HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Signal Processing and Acoustics

Pekka Korpinen

Using reconfigurable hardware to distribute network traffic for parallel processing

Licentiate's Thesis submitted in partial fulfillment of the requirements for the degree of Licentiate of Science in Technology.

Espoo, 6th April 2009

Supervisor:

Professor Jorma Skyttä

Second Examiner:

Marko Luoma, Lic.Tech.

Author: Pekka Korpinen

Title: Using reconfigurable hardware to distribute network traffic for parallel processing

Date: 6th April 2009 **Number of pages:** 95

Faculty: Faculty of Electronics, Communications and Automation

Professorship: S-88 Signal Processing Technology

Supervisor: Professor Jorma Skyttä

Second examiner: Marko Luoma, Lic.Tech.

The expanding diversity and amount of traffic in the Internet requires increasingly higher performing devices for protecting our networks against malicious activities. The computational load of these devices may be divided over multiple processing nodes operating in parallel to reduce the computation load of a single node. However, this requires a dedicated controller that can distribute the traffic to and from the nodes at wire-speed. This thesis concentrates on the system topologies and on the implementation aspects of the controller. A field-programmable gate array (FPGA) device, based on a reconfigurable logic array, is used for implementation because of its integrated circuit like performance and high-grain programmability. Two hardware implementations were developed; a straightforward design for 1-gigabit Ethernet, and a modular, highly parameterizable design for 10-gigabit Ethernet. The designs were verified by simulations and synthesizable testbenches. The designs were synthesized on different FPGA devices while varying parameters to analyze the achieved performance. High-end FPGA devices, such as Altera Stratix family, met the target processing speed of 10-gigabit Ethernet. The measurements show that the controller's latency is comparable to a typical switch. The results confirm that reconfigurable hardware is the proper platform for low-level network processing where the performance is prioritized over other features. The designed architecture is versatile and adaptable to applications expecting similar characteristics.

Keywords: hardware design, reconfigurable hardware, FPGA, Ethernet, network processing, distributed computing

Tekijä: Pekka Korpinen

Työn nimi: Verkkoliikenteen hajauttaminen rinnakkaisprosessoitavaksi ohjelmoitavan piirin avulla

Päivämäärä: 6.4.2009 **Sivumäärä:** 95

Tiedekunta: Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: S-88 Signaalinkäsittelytekniikka

Ohjaaja: Professori Jorma Skyttä

Toinen tarkastaja: TkL Marko Luoma

Internetin edelleen lisääntyvä ja monipuolistuva liikenne vaatii entistä tehokkaampia laitteita suojaamaan tietoliikenneverkkoja tunkeutumisia vastaan. Tietoliikennelaitteiden kuormaa voidaan jakaa rinnakkaisille yksiköille, jolloin yksittäisen laitteen kuorma pienenee. Tämä kuitenkin vaatii erityisen kontrolloijan, joka kykenee hajauttamaan liikennettä yksiköille linjanopeudella. Tämä tutkimus keskittyy em. kontrolloijan järjestelmätopologioiden tutkimiseen sekä kontrolloijan toteuttamiseen ohjelmoitavalla piirillä, kuten kenttäohjelmoitava järjestelmäpiiri (eng. field programmable gate-array, FPGA). Kontrolloijasta tehtiin yksinkertainen toteutus 1-gigabitin Ethernet-verkkoihin sekä modulaarinen ja parametrisoitu toteutus 10-gigabitin Ethernet-verkkoihin. Toteutukset verifioitiin simuloimalla sekä käyttämällä syntetisoituvia testirakenteita. Toteutukset syntetisoitiin eri FPGA-piireille vaihtelemalla samalla myös toteutuksen parametrejä. Tehokkaimmat FPGA-piirit, kuten Altera Stratix -piirit, saavuttivat 10-gigabitin prosessointivaatimukset. Mittaustulokset osoittavat, että kontrollerin vasteaika ei poikkea tavallisesta verkkokytkimestä. Työn tulokset vahvistavat käsitystä, että ohjelmoitavat piirit soveltuvat hyvin verkkoliikenteen matalantason prosessointiin, missä vaaditaan ensisijaisesti suorituskykyä. Suunniteltu arkkitehtuuri on monipuolinen ja soveltuu joustavuutensa ansiosta muihin samantyyppiseen sovelluksiin.

Avainsanat: laitteistosuunnittelu, ohjelmoitavat piirit, FPGA, Ethernet, verkkoliikenteen prosessointi, hajautettu laskenta

Preface

The work described in this thesis was carried out from February to December 2008 at the Signal Processing Laboratory of Helsinki University of Technology. The work was funded by Stonesoft corporation.

I wish to take this opportunity to express my gratitude to my supervisor Professor Jorma Skyttä for this research project and for the guidance given throughout the work. I would also like to thank Marko Luoma for his valuable comments. The colleagues at the laboratory deserve my deepest acknowledgments for the warm-hearted welcome and the general friendliness that I received during this work.

Finally, I want to thank my loving parents and my brother for their support during the past years. An extra special thank you goes to my wife Anna for the unending love and support, and for having persistency to proof-read the text.

Espoo, 6th April 2009

Pekka Korpinen

Contents

List of Figures

List of Tables

Abbreviations

ASIC	Application Specific IC
ASIP	Application Specific Instruction Set
ALM	Adaptive Logic Module
ATM	Asynchronous Transfer Mode
CAM	Content Addressable Memory
CDC	Clock Domain Crossing
CDR	Clock and Data Recovery
CIOQ	Combined Input and Output Queuing
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DC	Direct Current
DDR	Double Data Rate
DLL	Delay-Locked Loop
DRAM	Dynamic RAM
DSP	Digital Signal Processor
DUT	Design Under Test
EOF	End of Frame
FCS	Frame Checksum Sequence
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
GMII	Gigabit MII
GPP	General Purpose Processor
HDL	Hardware Description Language
HOL	Head of Line
IC	Integrated Circuit
IDS	Intrusion Detection System
IFG	Inter-frame Gap
IP	Intellectual Property or Internet Protocol
IPG	Inter-packet Gap
IPS	Intrusion Prevention System
IQ	Input Queuing
LAB	Logic Array Block
LAG	Link Aggragation Group
LAN	Local Area Network
LE	Logic Element

LLC	Logical Link Control
LUT	Look-Up Table
MAC	Media Access Control
MDIO	Management Data Input/Output
MDI	Medium Dependent Interface
MII	Media Independent Interface
MLAB	Memory LAB
MTU	Maximum Transmission Unit
MUX	Multiplexer
NAT	Network Address Translation
NIC	Network Interface Card
NOP	No Operation
NP	Network Processor
NRE	Non-Recurring Engineering
OQ	Output Queuing
QDR	Quad Data Rate
PCB	Printed Circuit Board
PCS	Physical Coding Sublayer
PHY	Physical layer
PLL	Phase-Locked Loop
PMA	Physical Medium Attachment sublayer
PMD	Physical Medium Dependent sublayer
PRNG	Pseudorandom Number generator
PSL	Property Specific Language
RAM	Random Access Memory
RGMII	Reduced Gigabit MII
RISC	Reduced Instruction Set Computer
RS	Reconciliation Sublayer
RTL	Register Transfer Logic
RTT	Round-Trip Time
SDR	Single Data Rate
SDRAM	Synchronous DRAM
SERDES	Serializer/Deserializer
SFD	Start Frame Delimiter
SGMII	Serial Gigabit MII
SI	Signal Integrity
SiP	System-In-a-Package
SoC	System-On-a-Chip
SOF	Start of Frame
SRAM	Static RAM
STB	Synthesizable Testbench
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits
VLAN	Virtual LAN
VOQ	Virtual Output Queue
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
XAUI	10 Gigabit Attachment Unit Interface
XGMII	10 Gigabit MII
XGXS	XGMII Extender Sublayer

Chapter 1

Introduction

Cisco is estimating that the traffic on the Internet will double every two years through 2012 [Cisco, 2008]. The diversity of the Internet traffic is increasing as well because of new applications and novel usages. Companies and governments rely on the Internet while communicating and making business with other parties. Although there are obvious benefits in the Internet, the information thefts and network attacks are a well-known disadvantage.

The processing of network traffic is often required when protecting against malicious activities. Due to the high volume and the complexity of network traffic, the processing requires high-performance computation that is sometimes difficult to accomplish with a single device. By using parallel processing the computation load may be divided over multiple devices, therefore reducing the requirements of a single device. However, this requires a special device that can distribute the traffic to the parallel units and then unite the results from the units back. As the distribution happens in both transmission directions, the distributing device is different from a normal load-balancer, which divides the incoming requests among several servers.

This thesis concentrates on studying the problem of network traffic distribution, on finding suitable system topologies, and especially on the implementation aspects of the distributing device. This device is later referred to as the controller.

