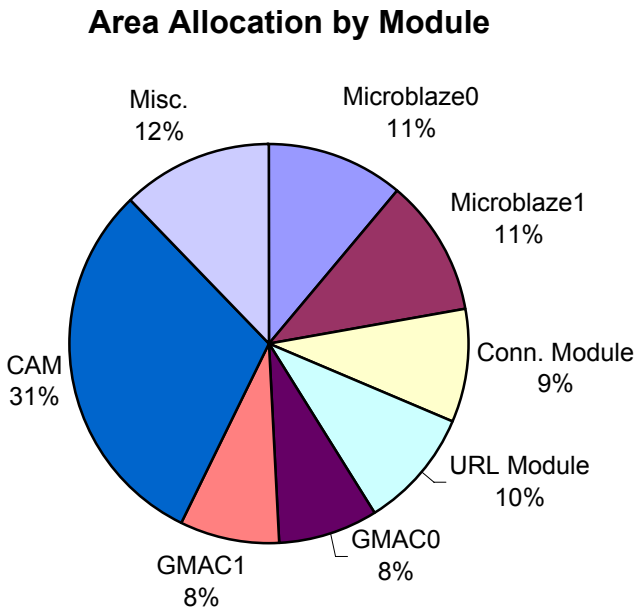

Figure 6.11: Area allocation by Module

Figure 6.11 depicts the area distribution by module. As it is shown in this figure, the CAM for the connections holds almost 31% of the total design area. This is also the main reason that the connection are stored in BRAMs using hash algorithms and only when there is a collision there are stored in the CAM. Each MicroBlaze occupies almost 11% of the area and each Gigabit Media Access Controller occupies 8%. The Connection Module and the URL Module occupies 9% and 10% respectively.

6.3.3 Power Consumption

The main drawback of FPGAs compared to ASICs is the power consumption. Hence, a detailed power analysis has been performed to identify the main sources of power consumption. The following design flow has been used. The system is synthesized, placed and routed using the Xilinx framework. Due to a lack of an evaluation board with 2 gigabit network ports the system has been evaluated from the placed and routed system. The design has been simulated using the Modelsim cycle-accurate simulator and the switching activity of the design has been extracted. The power consumption of the system has been

estimated using the Xilinx's XPower tool. This tool compiles the design and the switching activity and reports the power consumption.

Table 6.7 shows the distribution of the power consumption (without taking account the I/O dynamic power consumption). The power consumption was measured for back-to-back packets with average packet size 512 bytes and includes all the types of packets (from the establishment of the TCP-splice connection until the end of the connection). As it is shown in the table, the dynamic power consumption is 226mW while the total static power consumption is 682mW. Table 6.8 shows the energy that is consumed per packet type, using the data from Table 6.5 and the power consumption. The packet size of the Data packet is considered 512 bytes.

Figure 6.12 depicts the dynamic power distribution by type of the system. According to this figure 48% of the power consumption (108mW) is consumed by the clock distribution. The signals consume 26% of the overall power, while the logic (slices) consumes also 26% of the power. Figure 6.13 depicts the dynamic power distribution by module. As it is shown, each MicroBlaze processor consumes around 45mW, while the data and the instruction memory (BRAM) consumes 17mW. The URL parsing hardware accelerator consumes 4mW and the connection manager consumes only 1mW.

As a figure of merit, in [104] it is shown that the Intel IXP 2400 network processor consumes almost 13.3W for the IPv4 forwarding benchmark achieving 8Gbps forwarding throughput. The low power consumption in our scheme is mainly caused by the use of low frequency components. The typical network processor use micro-processors that operate in high frequency (e.g. the IXP 2400 micro-engines use 600MHz clock frequency). Since, the power consumption is proportional to the frequency the use of 100MHz clock frequency in our scheme keeps the power consumption to low level.

Table 6.7: Web switch power distribution

Type	Power (mW)
Dynamic	226
Quiescent 1.2V	238
Quiescent 2.5V	444
Total	908

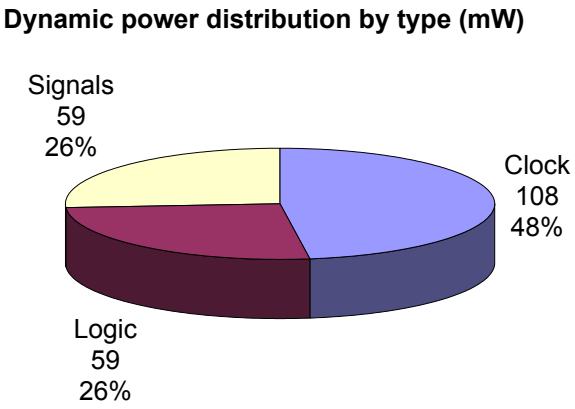

Figure 6.12: Dynamic power distribution by type

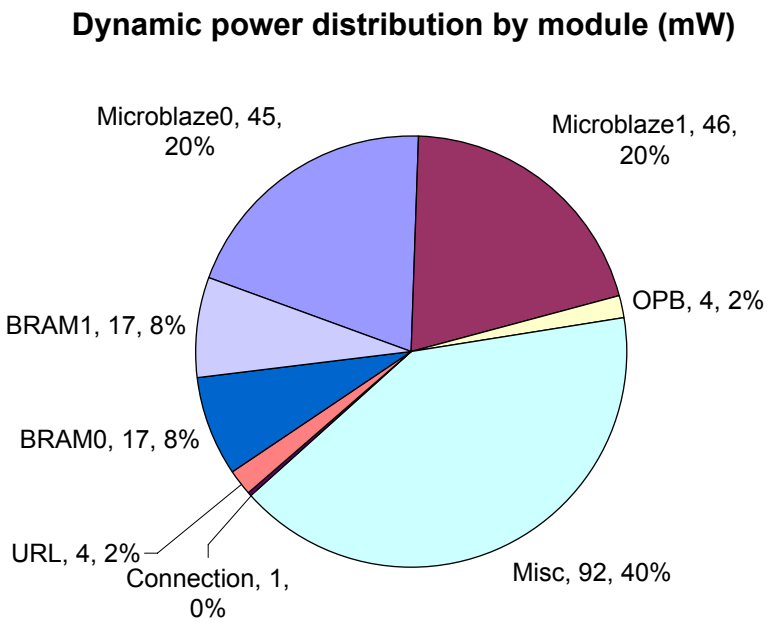

Figure 6.13: Power distribution by module

Table 6.8: Energy per packet type

	Packet Type	Energy (uJoule)
Control Packets	SYN	4.9
	ACK/Req.	7.9
	SYN/ACK	7.7
Data Packets	Data	6.2
	ACK	5.9

6.4 Reconfigurable Queue Scheduler

This section presents the evaluation of the Reconfigurable Queue Scheduler. The queue scheduler is compared against the ideal scheduler (Generalized Processor Sharing-GPS [47]) in terms of bandwidth allocations. The GPS assumes fluid traffic (infinitesimal packet sizes) therefore provides the fairest share of capacity of congested communication links. But because of this assumption of fluid traffic, it is not possible to implement the GPS exactly but only approximations of the algorithm. Therefore, the GPS is used as an ideal scheduler for comparison with other algorithms. Furthermore, this section presents the impact of the dynamic reconfiguration to the system. The main drawbacks of dynamic reconfiguration are twofold. The first one is that during the reconfiguration the queue scheduler is reset; consequently the information of the previous scheduler (e.g. the current queue) is not preserved in the queue scheduler. The second drawback is the timing overhead that it adds, since during the reconfiguration the part of the device that is being reconfigured can not be used. This overhead would cause too much delay in the packets waiting to be transmitted. In addition, the implementation of the schedulers shows that the DWRR occupies much less area than the WF2Q+ scheduler. Hence, when the DWRR scheme is selected, the remaining area would be empty. Thus, in the proposed design the DWRR scheme is used in the static area, while the WF2Q+ scheme is placed in the reconfigurable area as a page-able IP module. When the DWRR is used this spare area can be used for other IP modules such as encryption or compression units used in common network processing applications. In this case we examine the overhead of the dynamic reconfiguration in the allocation of the bandwidth, during the transfer of the scheduler from DWRR to WF2Q+ and vice versa. In the first case, when the system can switch to the WF2Q+ algorithm then the scheduler configures all the active

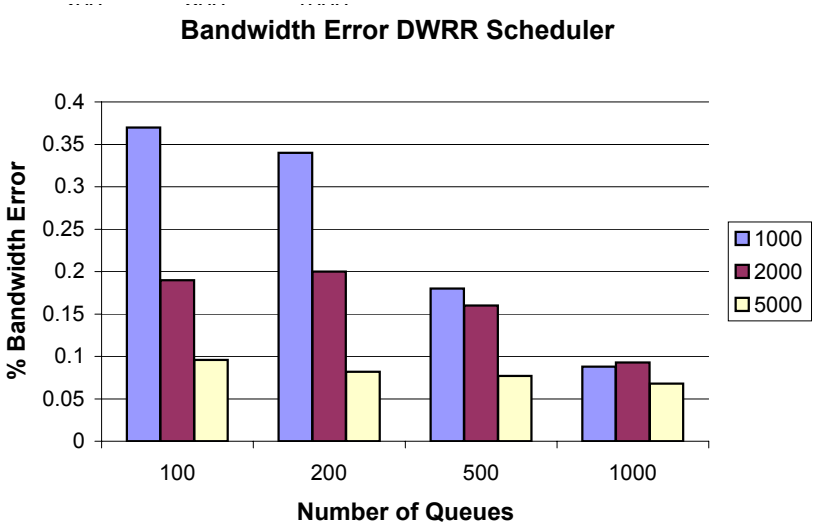

Figure 6.14: Bandwidth error for the DWRR

queues as if they have just arrived in the system (virtual time = 0, start time=0, finish time = start time + size/weight). In the second case, the DWRR counter for all the active queues is set to zero. A management unit can be used to activate the partial reconfiguration. To avoid unnecessary changes, the reconfiguration manager initializes a reconfiguration only if the number of active queues is less than a threshold for a minimum time span. For example, the active queues must be less then 341 queues for a certain time before the system executed the exchange of the schedulers.

Figure 6.14 depicts the impact of the dynamic reconfiguration in the allocation of the bandwidth for the DWRR scheduler. The figure depicts the difference in the bandwidth allocation between the ideal allocation (GPS) and the real allocation for several thresholds of dynamic reconfigurations. When the dynamic reconfiguration takes place after 1000 or 2000 packets then the error in bandwidth allocation compared to GPS varies from 0.02% to 0.06%. If the network is more stable, e.g. the number of active queues do not change very often the error in bandwidth allocation becomes almost negligible. For example, if the number of active queues changes at least every 5000 packets then the error in bandwidth allocation is almost 0.01% which is almost the same to the static version in which a DWRR scheduler is always used.

Figure 6.15 depicts the impact of the reconfiguration for the WF2Q+ scheduler.

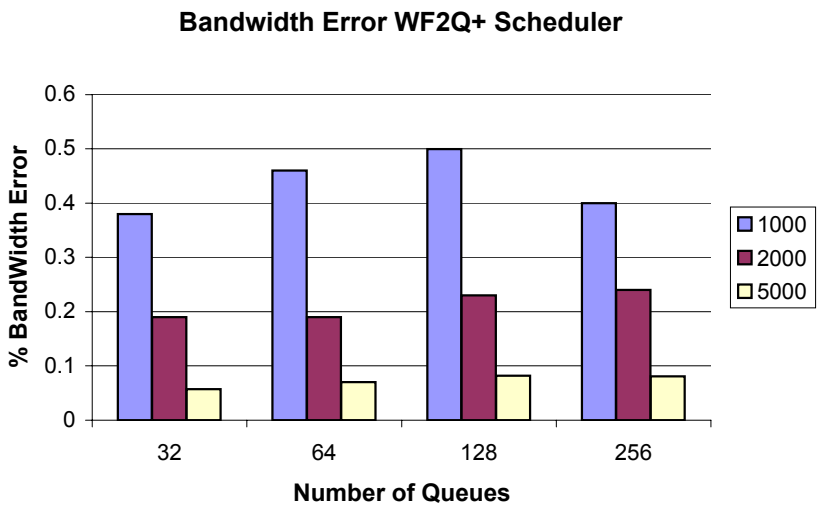

Figure 6.15: Bandwidth error for the WF2Q+

This case shows the error in the bandwidth allocation compared to the ideal scheme (GPS) for several smaller numbers of active queues (32, 64, 128, and 256) since the WF2Q+ scheduler supports less active queues. As it is shown the impact of the reconfiguration is also negligible (0.01% to 0.04%) in the bandwidth allocation.