The aim of this thesis is to design a controller capable of efficiently distributing network traffic in 10-gigabit Ethernet. The controller performs relatively simple processing tasks but at very high-speed. Although network processors (NP) are widely used in general network processing, in this thesis the controller is implemented with reconfigurable hardware. A field-

programmable gate array (FPGA) device, which is based on a reconfigurable logic array, is able to perform tasks almost as fast as an integrated circuit (IC) while still offering programmability. Because the design is done at hardware level, the result will be more optimal, but on the other hand may require considerably more time than with NP.

The results of this thesis are a modular, highly configurable implementation of the controller and the practical knowledge of the general suitability of an FPGA device for distributing network traffic. The results also include an applicability analysis of the different FPGA devices today.

Chapter 2 gives an overview to Ethernet, TCP/IP, network processing, and information security, and describes the distribution problem in more detail. Chapter 3 concentrates on reconfigurable hardware and the implementation aspects of Ethernet. The controller's design is shown and examined in Chapter 4. The results and analysis of the design are presented in Chapter 5 and Chapter 6 summarizes the work.

Chapter 2

Networking Technology and Security Concepts

This chapter provides background information that is needed in this thesis. First, Ethernet technologies, specifically the speeds 1Gbps and 10Gbps, are covered including frame structures and duplexing modes. The implementation aspects, such as physical interfaces, are explained later in Chapter 3. The description of TCP/IP suite covers header structures, forwarding algorithms, and how the protocol operates with Ethernet. Next, networking devices, such as routers and switches, are explained to give the knowledge what type of devices are actually used. The next sections examine the processing tasks happening inside a networking device and the internal structures of switching fabrics. Then, the text gives a brief introduction to networking security while concentrating on the roles of different devices in implementing the security. The chapter ends with a section that introduces the problems in a specific case of high-speed network traffic processing and presents a solution to it. The solution is later implemented in Chapter 4.

In this work, the word “frame” is used when referring to a transmissible data object related to a certain communication technology, such as to Ethernet. The word “datagram” refers to a data object in higher protocol layers, such as Internet Protocol, which is not targeted for any specific transport technology. The word “packet” is a generic term referring to any data object with defined boundaries.

2.1 Ethernet

Ethernet is the most commonly used networking technology in today's local area networks (LAN). The original Ethernet was invented by Bob Metcalfe of Xerox Palo Alto Research Center in 1973 [Spurgeon, 2000]. The official Ethernet standard was first published in 1983 by the IEEE 802.3 committee with the title "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications" containing the 10Mbps operation. The latest Ethernet standard, IEEE 802.3-2005, published in 2005 defines operation speeds 10Mbps, 100Mbps, 1Gbps, and 10Gbps.

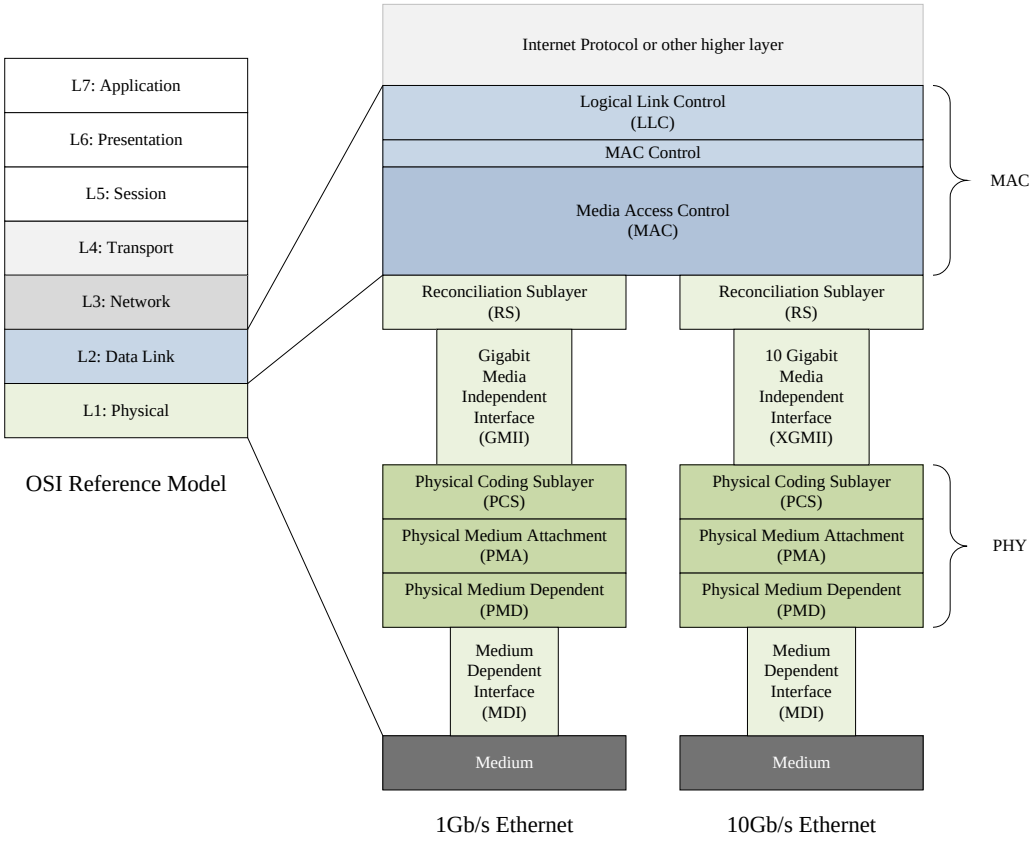


Figure 2.1: Division of Ethernet layers to Layer 1 and 2 of the OSI model.

The OSI (Open Systems Interconnection) model, defined by ISO 7498 standard [ITU-T94, 1994], provides a common layering structure for networking protocols. Ethernet operates on Layer 1 and 2 of the OSI reference model as depicted in Figure 2.1. The MAC sublayer, which provides frame transmission and reception services for the layers above, is discussed in this section. The details of the physical layer (PHY) are inspected later in 3.6. The upper layers of the OSI model are described in 2.2 while covering IP networking.

2.1.1 Half-duplex

The original Ethernet network used a shared medium, a coaxial cable, to which every station was connected to. Because the media was shared among all the connected stations only one station could send data at any given time, i.e. the communication was half-duplex. The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) media access control (MAC) protocol defined in the 802.3 standard allows the nodes, to share the same medium without a centralized control.

The next evolutionary step in Ethernet was to start using a dedicated medium instead of the shared coaxial cable. Here each station in the network has a dedicated media connection to a repeater. The repeater transmits the data received on one port to every other port, thus, emulating the behavior of the shared medium Ethernet. The use of dedicated medium will not help with collisions but installation and management of the network is easier than with coaxial cabling. In addition, the bandwidth is still shared among all the stations in the network.

A collision domain contains the stations contending for access to a shared network [Seifert, 1998]. A shared medium forms one collision domain, as does a network connected with repeaters. Because the collisions reduce the achieved bandwidth of the stations, the segmentation of collision domains is critical. A bridge device (see 2.3.3) can be placed to a network in order to split the collision domain into two separate domains. A bridge, operating on Data Link layer, can check the address of the received Ethernet frame and decide to which port it should be forwarded. This selective repeating reduces the amount of traffic, thus increasing the bandwidth and reducing the collision probability.

The half-duplex operation is specified for all speed classes, except for 10Gbps Ethernet. The standard specifies the half-duplex mode for 1Gbps Ethernet but it has not gained popularity. The half-duplex mode is not practical in gigabit speeds due to the round-trip timing requirements in collision detection. In 1Gbps Ethernet without the carrier extension the round-trip time would force the maximum cable length to be 20 meters (e.g. 1/10th of the 100BASE). Therefore the minimum frame had to be extended to 4096 bits from 512 bits in 1Gbps half-duplex mode. This of course affects the throughput when transferring small frames (under 512 bytes) since the extension consumes the available bandwidth.

2.1.2 Full-duplex

The full-duplex mode of Ethernet, introduced to the standard in 1997, allows simultaneous communication between two stations that are using a dedicated point-to-point link. This enables the traffic to flow in both directions doubling the capacity of the link. The full-duplex mode is available when the following requirements are met [IEEE802.3, 2005]:

1. The media must support bi-directional communication.
2. There are exactly two stations connected to the link.
3. Both stations support the full-duplex mode of operation.

The most common twisted-pair and fiber optic links fulfill the first requirement. The second requirement is achieved when using two computers in point-to-point fashion or when using a bridge device with a single device on each port of the bridge. Due to the second requirement, the CSMA/CD protocol is unnecessary as there are no collisions. Therefore, the transmissions will not suffer from collisions and the actual throughput can attain the theoretical bandwidth. In addition, because the CSMA/CD protocol in not needed, the full-duplex-only MAC is considerably smaller and easier to implement.

2.1.3 Media Access Control

Media Access Control (MAC) is a medium-independent sublayer on top of the Physical Layer. The two main functions of the sublayer are a) data encapsulation and b) media access management [IEEE802.3, 2005]. The first item includes framing, handling of source and destination addresses, and error detection. The latter function manages collision avoidance and handling, which are related to the half-duplex mode, and physical layer congestion (e.g. PHY's TX not ready), which is necessary also in the full-duplex mode.