This scheme also shows both of the scheduler can allocate quite accurately the bandwidth, since the error in bandwidth allocation for the static version is almost 0.01% compared to the ideal case, although in terms of latency the WF2Q+ outperforms the DWRR according to [42]. It is obvious that a network topology in which the number of active queues changes more rarely (more than every 10000 packets, which is a logical assumption according to [18]) then the impact of the dynamic reconfiguration would be infinitesimal.

As it is shown, the overhead of the reconfiguration causes almost no performance degradation in terms of bandwidth allocation, while the use of different schedulers has a major impact on the delay and the jitter for several applications. The timing overhead of the dynamic reconfiguration can be overcome using the DWRR as a static scheduler, hence the transfer of the scheduler from the DWRR to the WF2Q+ scheduler can be achieved in only one clock cycle, when the reconfiguration of the module will be over. Therefore it is shown that the dynamic reconfiguration of the queue scheduler can improve the performance of the network devices in terms of scheduling accuracy when they

are used efficiently by trying to adapt to varied number of active queues.

6.5 Conclusions

This chapter has presented the performance evaluation of the reconfigurable network processing platform for several applications. The performance evaluation of the configurable transactional memory controller has shown that the proposed controller can be used to reduce significantly the processing time of the synchronization of multi-processor platforms in network processing applications such as metering, traffic billing and accounting. At the same time, it offers an easier programming environment for the development of multi-processing applications.

Furthermore, the chapter has presented the performance evaluation for two typical applications in enterprise and edge networks. In the first case, the performance of a web switch that is used in server clusters has been presented and compared with a network-processor based switch. The results shows that the reconfigurable platform can achieve higher performance than the network processor-based implementation, and the proposed platform can be further scaled using addition accelerators to support more functions. Furthermore, the power measurement of the reconfigurable-based platform indicates that the use of hardware accelerators to perform the major network processing tasks can reduce significantly the power consumption compared to a network processor-based implementation. In the case of the reconfigurable edge router, the results shows the use of a reconfigurable platform that adapt to the network traffic processing requirements can be used to develop high performance network systems using much less area (thus lower static power consumption). Finally, the performance evaluation of the reconfigurable queue scheduler shows that the use of an adaptive scheduler that adapts to the number of active queues can provide lower latency and better bandwidth allocation when the number of active queues is small, while it can also support larger number of active queues sacrificing the latency and the bandwidth allocation, without any significant overhead during the reconfiguration of the scheduler.

Chapter 7

General conclusions

This chapter presents a short summary of the dissertation and the main contributions. Furthermore, it presents some future research directions in the area of reconfigurable network processing platforms.

7.1 Summary

In this dissertation, an integrated investigation was presented on how to address the current and future challenges in the network processing domain using reconfigurable architectures.

Chapter 3 described the proposed reconfigurable network processing platform and an integrated flow for this platform. The flow was described from the specification and the design space exploration to the implementation level. Furthermore, a design space exploration was performed on the proposed platform for a typical network processing device to find the optimum configuration for each network traffic pattern. A reconfigurable multi-service edge router was utilized as the case study. The edge router was formulated in analytical equations and a design space exploration was performed to find the optimum configuration. The results of the design space exploration showed that the utilization of a reconfigurable platform that adapts to the network processing requirements can improve the throughput of a network device.

Chapter 4 presented a configurable transactional memory controller that can be utilized in multi-processor reconfigurable platforms. The proposed scheme eliminates the deadlocks that are created from conventional locking schemes. Furthermore, the use of transactions provides an easier environment in the pro-

gramming of parallel processors. The proposed scheme can be configured based on the platforms's characteristics (e.g., number of processors) without adding significant area overhead.

Chapter 5 presented the hardware acceleration modules and the implementation of typical network processing applications to the proposed reconfigurable network processing platform. First, the implementation of a web-switch that can be used in server clusters was presented. Two hardware acceleration modules were developed in this case. The first one is used for the management of connections while the second one is used for the URL string parsing. Furthermore, a reconfigurable queue scheduler was presented. Two widely accepted schedulers were developed in reconfigurable logic and were combined to form an adaptive scheduler that can be reconfigured depending on the number of active queues. Finally, the implementation of a multi-service edger router than can adapt to the network processing requirements was presented. The application is mapped to the proposed platform and a configuration manager was presented that can control the reconfigurable platform based on the network characteristics (e.g., network fluctuation).

Finally, **Chapter 6** presented the performance evaluation of the proposed reconfigurable architectures. This chapter showed that the speedup of a reconfigurable edge router can be up to 1.4 compared with a static version. The reconfigurable-based web switch provides better throughput than a network processor-based implementation with reduced power consumption. The proposed reconfigurable queue scheduler provides low latency and better bandwidth allocation when the number of active queues is small, while it also supports higher number of active queues sacrificing the latency and the bandwidth allocation. The evaluation of the queue scheduler showed that the impact of the reconfiguration in bandwidth allocation is negligible. Finally, the proposed configurable transactional memory controller provides an easier programming environment for the programming of parallel processors and better performance compared to a conventional locking scheme.

7.2 Main Contributions

This dissertation presented an integrated view on the efficient utilization of reconfigurable logic for the development of network processing platforms. The main challenge in the area of network processing is how to handle efficiently the growing traffic, the increased protocol processing complexity and the network traffic fluctuations. The FPGAs have evolved to powerful device with

unique features for the development of network processing applications that combine the performance of hardware with the flexibility of software. The reconfigurable logic inside the FPGAs can be utilized to implement efficient hardware accelerators. These accelerator modules can handle the current and the emerging application requirements such as encryption, compression, media transcoding, and intrusion detection. The soft-core and hard-core processors can be utilized to provide the flexibility of software development that is extremely important in the network processing area where the applications continuously change. Furthermore, the reconfigurable nature of the FPGAs can be exploited both at design time (static reconfiguration) and at run-time (dynamic reconfiguration). At design time, the modules can be configured based on application's characteristics. For example, the URL parsing module can be configured based on the type of the content switching (application-based switch or directory-based switch). The dynamic reconfiguration can be utilized to cope with network traffic fluctuations. The edge router based on the proposed platform can adapt itself to the network traffic fluctuations providing higher throughput by changing the number of processors, the number of co-processors or the algorithm for the queue scheduling.