Preamble 7 bytes	SFD 1 byte	Destination 6 bytes	Source 6 bytes	Length/Type 2 bytes	Payload 46-1500 bytes	FCS 4 bytes

Figure 2.2: Format of a MAC frame.

The MAC frame format is presented in Figure 2.2. The preamble is used for synchronizing the receiver's timing. Each preamble byte has a value of $10101010b$. The Start Frame Delimiter (SFD) field indicates the start of a frame. Its value is $10101011b$. The destination field specifies the address where the frame is destined. The address can be unicast, multicast, or broadcast

address. The source field identifies the station sending the frame. The length/type field has two separate meanings. If the field's value is greater than or equal to $0x6000$, then the field contains the Ethernet Type, which identifies the used protocol, else the field contains the length of the frame. The payload is the actual transmitted data padded to the minimum length (see below).

The Frame Check Sequence (FCS) field contains a 32-bit cyclic redundancy code (CRC) value that is used to verify that the frame has been received correctly. The CRC is calculated over all fields in the MAC frame except preamble, SFD, and FCS. The error detection is based on dividing the data with a polynomial in modulo-2 and using the remainder as the checksum value [Peterson and Brown, 1961]. The Ethernet specification defines the polynomial to be $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

The order of bits in the MAC frame is as follows (except the FCS): the least significant bit and the most significant byte is transmitted first (i.e. words are transmitted in big-endian format). The FCS is transmitted in the order $x^{31}, x^{30}, ..., x^1, x^0$.

If the EType field contains the frame's length, Subnetwork Access Protocol (SNAP) is used to identify the frame's type. The payload is preceded by a 802.2 LLC header with constant values (DSAP=0xaa, SSAP=0xaa, Control=0x03), a 802.2 SNAP block with a constant value (Org code=00-00-00), and the actual type of the encapsulated protocol. The resulting frame is commonly called as IEEE802.2/802.3 Encapsulation. Other uses of LLC and SNAP and their details are out of the scope of this thesis.

The minimum length of the frame is dictated by the collision detection algorithm of the half-duplex operation. The minimum length is 512 bits, or 64 bytes, excluding preamble and SFD (+ the frame extension in 1Gbps half-duplex). In other words, the minimum size of the payload is 46 bytes. The maximum size of the normal frame is 1518 bytes (1500 bytes payload). Frames longer than the specified maximum size, called Jumbo frames, are supported by most of the network vendors, but the 802.3 standard does not recognize the Jumbo frames because it would break the interoperability with other 802 protocols, mainly 802.5 Token Ring and 802.11 Wireless LAN.

The specification defines the minimum spacing between two consecutive MAC frames to be 96 bit times. This constant has many name variations, including Interframe Gap (IFG), Inter-Packet Gap (IPG), and Interframe spacing. The term IFG is used in this work.

VLAN tagged frame

Virtual LANs (VLANs) are used in segmenting real LANs into virtual groups (see 2.3.3). The frames belonging to a certain VLAN are identified by a tag inside the MAC frame. The IEEE 802.1Q VLAN standard specifies an extension to the standard MAC frame that provides vendor-independent protocol for implementing VLANs [IEEE802.1Q, 2005]. The Ethernet standard contains this extended frame format [IEEE802.3, 2005].

The VLAN type and tag fields are inserted before the normal Length/Type field of the MAC frame. Beside the insertion of the VLAN type and tag fields, the original MAC frame remains untouched. The VLAN tagging enlarges the maximum frame length to 1522 bytes. The VLAN tagged MAC frame is shown in Figure 2.3.

Preamble 7 bytes	SFD 1 byte	Destination 6 bytes	Source 6 bytes	VLAN Type 2 bytes	VLAN Tag 2 bytes	Length/Type 2 bytes	Payload 46-1500 bytes	FCS 4 bytes

Figure 2.3: Format of a VLAN tagged MAC frame.

The VLAN Type field contains a reserved Ethernet Type value identifying that this frame contains a VLAN tag. The value $0x8100$ is reserved for C-VLAN (Customer VLAN) usage, thus separating it from other VLAN types such as S-VLAN (Service VLAN) [IEEE802.1ad, 2006]. The VLAN Tag field has a 3-bit user priority field, a 1-bit canonical format indicator (CFI), and a 12-bit VLAN identifier (VID). A frame with VID 0 is not a VLAN frame but uses the VLAN tag for priority purpose. The value $0xfff$ is reserved for implementation use.

2.1.4 Ethernet flow control

In the half-duplex mode, the receiving station can abuse the CSMA/CD protocol to achieve a kind of flow control. It may force collisions with incoming frames or send carrier signal to make the channel to look busy. The other stations will see this and are forced to retry or defer their transmissions. In the full-duplex mode this is not possible because of the dedicated bi-directional links.

The real Ethernet flow control (802.3x) is implemented in the optional MAC Control sublayer with the MAC Control PAUSE operation [IEEE802.3, 2005, Clause 31,Annex 31B]. The operation of the flow control is relatively simple. A station monitors the used space counter of the receive buffer. When the counter exceeds a certain limit, the station transmits a Pause frame.

The station on the other end of the link will receive and process the Pause frame and stop transmitting. The Pause frame contains a parameter that tells the station for how long to pause the transmission. The pause time is measured in units of pause_quanta, equal to 512 bit times of the line speed. The station can cancel the already issued pause request by sending a new Pause frame with a zero value. Thereby, the flow-control can be used in simple STOP-START manner or to specify an estimate how long it takes the congestion to clear out. Note that the paused station is inhibited from sending data frames but is allowed to send control frames (i.e. Pause frames).

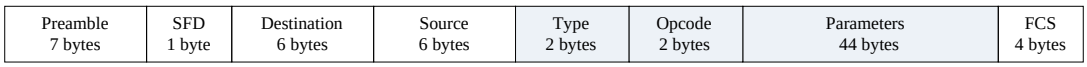


Figure 2.4: Format of a MAC control frame.

The Pause frame structure follows the standard minimum length MAC frame as depicted in Figure 2.4. The Destination field has the MAC address `01-80-c2-00-00-01`, which is reserved for the Pause function. The Source field has the station's own address as normally. The Type field contains the value $0x8808$ identifying this as the MAC control frame. Only one value is currently specified for the Opcode field, $0x0001$, the Pause frame opcode. The Pause frame has one 16-bit parameter, the pause time, which is placed on the beginning of the Parameters field. The rest of the parameter field is set to zero.

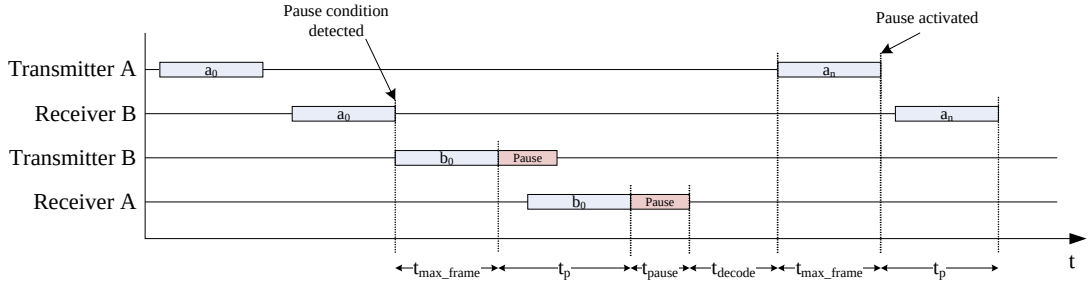


Figure 2.5: Worst-case delay in Pause operation.

The standard defines how fast the station must react to Pause frame. On 100Mbps or slower links the station shall not begin to transmit a new frame more than 1 pause_quanta after the successful reception of a Pause frame. At 1000Mbps and 10Gbps the time is 2 and 64 pause_-quantas, respectively. The buffering requirements and the pause signaling thresholds can be estimated by calculating the worst-case delay. Let's assume the link has stations A and B. B detects that the receive buffers are close to overflow after receiving the frame a_0 from A and issues a Pause frame request to be sent (Figure 2.5). The transmitter of B begins to send a max-

imum length frame and can not be preempted, thus the Pause frame is delayed by t_{max_frame} (note, includes the IFG period). The receiver of A sees the beginning of the Pause frame after propagation delay t_p and receives it after t_{pause}. A must react to the Pause frame in t_{decode}. It may be that A has just begun to transmit a frame at the same time when the Pause frame was decoded, so the pause comes in effect after t_{max_frame}. B will see this after the propagation delay t_p. So, the total time seen by B is given by Equation 2.1.

$$t_{total} = 2t_p + 2t_{max\_frame} + t_{pause} + t_{decode} \tag{2.1}$$

By multiplying Equation 2.1 with the data rate R_b, given in bits per second, the equation can be translated from time units into bits. The following equation tells how many bits the transmitter of A can sent before obeying the Pause request.

$$b_{total} = \underbrace{2t_pR_b}_{b_m} + \underbrace{2b_{max\_frame} + b_{pause} + b_{decode}}_{b_c} \tag{2.2}$$

The first part of Equation 2.2, b_m, represents the buffering requirements due to the medium and it depends on the electrical properties of the medium (i.e. propagation velocity of the signal) and the length of the link. The second part, b_c is constant for the given Ethernet speed.