Overall, this dissertation has presented an integrated view on how to design efficient network processing devices by exploiting the unique features of the FPGAs. The main contribution of this dissertation was to identify which parts of a network processing device can be benefit the most from the reconfigurable nature of the FPGAs and to propose efficient architectures for these parts. The three main parts of a network processing platforms (hardware accelerators, queue schedulers, and pool of parallel processors) have been explored and efficient organizations were proposed. In each of these cases the proposed organizations were presented that exploit either the static reconfiguration or the dynamic reconfiguration. Finally, an integrated performance evaluation was performed in these proposed schemes and it was showed that reconfigurable platforms can address efficiently the current and the near-future challenges in the area of network processing. The main contributions are summarized in the following:

- We proposed an integrated design flow for the reconfigurable network processing platform. The flow include the steps from the problem formulation and the design space exploration to the implementation and the tuning of the configuration.
- We introduced a configurable transactional memory controller that can significantly improve the processing time and provide an easier pro-

gramming framework for the multi-processor network processing platforms. Furthermore, it can be configured to meet the application requirements without adding much area overhead to the platform.

- We introduced a reconfigurable queue scheduler that can be used to provide better results based on the number of active queues. The reconfiguration overhead can be overcome efficiently by using an area-efficient algorithm that remains always in the platform. In case that the number of active queues is small, and there is available area in the platform a more accurate algorithm can be utilized.
- We proposed a reconfigurable network processing platform that can be used for the efficient implementation of edge, access, and enterpise network application. A web switch has been implemented in the platform that can sustain higher bandwidth and consumes lower power compared to a network-processor based web switch.
- We implemented a reconfigurable router for multi-service edge/access routers. The router adapts itself to the network traffic processing requirements by changing the number of processors and co-processors. The reconfiguration overhead can be eliminated by using at least one module for each network traffic pattern and by replicating only the modules that affect the critical processing time of the network traffic.

Overall, the performance evaluation of the proposed platform showed that FPGA-based platforms can be an efficient alternative to the network processors and can face the current and future challenges in the domain of the network processing. Specifically, regarding the challenges that were presented in Section 1.3 the reconfigurable network processing platforms can face efficiently:

- **the increase of the network traffic**: by the use of hardware accelerators and multi-processing architecture supporting transactions
- **the network traffic complexity**: by the use of hardware accelerators that can be relief the processor from time-consuming payload processing
- **the network traffic fluctuation**: by the use of dynamically reconfigurable modules that can adapt to the network traffic requirements
- **the power consumption**: by the use of smaller device exploiting the dynamic reconfiguration and by the use of hardware modules that are more efficient than processor in terms of power consumption

- **the reduced Time-to-Market**: by the inherent reconfigurability (at hardware and software level) of the FPGAs that can be used to support emerging applications

7.3 Future Research Directions

In this dissertation a reconfigurable platform for network processing was presented. The proposed platform can sustain efficiently the processing requirements of devices located at edge, access and enterprise networks. This sections presents some future research directions that could further improve the reconfigurable network processing platforms.

- A research direction could be the investigation of a reconfigurable Network-on-Chip to be used to interconnect the processors and the accelerators depending on the application. The standard shared bus interconnection is expected to reach the upper bound in terms of bandwidth. Therefore, the use of a reconfigurable NoC that can be configured based on the application requirements could provide higher throughput. For example, a NoC that provide QoS features to specific streams (e.g., higher bandwidth to real-time flows such as VoIP) should be able to provide better characteristics in terms of latency and jitter.

- In this thesis some hardware acceleration units have been developed to evaluate the performance of the proposed platform for some mainstream applications (e.g, routing and content switching). The proposed platform could be further expanded by the utilization of additional hardware accelerators for classification, intrusion detection, media transcoding, etc. The utilization of additional accelerators can extend the use of the proposed platform for the development of other network devices such as firewalls, media gateways, wireless base stations, etc.

- Another research direction could be the investigation of more hardware acceleration units that are connected directly to the soft-core processor. Currently, the MicroBlaze can be configured to include barrel shifter and bit matching modules instructions. In the proposed platform, a checksum acceleration unit has been attached to the processors to speedup the header processing. A profiling of network application could reveal more modules to be developed to further increase the performance of the proposed platform.

- Furthermore, the use of other processor architectures could be investigated to show if they will have a major impact to the system's performance. For example, the use of multi-threaded processor combined with reconfigurable logic could be investigated. The utilization of multi-threaded processors can hide the memory and the communications (between the processor and the accelerators) latency, thus increasing the overall performance.

- Finally, a possible research direction would be the investigation of incorporating a hard-wired network processor into the current FPGAs. The incorporation of hard-wired general purpose processors in the FPGAs (e.g., the PowerPC in the Xilinx FPGAs) was a major step in the development of reconfigurable computing, that combined the performance of the fixed general purpose processor with the flexibility of the reconfigurable logic. A possible replacement of the hard-core GPP with a network processor could further rise the use of FPGAs as a complete system for a network processing device.

Bibliography

[1] P. Crowley, M. E. Fluczynski, J.-L. Baer, and B. N. Bershad, "Characterizing processor architectures for programmable network interfaces," in *Proceedings of the International Conference on Supercomputing*, pp. 54–65, 2000.

[2] ITRS, "System drivers." International Technology Roadmap for Semiconductors, 2003.

[3] H. J. Chao and B. Liu, *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.

[4] N. McKeown, "Network processors and their memory," in *3rd Network Processsor Workshop, Invited Talk*, 2004.

[5] A. Telikepalli, "Power vs. perfromance, the 90nm inflection point." Xilinx white paper, Xilinx Inc, 2006.

[6] D. Comer and L. Peterson, *Network Systems Design Using Network Processors*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003.

[7] N. Shah, "Understanding network processors," Master's thesis, University of California, Berkeley, September 2001.

[8] "ASIC and FPGAs for Telecom Applications: A Buyer Survey." Lightreading's Components Insider, vol. 2, no.3, July 2006.