The propagation delay t_p can be calculated using the following equation

$$t_p = \frac{d_{link}}{v_p} = \frac{d_{link}\sqrt{\epsilon_r\mu_r}}{c} \tag{2.3}$$

where d_{link} is the length of the link, v_p is the velocity of propagation equal to $\frac{c}{\sqrt{\epsilon_r\mu_r}}$ where c is the speed of light, and ϵ_r and μ_r are relative permittivity and permeability of the medium. For unshielded twisted-pair category 5 cabling v_p is about 0.6c and for fiber 0.67c.

Typically the receiver removes the Preamble and SFD fields as well as the FCS field before putting the frame to the receive buffer. This means that for every received frame 96 bits are discarded. Also, there has to be at least 96 idle bits (IFG) between consecutive frames. Compared to the maximum frame length, 12240 bits, these are negligible and thus, the maximum buffering requirement can be approximated by simply using the total bit count b_{total}.

The calculated buffering requirements for different Ethernet links are summarized in Table 2.1. The "(none)" row contains only the b_c part of Equation 2.2. 10GbE requires twice as large buffers as 1GbE on link lengths below 100m. On long-range links the ratio increases as the shorter bit-width begins to dominate the b_m part of Equation 2.2. Note that these figures only represent the buffering space dedicated to the pause operation to function properly, they do not contain the space requirements of normal operation.

Table 2.1: Minimum buffer requirements for pause operation.

	Buffer size (kB)	
Medium	1GbE	10GbE
(none)	3.2	7.1
10m Cat5	3.2	7.2
100m Cat5	3.4	8.4
100m Fiber	3.3	8.3
2km Fiber	5.6	31.5
5km Fiber	9.3	68.1

2.2 TCP/IP suite

The development of the Internet began in ARPAnet, the world's first wide-area packet-switching network, which was developed in 1968 by the Advanced Research Projects Agency (ARPA) of the United States Department of Defense (DoD) [Hall, 2000]. ARPAnet's design provided two important novelties: a vendor-independent networking standard that could interconnect computers with proprietary networking technology, and a built-in fault-tolerance so that a loss of a network node would have only a local effect. Other network providers started to connect to ARPAnet sites and then to each other forming now a network of networks, the Internet.

The Internet protocol suite, called TCP/IP suite, is a collection of several protocols relying on each other [Comer, 1988]. Although the TCP/IP suite layering does not exactly follow the OSI reference model, it can be loosely mapped to the OSI model as shown in Figure 2.6. Here the original Link layer of [Braden, 1989] is splitted into Data link and Physical layers despite the fact that the Internet protocol suite is link agnostic. This helps to relate the Internet protocol to the earlier discussed Ethernet domain, which implements the OSI Layers 1 and 2. The Internet Protocol (IP) works on Network Layer (L3) of the OSI reference model and the transport protocols (TCP and UDP) on Transport Layer (L4). OSI Layers 5 through 7 are compacted into a single Application layer that covers everything above Layer 4.

2.2.1 Internet Protocol

The Internet Protocol (IP) offers connectionless packet delivery service to the higher layers and hides the used network technology (e.g. Ethernet). It provides a way to transmit data blocks, called (IP) datagrams, from source node to destination node over "an interconnected system of networks". However, the delivery of a datagram or the ordering of received datagrams are not guaranteed. If necessary, the upper layers may implement the missing features (e.g. reordering or reliable delivery) within their domain.

Currently there are two versions of Internet Protocol in active use. The first widely deployed version, IPv4, was published in [Postel, 1981a]. The standard was developed to be used in much smaller scale than it is today, so its features (mainly the limited address space) are not designed for such a massive network as the Internet is today. The Network Addressing Translation (NAT) technology [Srisuresh and Egevang, 2001] has given some more time for the 1980s' protocol but the end is closing in as the address space is becoming exhausted. The discussion for the successor of IPv4, began already in the early 1990s under the title "IP Next Generation Protocol" (IPng). The IPng was published in [Deering and Hinden, 1995] with title "Internet Protocol version 6" (IPv6) and later revised in [Deering and Hinden, 1998].

IP datagram's path from a source node to a destination node may involve multiple intermediate nodes. After a reception of a datagram each node decides where to forward the datagram next. The decision is based on a forwarding table. An entry in the table gives the address of the next hop node. The lookup may be for a specific destination address, or it may be for a destination network. If there are no matching entries, a default gateway is selected. This means that there are two basic forwarding operations. If the destination address is located on one of the connected networks then the IP datagram is sent directly to the final destination. This is called *direct forwarding*. Otherwise, the datagram is sent to a gateway node, which hopefully sends the datagram forward. This is called *indirect forwarding*.

OSI model		Internet Protocol		Example
Layer 7	Application	L5	Application	FTP, Telnet, HTTP
Layer 6	Presentation			
Layer 5	Session			
Layer 4	Transport	L4	Transport	TCP, UDP
Layer 3	Network	L3	Internet	IP
Layer 2	Data link	L2	Data link	MAC, ARP
Layer 1	Physical	L1	Physical	10GBASE-T

Figure 2.6: Internet protocol layers compared to the OSI reference model.

The forwarding table is created by the routing process that calculates optimal paths using routing algorithms and routing tables. A core network router has typically multiple routes available for the given destination address. In these cases the routing algorithm tries to select the best match among all the possible routes. This decision may depend on multiple metrics, such as hop count, bandwidth, load, delay, reliability, and cost [Doyle, 1998].

The datagram format of IPv4 is shown in Figure 2.7. The most relevant fields for this thesis are Version, Length, Protocol, and IP addresses. The Version field contains decimal number 4 identifying this as IPv4. The Length contains the length of the IP header in 32-bit words. The minimum valid value is 5, therefore the IP header's length can range from 20 (typical) to 60 bytes. The Protocol field identifies what protocol the data follows (e.g. TCP) and is used to deliver the data to the correct recipient at the destination. The Source and Destination IP address fields contain the IP address of the origin and the ultimate destination of the datagram, respectively. The data field, containing the user's (e.g. TCP layer) data, begins after the header fields and is always located at the 32-bit boundary.

The datagram used in IPv6 is shown in Figure 2.8. The basic header is simpler than in IPv4 but can be extended using header extensions. The standard states that a full implementation of IPv6 must have support for the following six extensions: Hop-by-Hop Options, Routing, Fragment, Destination Options, Authentication, and Encapsulating Security Payload. The Hop-by-Hop extension is the only extension that every node along the path must examine and process, other extensions are for the destination node(s) only. The Version field has a decimal value 6 separating it from an IPv4 datagram. The Next header field functions similarly to the Protocol field of IPv4. It identifies the next header (e.g. TCP payload or Fragment Extension) that follows the IPv6 header. Compared to IPv4, the IP address fields are extended to 128-bit.

2.2.2 TCP and UDP

User Datagram Protocol (UDP) is a connectionless protocol offering transmission of short messages, UDP datagrams. It handles message boundaries but does not guarantee reliability or ordering of the messages. The UDP datagram format is depicted in Figure 2.9 [Postel, 1980b]. UDP uses the Destination address and the Destination port information when demultiplexing the incoming datagram to the recipient. It is identified by the value 17 in the IP header's Protocol field.

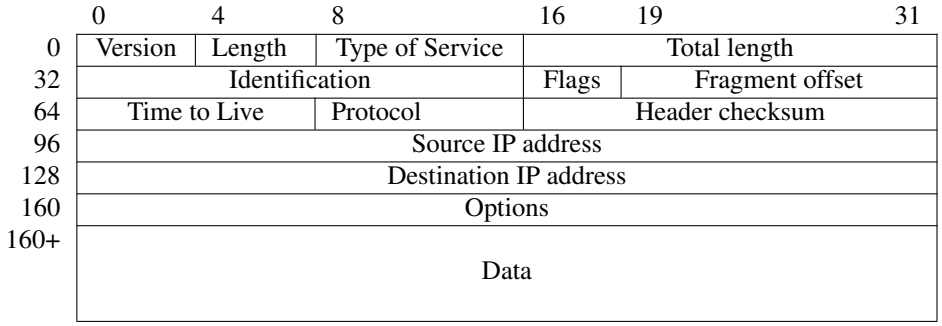


Figure 2.7: Format of an Internet datagram version 4 (IPv4).

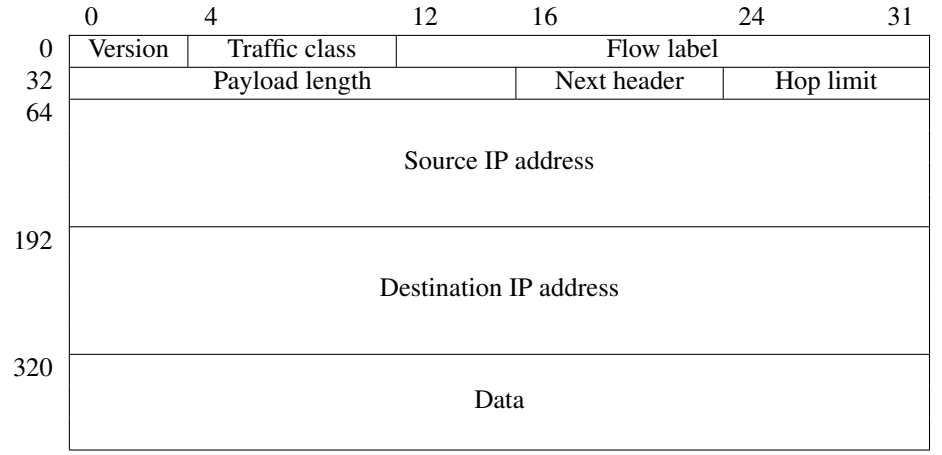


Figure 2.8: Format of an Internet datagram version 6 (IPv6).

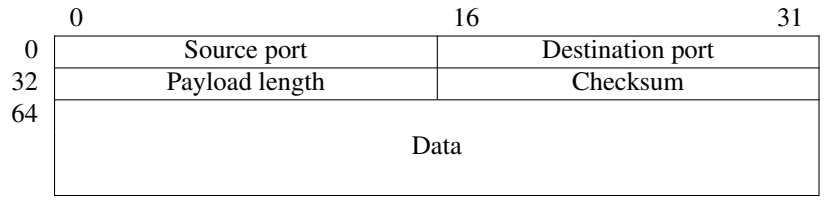


Figure 2.9: Format of a UDP datagram.

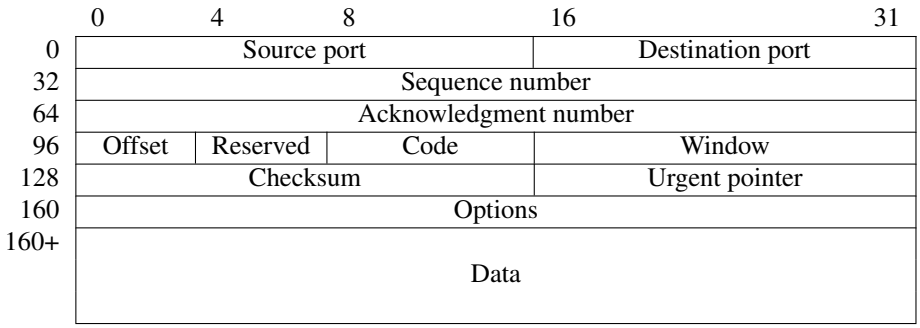


Figure 2.10: Format of a TCP segment.

Transmission Control Protocol (TCP) provides a reliable socket-based, full-duplex byte-stream communication. It is defined in [Postel, 1981b; 1980a; Cerf et al., 1974]. The protocol is connection-oriented and uses a three-way handshake to establish a connection. First, the client sends a SYN (synchronize) to the server. The server replies with a SYN-ACK (synchronize-acknowledgment). Finally, the client finishes the procedure with an ACK. The flow control is handled by using a concept known as a sliding window. The individual packets of TCP communication are called segments. The format of a TCP segment is shown in Figure 2.10. It is identified by the value 6 in the IP header's Protocol field. A TCP connection is identified by a five tuple (Protocol, Source address, Source port, Destination address, Destination port).

2.2.3 IP over Ethernet

IP, being a network-technology independent protocol, can operate in Ethernet networks. The IP datagram is transmitted in the payload of a standard Ethernet frame and is identified by the Type field of value $0x0800$ [Hornig, 1984]. Figure 2.11 shows an example where user data is sent over a TCP connection. User data is first segmented into two TCP segments. This is done in order to fit the data into a transmissable unit, which is limited by the MTU (Maximum Transmission Unit) of the transport technology (i.e. Ethernet). Each TCP segment is then encapsulated into an IP datagram that is further placed into an Ethernet frame with the correct Type field. The depicted process is applicable also to UDP datagrams. The operation on the receiving node is alike. The IP layer gets an IP datagram and based on the IP header gives the payload to the upper layer. The upper layer, here TCP, process the TCP header and forwards the data to the application. Because TCP is message agnostic, the application may receive the data in multiple parts although it was sent from the application in a single request.

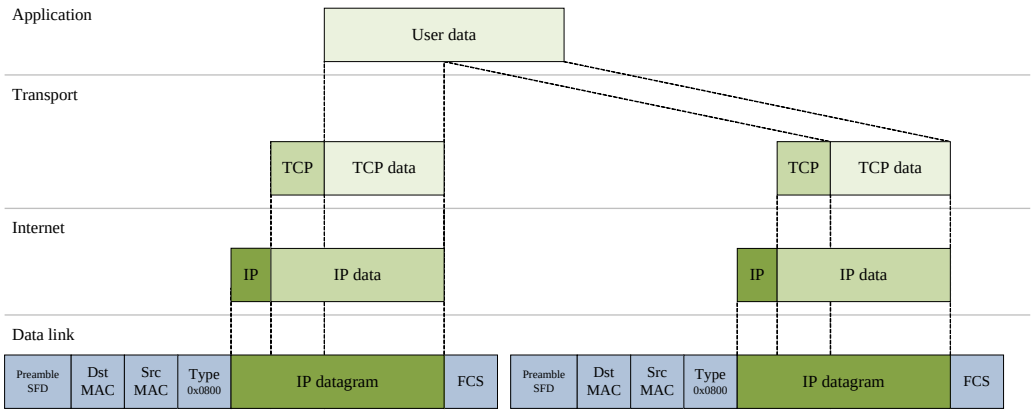


Figure 2.11: Data encapsulation and segmentation in TCP/IP suite operating in Ethernet.

2.3 Network devices

Network devices may be categorized to four fundamental categories based on the level they operate on the OSI model. A hub device operates on Layer 1, a bridge device on Layer 2, a router on Layer 3, and a host on all layers.

2.3.1 Host

A network host is most commonly a computer that has a network interface card (NIC). The network interface card implements the OSI Layers 1 and 2 (e.g. physical and MAC layer of Ethernet) and the computer's operating system implements the other layers (e.g. TCP/IP stack). A host can also act as a router (or, a gateway) if it is configured to do so.

Normally, a host only wants to receive frames that are destined to it (including broadcast frames), and configures the NIC to filter other frames. The NIC can be also set to *promiscuous mode* where all frames are accepted and passed to the operating system. This is mostly used while debugging the network.

2.3.2 Hub

A network hub, sometimes called a repeater, is a networking device operating on OSI Layer 1 and having multiple Ethernet ports connected to different network segments. The device acts as a simple repeater, receiving signals from one port and transmitting them to all the other ports. The signal is regenerated but the conveyed data is not interpreted. Consequently, all the stations in the formed multi-segment network can communicate with each other, but only one station can transmit at a time. In other words, the connected stations still belong to the same single collision domain. The hubs are not anymore used due to the availability of low-cost network switches.

2.3.3 Bridge and switch

A bridge is used to connect networks together, and a multi-port bridge is commonly known as a switch. The major difference between a hub and a bridge is that the bridge operates on OSI Layer 2. Instead of blindly repeating every frame, the bridge checks whether the incoming

frame should be forwarded to a port or not. The most important benefit is that the bridge device divides the physical links into separate collision domains and therefore supports full-duplex communication (see 2.1.2). A bridge may also support bridging between different types of networks, for example, between Ethernet and Wireless LAN (IEEE 802.11). The bridging operations on all IEEE 802 network types are specified in [IEEE802.1D, 2004].

The bridge may have static forwarding rules or it may learn the rules dynamically from the incoming frames using a learning algorithm. In the learning algorithm, the source MAC address of a received frame is put into a forwarding table with the information about the ingress port (i.e. source port). If the destination MAC address is found in the forwarding table, the frame can be sent to the associated port. Otherwise, the frame is sent to all the other ports. This action is referred as *flooding* and it resembles the operation of a hub. A frame is also flooded in the case of a broadcast frame.

The virtual LAN concept [IEEE802.1Q, 2005] extends the forwarding algorithm by including the VLAN ID to the forwarding decision. The forwarding happens only if the egress port belongs to the VLAN group of the incoming frame. If the incoming frame is not VLAN tagged (see 2.1.3), a configurable port VLAN ID (PVID), which defaults to 1, is applied to it. By default, the frame is untagged (i.e. VLAN tag is removed) by the egress port. In summary, the VLAN technology increases network's performance by constraining broadcast traffic and frame flooding to inside a VLAN group, and adds security by isolating traffic streams.