[9] "ESL Expands the Reach of FPGAs." XCell Journal, Issue 58, Xilinx Inc., 2006.

[10] A. Odlyzko, "Internet traffic growth: Sources and implications," in *Proceedings of the Optical Transmission Systems and Equipment for WDM Networking II*, pp. 1–15, 2003.

[11] L. G. Roberts, "Beyond moore's law: Growth trends," *Computer*, vol. 33, no. 1, pp. 117–119, 2000.

[12] G. Varghese, *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[13] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," in *Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM'93)*, (New York, NY, USA), pp. 183–193, ACM Press, 1993.

[14] V. Paxson and S. Floyd, "Wide-area traffic: the failure of poisson modeling," in *Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM'94)*, (New York, NY, USA), pp. 257–268, ACM Press, 1994.

[15] P. B. Danzig, K. Obraczka, and A. Kumar, "An analysis of wide-area name server traffic: a study of the internet domain name system," in *Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM'92)*, (New York, NY, USA), pp. 281–292, ACM Press, 1992.

[16] C. L. Williamson, "Network traffic measurement and modeling," in *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 56–57, ACM Press, 1995.

[17] S. McCreary and K. Claffy, "Trends in wide area IP traffic patterns - A view from ames Internet exchange," *Proceedings of the 13th ITC Specialist Seminar on Internet Traffic Measurement and Modelling, Monterey, CA*, 2000.

[18] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," in *IEEE Network*, pp. 10–23, November/December 1997.

[19] A. Madhukar and C. Williamson, "A Longitudinal Study of P2P Traffic Classification," in *MASCOTS '06: Proceedings of the 14th IEEE Inter-*

national Symposium on Modeling, Analysis, and Simulation, (Washington, DC, USA), pp. 179–188, IEEE Computer Society, 2006.

[20] D. Tang and M. Baker, "Analysis of a local-area wireless network," in *Mobile Computing and Networking*, pp. 1–10, 2000.

[21] K. Claffy, *Internet traffic characterization*. PhD thesis, University of California, San Diego, 1994.

[22] "Internet2 NetFlow, Weekly Reports, Week of 2007-10-22." http://netflow.internet2.edu.

[23] P. Lysaght, "Dynamic Reconfiguration of Xilinx FPGAs." Presentation in IEEE International Conference on Field Programmable Logic and Applications, 2006, www.xilinx.com/univ.

[24] P. C. Lekkas, *Network Processors: Architectures, Protocols and Platforms*. New York, NY, USA: McGraw-Hill, Inc., 2003.

[25] R. Warden, "Design considerations for edge routers." EETimes Comms-Design on-line article, 2002.

[26] F. Baker, "RFC 1812, Requirements for IP Version 4 Routers." Request for Comments (RFC), 1995.

[27] P. Gupta, *Algorithms for routing lookups and packet classification*. PhD thesis, Stanford University, 2000.

[28] R. Thayer, N. Doraswamy, and R. Glenn, "RFC 2411, IP Security Document Roadmap." Request for Comments (RFC), 1998.

[29] R. Friend, "Understaning LZS Compression in TLS security: A tutorial." EETimes on-line article, 2005.

[30] "SAN Extension with Compression and Encryption in WAN Environments." Cisco Whitepaper, Cisco Systems, Inc., 2004.

[31] V. P. L. Ferrigno, S. Marano and A. Pietrosanto, "Balancing computational and transmission power consumption in wireless image sensor networks," in *Proceedings of the 2005 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pp. 1–6, 2005.

[32] D. Maniezzo, K. Yao, and G. Mazzini, "Energetic trade-off between computing and communication resource in multimedia surveillance sensor network," in *Proceedings of the 4th International Workshop on Mobile and Wireless Communications Network, 2002*, pp. 373–376, 2002.

[33] C. Spackman, "Compression/decompression tradeoffs for data networking and storage." EETimes on-line article, 2007.

[34] C. S. Tye and G. Fairhurst, "A Review of IP Packet Compression Techniques." Proceedings of PostGraduate Networking Conference, Liverpool, 2003.

[35] H. Khan and Z. Naseh, *Designing Content Switching Solutions*. Cisco Press, 2006.

[36] D. A. Maltz and P. Bhagwat, "TCP splicing for application layer proxy performance," Research Report RC 21139, IBM, Mar. 1998.

[37] G. Apostolopoulos, D. Aubespin, V. G. J. Peris, P. Pradhan, and D. Saha, "Design, implementation and performance of a content-based switch," in *Proceedings of the Conference on Computer Communications (INFOCOM)*, pp. 1117–1126, 2000.

[38] L. Zhao, Y. Luo, L. Bhuyan, and R. Iyer, "Design and implementation of a content-aware switch using a network processor," in *HOTI '05: Proceedings of the 13th Symposium on High Performance Interconnects*, (Washington, DC, USA), pp. 79–85, IEEE Computer Society, 2005.

[39] L. Zhao, Y. Luo, L. Bhuyan, and R. Iyer, "SpliceNP: a TCP splicer using a network processor," in *ANCS '05: Proceedings of the 2005 symposium on Architecture for networking and communications systems*, (New York, NY, USA), pp. 135–143, ACM Press, 2005.

[40] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a robust software-based router using network processors," *SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 216–229, 2001.

[41] A. Cohen, S. Rangarajan, and H. Slye, "On the performance of TCP splicing for URL-aware redirection," in *USITS'99: Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems*, (Berkeley, CA, USA), pp. 11–11, USENIX Association, 1999.

[42] C. Semeria, "Supporting differentiated service classes: Queue scheduling disciplines." White Paper, Juniper Networks, Inc., 2001.

[43] "Congestion management overview." White paper, Cisco Systems, Inc.

[44] "Bringing comprehensive quality of service capabilities to next-generation networks." White Paper, Freescale Semiconductors, Inc.

[45] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 175–185, 1998.

[46] J. C. R. Bennett and H. Zhang, "WF2Q: Worst-Case Fair Weighted Fair Queueing," in *Proceedings of the Conference on Computer Communications (INFOCOM)*, pp. 120–128, 1996.

[47] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[48] J. C. R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, 1997.

[49] D. Stiliadis, *Traffic scheduling in packet-switched networks: analysis, design, and implementation*. PhD thesis, University of California, Santa Cruz, 1996.

[50] "Technical Guide to APP550 Network Processor." Agere Systems Inc., May 2003.

[51] "Challenges in Designing 40-Gigabit Network Processors." EZChip Inc., White paper, June 2002.

[52] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration, the VLSI Journal, Elsevier*, vol. 38, pp. 131–183, December 2004.

[53] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer, "Comparing analytical modeling with simulation for network processors: A case study," in *Design, Automation, and Test in Europe (DATE), Munich, Germany*, pp. 256–261, March 2003.

[54] L. Thiele, S. Chakraborty, M. Gries, and S. Knzli, "Design space exploration of network processor architectures," in *First Workshop on Network Processors at the 8th International Symposium on High Performance Computer Architecture (HPCA8)*, 2002.

[55] J. Yujia, S. Nadathur, R. Kaushik, and K. Kurt, "An Automated Exploration Framework for FPGA-Based Soft Multiprocessor Systems," in *Proceedings of the 2005 International Conference on Hardware/Software Codesign and System Synthesis (CODES-05)*, pp. 273–278, September 2005.

[56] T. Wolf and M. A. Franklin, "Design tradeoffs for embedded network processors," in *ARCS '02: Proceedings of the International Conference on Architecture of Computing Systems*, (London, UK), pp. 149–164, Springer-Verlag, 2002.

[57] T. Wolf and M. Franklin, "Commbench - a telecommunications benchmark for network processors," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 154–162, 2000.

[58] K. Lahiri, A. Raghunathan, and S. Dey, "System level performance analysis for designing on-chip communication architectures," in *IEEE Trans. on Computer Aided-Design of Integrated Circuits and Systems*, vol. 20, pp. 768–783, 2001.

[59] P. Paulin, C. Pilkington, and E. Bensoudane, "StepNP: A System-Level Exploration Platform for Network Processors," *IEEE Design and Test*, vol. 19, no. 6, pp. 17–26, 2002.

[60] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard, "Application of a Multi-Processor SoC Platform to High-Speed Packet Forwarding," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, pp. 300–58, IEEE Computer Society, 2004.

[61] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable network packet processing on the field programmable port extender (FPX)," in *FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, (New York, NY, USA), pp. 87–93, ACM Press, 2001.

[62] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *DAC '02: Proceedings of the 39th conference on Design automation*, (New York, NY, USA), pp. 343–348, ACM Press, 2002.

[63] I. Hadzic, J. Smith, and W. Marcus, "On the fly programmable hardware for networks," in *Proceedings of the IEEE Global Telecommunications Conference, GLOBECOM'98*, pp. 821–826, 1998.

[64] R. Ohlendorf, A. Herkersdorf, and T. Wild, "FlexPath NP: a network processor concept with application-driven flexible processing paths," in *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, (New York, NY, USA), pp. 279–284, ACM Press, 2005.

[65] C. Albrecht, R. Koch, and E. Maehle, "DynaCORE; A Dynamically Reconfigurable Coprocessor Architecture for Network Processors," in *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, (Washington, DC, USA), pp. 101–108, IEEE Computer Society, 2006.

[66] N. G. Bartzoudis, "Reconfigurable computing and active networks," in *Proceedings of the Engineering of Reconfigurable Systems and Algorithms*, 2003.

[67] N. G. Bartzoudis, "Active networking using programmable hardware," in *Proceedings of the PostGraduate Networking Conference*, 2003.

[68] G. Memik, S. O. Memik, and W. H. Mangione-Smith, "Design and analysis of a layer seven network processor accelerator using reconfigurable logic," in *FCCM '02: Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, (Washington, DC, USA), p. 131, IEEE Computer Society, 2002.

[69] A. Dollas, D. Pnevmatikatos, N. Aslanides, S. Kavvadias, E. Sotiriades, S. Zogopoulos, K. Papademetriou, N. Chrysos, K. Harteros, E. Antonidakis, and N. Petrakis, "Architecture and Application of PLATO, A Reconfigurable Active Network Platform," in *FCCM '01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, (Washington, DC, USA), pp. 101–110, IEEE Computer Society, 2001.

[70] U. Nordqvist, *Protocol Processing in Network Terminals*. PhD thesis, Linkoping University, 2004.

[71] T. Henriksson, *Intra-Packet Data-Flow Protocol Processor*. PhD thesis, Linkoping University, 2003.

[72] "Processor IP Reference Guide." Application Note, Xilinx, Inc., February 2005.

[73] C. S. M. Gries, C. Kulkarni and K. Keutzer, "Exploring trade-offs in performance and programmability of processing element topologies for network processors," in *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*, pp. 75–87, February 2003.

[74] X. Huang and T. Wolf, "A methodology for evaluating runtime support in network processors," in *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, (New York, NY, USA), pp. 113–122, ACM Press, 2006.

[75] V. Manral, "RFC 4835, Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)." Request for Comments (RFC), 2007.

[76] R. P. A. Shacham, R. Monsour and M. Thomas, "RFC 3173, IP Payload Compression Protocol (IPComp)." Request for Comments (RFC), 1998.

[77] B. Crepps, "How to provide a power-efficient architecture." EETimes on-line article, Intel Corporation, 2007.

[78] C. Sotiriou and Y. Papaefstathiou, "Design-space exploration of a cryptography algorithm," in *Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems*, pp. 858–861, 2003.

[79] J. Bicknell, "Statistics engine reduces packet processor cycles 90%." IDT, Online article on Network Systems Design Line, 2006.

[80] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proceedings of the 20th International Symposium on Computer Architecture (ISCA)*, pp. 289–300, 1993.

[81] L. Hammond, B. D. Carlstrom, V. Wong, M. Chen, C. Kozyrakis, and K. Olukotun, "Transactional coherence and consistency: Simplifying parallel hardware and software," *IEEE Micro*, vol. 24, no. 6, pp. 92–103, 2004.

[82] A. McDonald, J. Chung, H. Chafi, C. C. Minh, B. D. Carlstrom, L. Hammond, C. Kozyrakis, and K. Olukotun, "Characterization of TCC on

Chip-Multiprocessors," in *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, (Washington, DC, USA), pp. 63–74, IEEE Computer Society, 2005.