Link Aggregation Group (LAG) is formed when one or more links are aggregated together [IEEE802.1AX, 2008]. The use of multiple physical links results in increased availability and bandwidth but also might cause frame reordering. To prevent this, a distribution algorithm is used to decide which link of the LAG is used for sending a frame. Basically all frames belonging to a same conversation should be sent over same link. The criteria for a conversation varies between LAG implementations. A very simple algorithm might use only the MAC addresses resulting in poor load balancing characteristics.

2.3.4 Router

A network router is similar to a switch but operates in the TCP/IP domain, i.e. on OSI Layers 3 and 4 [Baker, 1995]. Consequently, these devices are also referred as L3 or L4 switches. The clear distinction between the switch and the router is that with routers the forwarding decision

is based on the IP datagrams, not on Ethernet frames. A router maintains a routing table that is used when deciding where to forward the incoming datagrams. It must be noted that the routers are not tied to Ethernet technology but may operate also with other network technologies, such as in ATM.

Firewalls and load balancers are variations of the generic router. They both forward packets but the next hop selection depends on, beside the normal routing table, application specific rules and knowledge. A firewall inspects the packets up to OSI Layer 7 and then decides, based on the configured firewall rules, whether the packet should be discarded or forwarded. A load balancer tries to divide the work load between multiple servers by routing the incoming traffic differently from connection-to-connection. The destination selection is based on various algorithms, such as round-robin, weighted round-robin, least connections, least load, or fastest respond. In general, both the firewall and the load balancer maintain state information about the active connections (e.g. TCP), hence requiring more processing power and memory than an ordinary L3 router.

2.4 Network traffic processing

When a frame is received or transmitted it undergoes various processing steps. These phases are commonly divided into two groups as illustrated in Figure 2.12 [Comer, 2004]. The division between ingress and egress processing is not tight but rather a way to understand the overall picture.

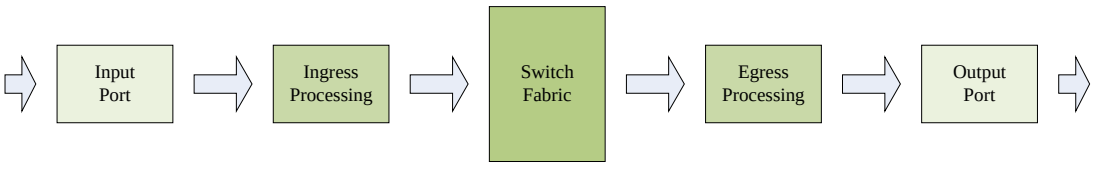


Figure 2.12: Ingress and egress processing phases.

This section first concentrates on the different processing phases. The fundamental processing architectures are described before inspecting various implementation options. The section ends with an analysis of the required processing speeds.

2.4.1 Packet processing phases

The processing can follow one of the two main design philosophies [Giladi, 2008]. In *store-and-forward* designs the incoming packet is stored entirely to a temporary buffer before the packet processing begins. This phasing allows more complex packet processing steps to be performed but also increases the latency and the buffering requirements. In *cut-through* designs the processing begins immediately when the first bits of the packet is received and continues as more data is received. This naturally reduces the latency and the buffering requirements but also complicates the design. The worst side-effect of the cut-through is that it may increase the network traffic by passing erroneous packets through it. This is possible because the frame's verification results are available only after it has been received completely.

The packet processing tasks can be divided into few categories that are described next [Comer, 2004; Paulin et al., 2003; Shah, 2001; Giladi, 2008]. The names of the phases are typed in bold. Except for the control processing phase, all of the phases are operated at the wire-speed.

Header parsing phase analyzes the packet and extracts the relevant header fields. These headers are then passed to other phases, such as the phase described next.

Packet classification phase maps the packet to one of the statically or dynamically configured flows that are specified by a set of rules [Gupta and McKeown, 2001]. The flow is a logical path that similar packets should take. For example, an IP datagram with a TCP segment could be assigned to Flow-1, and others to Flow-2. Naturally the classification is always based on the information in the current packet. If the classification depends also on the previous packets then its called stateful classification. Compared to stateless classification, this requires additional memory resources.

Forwarding phase decides where to transmit the packet. The decision is based on the assigned flow. This includes both output port selection and the calculation of the next hop.

Search and lookup phase is a common operation used by almost all other phases. This is the most critical point in the IP processing chain, and can utilize many structures and algorithms [Chao, 2002; Ruiz-Sanchez et al., 2001]. In addition to exact match, the search can provide results based on pattern or best match [Waldvogel, 2000].

Data manipulation phase modifies the contents of the packet, for instance when adding VLAN tags or decrementing TTL/Hop count.

Error detection and data encryption phase uses various algorithms to detect errors, prevent errors, or encrypt data. The phase is on the other hand computationally the most demanding phase, but also the easiest to accelerate because of the relatively static algorithm selection (e.g. CRC-32 or IPSec).

Queue management phase administers the input and output queues of the data path as well as any temporary buffers. This includes memory addressing and buffer state monitoring, as well as quality of service (QoS) type operations.

Control processing phase combines a vast amount of tasks that do not need to or can not be run at wire-speed, referred usually as control-plane tasks. These include traffic measurements, system statistics, exception handling, and overall management of the system. The exceptions are typically route related where the lookup phase is unable to determine the forwarding address due to the missing information in the look-up memory.

2.4.2 Processing architectures

Three generic processing architectures are depicted in Figure 2.13. They all implement some functions f(), g(), and h(). The first architecture implements the functions with a single block. The second architecture separates the functions and implements them in series. When the function A finishes, the function B begins to process and the function A is free to begin a next job. This is called pipelining and it resembles an assembly line in a factory. The third architecture has multiple units operating in parallel, thus offering multiple packets to be processed at the same time. The units in the parallel architecture can be further pipelined as well, resulting in a hybrid pipeline-parallel architecture. Usually the used architecture is not a direct implementation of the aforementioned types but rather a combination of them.

The architectures in Figure 2.13 differ on bandwidth (i.e. operating speed), cost, and latency [Culler et al., 1999]. Lets begin by saying that the architecture A must operate at wire-speed. The architecture B with the pipelined design also operates at wire-speed but the individual parts can take the same time as the functions f(), g(), and h() in total in the architecture A. That is, each function has more time to finish the task thus making the design easier and less complex. The disadvantage of pipelining is the increased latency and the buffering need between the stages. The pipeline exceptions (e.g. hazards and hazard related stalls) [Patterson and Hennessy, 1998] may degrade the performance unless solved in the early design phase.

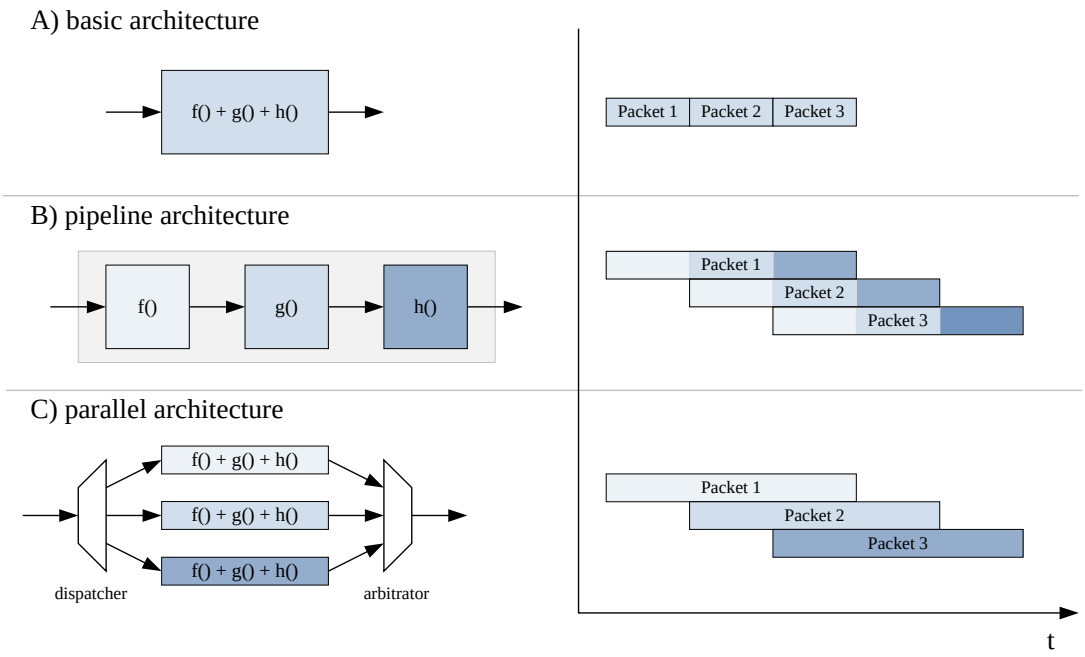


Figure 2.13: Three common processing architectures.

The units of the parallel architecture in Figure 2.13b are running at the third of the wire-speed as the incoming data is split equally to the three individual worker units. However, the dispatcher and the arbitrator operate at the wire-speed. The parallel architecture shares the same latency issue the pipelined design had but costs more (e.g. in terms of logic area) as the functions are replicated. Also, the parallelism requires additional logic to keep the parallel running parts aware of each other. The concurrency issues, such as race conditions, starvations, and locking, must be properly handled.