[83] R. Rajwar and J. R. Goodman, "Speculative lock elision: enabling highly concurrent multithreaded execution," in *MICRO 34: Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, (Washington, DC, USA), pp. 294–305, IEEE Computer Society, 2001.

[84] R. Rajwar and J. R. Goodman, "Transactional execution: Toward reliable, high-performance multithreading.," *IEEE Micro*, vol. 23, pp. 117–125, Nov-Dec 2003.

[85] R. Rajwar and J. R. Goodman, "Transactional lock-free execution of lock-based programs," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, (New York, NY, USA), pp. 5–17, ACM Press, 2002.

[86] S. Lie, "Hardware support for unbounded transactional memory," Master's thesis, May 2004. Massachusetts Institute of Technology.

[87] S. Grinberg and S. Weiss, "Investigation of Transactional Memory Using FPGAs," in *Proceeding of IEEE 24th Convention of Electrical and Electronics Engineers in Israel*, 2006.

[88] N. Njoroge, S. Wee, J. Casper, J. Burdick, Y. Teslyar, C. Kozyrakis, and K. Olukotun, "Building and Using the ATLAS Transactional Memory System," in *Workshop on Architecture Research using FPGA Platforms, 12th International Symposium on High-Performance Computer Architecture*, 2006.

[89] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Lecture Notes in Computer Science*, vol. 2161, 2001.

[90] J.-L. Brelet and B. New, "Designing Felxible, Fast CAMs with Virtex Family FPGAs." Xilinx Application Note, XAPP203, September 1999, Xilinx, Inc.

[91] A. Moestedt and P. Sjdin, "IP Address Lookup in Hardware for High-Speed Routing," in *IEEE Hot Interconnects VI*, pp. 31–39, 1998.

[92] "UCB-home-IP traces Nov 17." The Internet Traffic Archive: UC Berkeley Home IP Web Traces.

[93] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. Cambridge, MA, USA: MIT Press, 2001.

[94] "Compact Trie." National Institute of Standards and Technology (NIST), http://www.nist.gov.

[95] "Web caching." San Diego Super Computer Center Web Traces.

[96] H. Fallside, "Queue manager reference design." XAPP511, Xilinx Application Note, Xilinx, Inc., 2004.

[97] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, (New York, NY, USA), pp. 231–242, ACM Press, 1995.

[98] "Two flows for partial reconfiguration: Module based or difference based." XAPP290, Xilinx Application Note, Xilinx, Inc., 2004.

[99] R. Usselmann, "DES/Triple DES IP Core." OpenCores.org.

[100] Z. Zeng, "Lempel-Ziv Hardware Implementation Project." University of Masschusetts.

[101] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, (Washington, DC, USA), pp. 3–14, IEEE Computer Society, 2001.

[102] "Argus open project-auditing network activity." QoSient, LLC., 2006.

[103] K. Siriwong, L. Lipsky, and R. Ammar, "Study of bursty internet traffic," in *Sixth IEEE International Symposium on Network Computing and Applications, NCA 2007*, pp. 53–60, 2007.

[104] "IXP2400 Intel Network Processor IPv4 Forwarding Benchmark Full Disclosure Report for Gigabit Ethernet." Network Processing Forum, 2003.

List of Publications

1. C. Kachris, S. Wong, and S. Vassiliadis, **Design and Performance Evaluation of an Adaptive FPGA for Network Applications**, in *Elsevier Microelectronics Journal*, accepted for publication, to appear in 2008

2. C. Kachris, and S. Vassiliadis, **A Reconfigurable Platform for Multi-Service Edge Routers**, In *Proceedings of the 20th ACM Conference on Integrated Circuits and Systems Design (SBCCI'07)*, pp. 165-170, Rio de Janeiro, Brazil, September 2007

3. C. Kachris, and S. Vassiliadis, **Design Space Exploration of Configuration Manager for Network Processing Applications**, In *Proceedings of IEEE International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS'07)*, pp. 34-40, Samos, Greece, July 2007

4. C. Kachris, and C. Koulkarni, **Configurable Transactional Memory**, In *Proceedings of 15th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*, pp. 65-72, Napa Valley, CA, USA, April 2007

5. C. Kachris, and S. Vassiliadis, **Design of a Web Switch in a Reconfigurable Platform**, In *Proceedings of the 2nd ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'06)*, pp. 31-40, San Jose, CA, USA, December 2006

6. C. Kachris, and S. Vassiliadis, **A Dynamically Reconfigurable Queue Scheduler**, In *Proceedings of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL'06)*, Madrid, Spain, August 2006

7. C. Kachris, and S. Vassiliadis, **Performance Evaluation of an Adaptive FPGA for Network Applications**, In *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping (RSP'06)*, pp. 54-62, Chania, Greece, June 2006

8. C. Kachris, and S. Vassiliadis, **Analysis of a Reconfigurable Network Processor**, In *Proceedings of the 13th Reconfigurable Arcitectures Workshop (RAW'06)*, pp. 1-8, IEEE IPDPS, Rhodos, Greece, April 2006

Earlier publications, not directly related to this dissertation:

1. L. Mhamdi, M. Hamdi, C. Kachris, S. Wong, and S. Vassiliadis, **High-performance switching based on buffered crossbar fabrics**, *Computer Networks Journal*, pp. 2271-2285, September 2006, Vol. 50, No. 13

2. L. Mhamdi, C. Kachris, and S. Vassiliadis, **A reconfigurable hardware based embedded scheduler for buffered crossbar switches**, In *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'06)*, Monterey, CA, USA, February 2006

3. A. Dollas, I. Ermis, I. Koidis, I. Zisis, C. Kachris, **An Open TCP/IP Core for Reconfigurable Logic**, In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'05)*, Napa Valley, CA, USA, April 2005

4. P.D. Dimitropoulos, C. Kachris, D.P. Karampatzakis, G.I. Stamoulis, **A new SOI monolithic capacitive sensor for absolute and differential pressure measurements**, In *Sensors and Actuators A: Physical*, vol. 123-124, pp. 36-43, 2005

5. I. Papaefstathiou, G. Kornaros, T. Orphanoudakis, C. Kachris, **Queue management in Network Processors**, In *Proceedings of Design, Automation and Test in Europe (DATE'05)*, Munich, Germany, March 2005

6. A. Nikologiannisa, I. Papaefstathiou, G. Kornaros, and C. Kachris, **An FPGA-based queue management system for high speed networking devices**, In *Elsevier Journal on Microprocessors and Microsystems*, vol. 28, issues 5-6, pp. 223-236, 2004

7. C. Kachris, N. Bourbakis, and A. Dollas, **A Reconfigurable Logic-Based Processor for the SCAN Image and Video Encryption Algorithm**, In *International Journal of Parallel Programming*, vol. 31, no. 6, pp. 489-506, 2003

8. A. Dollas, C. Kachris, and N. Bourbakis, **Performance Analysis of Fixed, and Custom Architectures for the SCAN Image and Video Encryption Algorithm** , In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'03)*, Napa Valley, CA, USA, April 2003

9. C. Kachris, A. Dollas, N. Bourbakis, S. Maniccam, **A Reconfigurable Logic-based Processor for the SCAN Image and Video Encryption Algorithm**, In *Workshop on Application Specific Processors (WASP'02)*, Istanbul, Turkey, November 2002

Samenvatting

Dit proefschrift beschrijft ons onderzoek om herconfigureerbare hardware te gebruiken ten einde flexibele, goed presterende en power-efficiënte netwerk apparaten te ontwerpen die in staat zijn om zich aan te passen aan de eisen van netwerkapplicaties en netwerkverkeer die continu veranderen. Het voorgestelde netwerkverwerkingsplatform is voornamelijk bedoeld voor access-, edge- en enterprise-apparaten. Deze apparaten hebben te maken met minder bandbreedte dan apparaten binnen core-netwerken, maar zij moeten meer operaties per pakket uitvoeren (bijvoorbeeld: payload processing). Bovendien moeten zij flexibel zijn om toekomstige netwerkapplicaties te kunnen ondersteunen. Een veelbelovende technologie om deze apparaten te verwezenlijken zijn de Field-Programmable Gate Arrays (FPGAs). FPGAs zijn typisch in staat om flexibiliteit (door herconfiguratie) met prestatie (door parallelle hardware implementaties) te combineren. Zij zullen dus in staat moeten zijn om aan de eisen gesteld door edge- en access netwerkapparaten te voldoen.

Wij presenteren een herconfigureerbaar verwerkingsplatform dat gebruik maakt van herconfigureerbare hardware-versnellers, een herconfigureerbare queue scheduler en een herconfigureerbaar transactioneel geheugencontrolemechanisme. We hebben de prestaties en randvoorwaarden van het platform geformuleerd als een integer optimalisatieprobleem en een geïntegreerd ontwerpproces gepresenteerd voor het platform. Verder onderzoeken wij in dit proefschrift statische en dynamische herconfiguratie. Statische herconfiguratie wordt gebruikt om de verscheidenheid van eisen ten opzichte van netwerkverwerking te ondervangen terwijl dynamische herconfiguratie wordt toegewend om de fluctuaties van netwerkverkeer te kunnen ondervangen.

Twee representatieve apparaten zijn geïmplementeerd en geëvalueerd op het voorgestelde platform: een multi-service edge router en een content-based (web) switch. In het eerste apparaat wordt dynamische herconfiguratie toegepast om fluctuaties in het netwerkverkeer op te vangen. Het apparaat controleert het netwerkverkeer en past zich aan de netwerkfluctuaties aan en houdt daartoe rekening met herconfiguratietijden. In het tweede apparaat wordt een herconfigureerbare architectuur gebruikt voor een content-based web switch. Deze wordt tevens vergeleken met een gebruikelijke netwerkprocessor in termen van prestatie en vermogen. Het platform herbergt verscheidene processoren die kunnen worden verwisseld om verschillende typen van switching uit te voeren (bijvoorbeeld: URL-gebaseerde of cookie-gebaseerde switching).

Ook wordt het gebruik van herconfigureerbare logic onderzocht voor gebruik in queue scheduling binnen netwerkapparaten. Hiertoe wordt een herconfigureerbare queue scheduler gepresenteerd die zich aanpast aan het aantal actieve queues welke gebruikt kan worden in edge routers en web switches.

Als laatst presenteren wij transactionele geheugens die kunnen worden gebruikt voor multiprocessing platformen bedoeld voor netwerkapplicaties. Het gepresenteerde configureerbare transactionele geheugencontrolemechanisme kan worden geconfigureerd gebaseerd op de applicatie en platformeigenschappen (bijvoorbeeld: aantal processoren), kan een makkelijkere programmeerraamwerk voor multi-processor herconfigureerbare platformen vormen, en biedt verbeterde prestaties vergeleken met traditionele uitsluitingsmethoden. Het resultaat van het onderzoek dat is gepresenteerd in dit proefschrift toont aan dat FPGAs een efficiënt alternatief voor netwerkprocessoren kan vormen en niet alleen kan worden gebruikt voor lagere netwerklagen, maar ook kan dienen als een compleet platform voor toekomstige netwerkverwerkingsapplicaties.

Curriculum Vitae

Christoforos Kachris was born on the 30th of April, 1978 in Athens, Greece. In 1996 he was admitted in the Electronic and Computer Engineering department of the Technical University of Crete, Greece, where he obtained the Diploma (*ptychio*: 5-years degree) and the M.Sc. in Electronic and Computer Engineering in 2001 and 2003, respectively.

In 2003, he joined Ellemedia Technologies, Athens, as an IC/FPGA engineer, where he worked in several European-Union research projects related to the development of network processors for broadband residential and media gateways.

In January 2005, he joined the Computer Engineering (CE) laboratory of Delft University of Technology (TU Delft), The Netherlands, as a researcher, where he worked towards his Ph.D. degree with scientific advisor prof. dr. Stamatis Vassiliadis. In 2006, he worked as an intern research engineer at Xilinx Research Labs (Networks Group), in San Jose, CA, USA.

Christoforos Kachris is a member of the IEEE, the ACM and the Technical Chamber of Greece. During his Ph.D. he contributed to the HiPEAC Roadmap on embedded systems and served as a peer reviewer in several major international journals and conferences.

His current research interests include: reconfigurable computing, network processing, image processing, embedded systems, multi-processor SoCs, and computer architecture.