There is a limit, known as Amdahl's law, how far an algorithm can be parallellized [Amdahl, 1967]. That is, the speedup gained from parallelism saturates at some point depending on how large a fraction of the algorithm is actually parallelizable. The law is given as

$$\text{Speedup} = \frac{1}{(1-f) + \frac{f}{n}} \tag{2.4}$$

where f is the percentage of the algorithm that is parallelized and n is the amount of parallelized units. If f is small, then the parallelism is not an effective way to improve performance. If the algorithm is suitable for parallel architecture (e.g. f is near 100%), the speedup approaches n. An extension to Equation 2.4, which takes into an account the cost of parallelism, can be found in [Hill and Marty, 2008].

2.4.3 Implementation options

Different hardware options offer a wide range of characteristics and naturally not all of them are desirable. The *performance* is on top of the feature list because the wire-speed operation is essential in network processing. The *flexibility* offers the possibility to make adjustments to the protocols and the applications. The flexibility is achieved by using programmable cores (e.g. processors) with software or programmable hardware (e.g. FPGAs). Unfortunately the flexibility and the performance are often opposing forces. The *development time* is crucial as it affects the time-to-market figure. The cost aspect is the number one limiting factor. The *development cost* determines how much the designing of the hardware costs. Off-the-shelf products, such as intellectual property (IP) blocks, are used in reducing the development time. However, this increases the *integration cost*. The *power efficiency* is becoming important as the consumers are more environmentally aware.

Application specific integrated circuit (ASIC) is a fully custom integrated circuit (IC) that offers the best performance. The flexibility is extensive until the end of the lengthy design phase. Once produced, an ASIC will do what it was designed to do and nothing more. The designing of an ASIC is costly and requires a lot of working hours but on the other hand the manufacturing is relatively cheap. The non-recurring engineering (NRE) costs dominate until a break-even point, after which the unit cost decreases rapidly. Thus, the ASIC technology is not feasible for low-volume products (e.g. under 1000 units). Although an ASIC is targeted to a specific problem, the design is still more or less a compromise to maximize the volumes.

General purpose processors (GPP) are ICs containing a software programmable processing core. This offers flexibility as the processing can be altered by simply changing the software. The GPPs are mass-products and thus are likely to be very low cost. In addition, mass-production means that the software tools and product support are at a good level. In addition, GPPs, such as PowerPCs, are familiar technologies to software developers hence reducing the time-to-market. The clear downside of GPP is the performance because the processor and the instruction set are designed to be as generic as possible and not targeted to any specific task.

Co-processors are application specific accelerators. A GPP may use a co-processor to handle a specific task, for example the checksum calculation of a packet. This leaves the GPP free to do other tasks while the co-processor is running (cf. parallelism). Due to the advancements in IC technology the co-processors are being replaced by System on a Chip solutions.

Application specific instruction processor (ASIP) are programmable processors targeted to a specific domain. The ASIPs try to combine the best features of ASICs and GPPs by using a domain specific hardware architecture to provide the performance and a domain specific instruction set to offer flexibility. Digital signal processor (DSP) is the most known ASIP. It has fast data paths and calculation units to target the demands of real-time signal processing. DSPs are not very suitable for network processing, though, as the domain requires other features than just raw computing. Network processors (NP) are ASIPs targeted specifically to the network domain [Shah, 2001; Comer, 2004; Giladi, 2008]. They are relatively new but are already rapidly gaining success in network products. The ASIPs are harder to program due to their unique architecture and also their cost is higher than GPPs as they are not such a mass-product.

Reconfigurable hardware offers flexibility by introducing hardware that can be reprogrammed anytime. The design flow resembles ASIC design but instead of manufacturing custom ICs, the technology uses off-the-shelf Field Programmable Gate Array (FPGA) devices. This means that while the design time and cost (NRE) are lowered, the unit cost does not follow the economics of scale as the ASICs do. The performance, the hardware flexibility, and the power efficiency are worse than in ASICs but the gap is constantly narrowing. The comparison between 90nm ASIC and FPGA technology [Kuon and Rose, 2007] shows that FPGA is about three times slower than ASIC and consumes 12 times more power. Nevertheless, the offered programmability and the smaller NRE usually outweigh these downsides. Some FPGA vendors provide a migration path from an existing FPGA design to a hard-wired IC (e.g. Altera's HardCopy or Xilinx' EasyPath) in order to benefit from the ASIC-like unit cost. The reconfigurable hardware is a valid alternative to low-volume products that have distinct requirements for architecture or computation [Sezer et al., 2005]. FPGAs are widely used as a prototyping platforms for ASIC designs. In 2002, over 70% of the programmable logic devices were being used in networking and communications applications [Bielby, 2002].

System on a Chip (SoC) is an IC that integrates separate functions or even ICs to a single chip. For example, a GPP with a couple of co-processors, a few network ports, and even a reconfigurable fabric, might be implemented in a chip. Of course, being a custom IC, this shares the same cost policy as the ASICs. For mass-products this offers savings in design and manufacturing costs as well as in power efficiency. Another similar technology, called **System in a Package** (SIP), bonds separate ICs into one physical package. The cost is slightly lower as this utilizes existing ICs (only the packaging process is custom) while the performance is similar to discrete ICs.

There is no single choice for a network domain due to the wide ranging network products, requirements, and processing tasks. That said, there are still some guidelines. The performance of ASIC is typically required in the high-end high-throughput parts as the switch fabric where there is no need for flexibility. NPs are commonly used to implement the ingress and egress processing, often in parallel configuration. They are most probably accompanied by a GPP to handle the control-plane processing. The reconfigurable hardware has its place in low-volume products that require hardware-like performance and reprogrammability or unique features that the mainstream ICs do nt have.

2.4.4 Processing rate requirements

The requirements in packet processing are two-fold. First of, the processing units must be capable of processing incoming data at wire-speed, i.e. at the theoretical maximum transmission speed of the given network technology. The incoming data rate (given in bits per second, bps) affects the buffering requirements directly. On the other hand, some of the protocol processing tasks (e.g. routing table look-up) have a fixed cost per packet. This means that the maximum incoming packet rate (given in packets per second, pps) is also an import design criteria. The packet rate sets the worst case limit for look-up table bandwidths and for computational processing.

The packet and data rates of small and large packets in Ethernet are presented in Table 2.2. Compared to the large packets, there are 18 times more packets arriving per second when using small packets. On the other hand, the small packets contain 27% less data than the large packets. The calculations include all the bits of the MAC frame as well as the IFG. The data rate excludes the MAC preamble, SFD, and the FCS fields as they are not needed by the processing units (FCS is checked by the network interface). To conclude, in Ethernet and other network technologies with variable length frames, the maximum packet rate is achieved when using the smallest packets and the maximum data rate when using the largest packets.

Table 2.2: Maximum packet and data rates when using the smallest and the largest frame lengths.

	Small packets		Large packets	
	kpps	Mbps	kbps	Mbps
100BASE	148.8	71.4	8.1	98.4
1000BASE	1488.1	741.3	81.3	984.4
10GBASE	14881.0	7142.8	812.7	9844.0

2.5 Switching fabrics

A switching fabric forwards network packets from an input port to an output port as was shown in Figure 2.12. The ingress processing has determined the destination port and it is up to the fabric to arrange the delivery. For the fabric, the network packets are data blocks that have an associated destination port, or multiple ports in case of multicast or broadcast frame. The switch fabrics are characterized by their internal architectures and their buffering strategies. This section describes the fundamentals of those areas and about port scheduling.

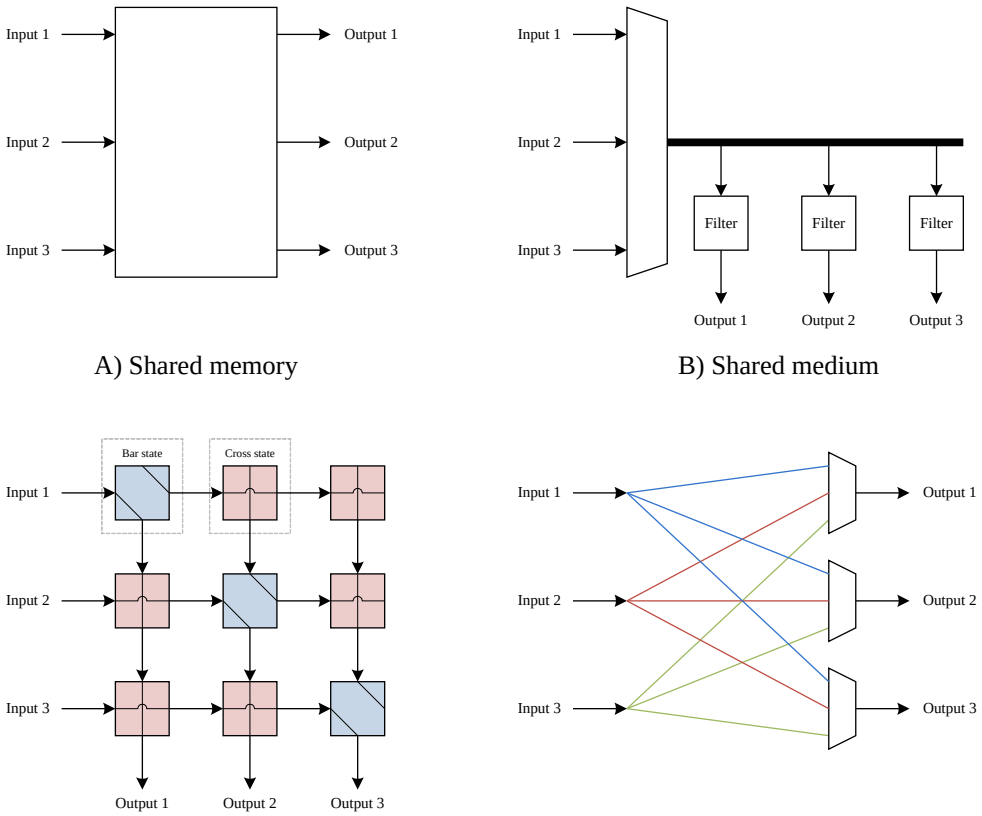
2.5.1 Switch architectures

The switch architectures can be categorized into two groups: time-division switching (TDS) and space-division switching (SDS) [Chao and Liu, 2007]. In TDS, the forwarded packets are placed into a shared memory or multiplexed into a shared medium. This is inexpensive in terms of area but requires the shared element to operate at very high speeds, at least N times the wire-speed, when N is the number of input ports. SDS can be further divided into single-path and multipath switches. In single-path switches, only one path exists between an input and an output port. In multipath switches, there are multiple connections between the ports, thus creating flexibility and fault tolerance but requiring more complex control logic than the single-path. A switch fabric is said to be non-blocking if a packet can be always delivered to an idle port, regardless of other traffic. TDS fabrics are non-blocking if they operate at least N times faster than the wire-speed. In SDS, the internal architecture determines whether it is non-blocking or not.

Next, four different switch architectures, shown in Figure 2.14, are described in more detail. Single-path multistage architectures, such as Banyan, and all multipath architectures, such as Augmented Banyan and Clos, are not covered here. They are more suitable for large-scale switches, which can not bear the steep $O(N^2)$ complexity increase of a single-stage architecture [Chao, 2002].

Shared memory

In shared-memory switches, as in Figure 2.14a, the incoming frames are stored to a memory. In order to fulfill the non-blocking requirement, the memory must be capable of combined



A) Shared memory

B) Shared medium

C) Crossbar

D) Fully interconnected

Figure 2.14: Time-division (a and b) and space-division (c and d) switch architectures.

read and write rates at the aggregated traffic rate of the input ports. Because of this higher internal speed, buffering is needed at the output ports. Although fast memories are expensive, the memory utilization is optimal (more details in 2.5.2). Multicasting is not a straightforward operation because the packets are stored in queues per output port. Special tricks, such as a dedicated multicast queues, can be used to overcome this limitation with the expense of more complex hardware.

Shared medium

As described, the interconnect in the shared-medium switches, depicted in Figure 2.14b, must operate at the aggregated speed of the input ports in order to be non-blocking. The packets are multiplexed to the shared bus from input port. All output ports will listen the bus and detect the packets destined to them. This allows simple multicasting as a packet may be sent simultaneously to multiple outputs. The output ports must have buffers because of the mismatch between the bus-speed and the wire-speed.

Crossbar

Crossbar, shown in Figure 2.14c, is the simplest matrix-type switching architecture. An N-input, M-output crossbar has N times M blocks called crosspoints. Each crosspoint has two possible states: cross (shown in blue in the figure) and bar (shown in red). In the default state, the cross state, the crosspoint connects the left input to the right output and the top input to the bottom output. The outputs are swapped in the bar state. If Input i wants to connect to Output j, the crosspoint at (i, j) is set to the bar state. The crossbar switch is non-blocking (i.e. no internal contention), but suffers from output contention when two or more ports try to send packet to the same output port. The output contention is solvable with buffering or increasing the switching speed, otherwise buffering is not needed.

The vertical crosspoints are connected to the associated output port via a multiplexer or a tri-state bus. It requires M multiplexers with N-inputs to implement an NxM-switch, or NxM tri-state drivers with M buses. The crosspoints are controlled either using implicit external control (e.g. from a scheduler) or by inserting information about the output port(s) to the packets so that the crosspoints can choose the correct state when they receive a packet. The latter is called self-routing.

The crossbar switch is able to support multicasting, although this increases the complexity of the scheduling. In crossbar architecture, each output port has a dedicated scheduler logic that accepts requests from input ports. With multicasting, the input port must get grants to all output ports of the multicast simultaneously before sending the packet to the switching fabric. Therefore, the multicasting increases the likelihood of output contention and decreases the aggregated throughput.

Fully interconnected

In a fully interconnected switch, as in Figure 2.14d, every input and output port pair has a dedicated connection that is not shared among others. The packets from a input port is broadcasted to every connected output port. The output port will filter the packets as in the shared-medium switch, but on the contrary, packets may be received simultaneously from several input ports. The speed-up requirement of the shared-medium switch is changed to the space overhead by using separate connections in this switch architecture.

Due to the possibility of multiple simultaneous packets, the architecture requires output buffering. If these buffers are implemented, the architecture becomes topologically identical with the crosspoint-buffered switch (see 2.5.2), thus providing the same performance and implementation complexity [Chao and Liu, 2007].

2.5.2 Buffering strategies

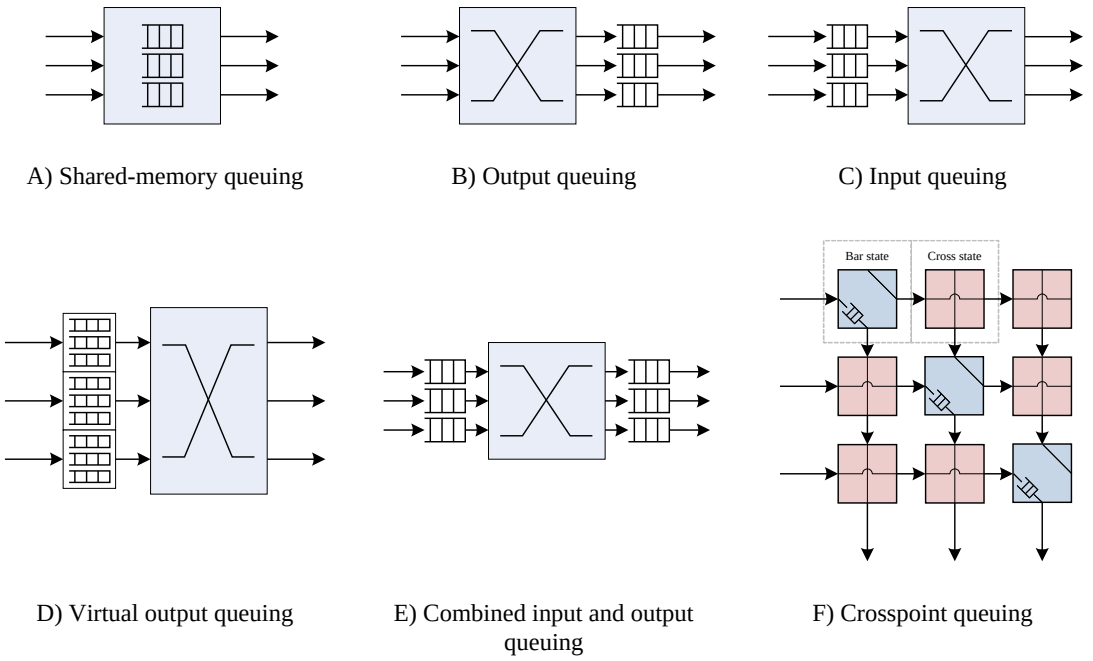
Buffers and buffering strategies are key parts of switching fabrics. Although buffering causes latency and requires additional memory, it is still required in order to smooth down the bursty traffic and to resolve congestions. For example, when two or more ports have packets destined to the same port, only one packet can be accepted by the output port at a time and the other packets must be placed in a buffer. This is referred to as output contention. Of course, no amount of buffering will be enough if the situation lasts for a long time. Also, large buffering causes large latencies, and from the receiving host's point of view, large latencies may be as bad as lost packets. The output port contention can be alleviated by speeding up the forwarding rate of the fabric. For example, if the forwarding rate is double compared to the wire-speed, two input ports can access the same output port during the same time slot. Small buffers are still needed due to the speed differences, though.

Buffers usually operate in First-In-First-Output (FIFO) manner, where the packets are retrievable in the order they are received. More advanced buffers may allow random access to the packets. Naturally, this increases the complexity of the surrounding logic and requires memories that support random access efficiently, but also allows packets to be used in other than FIFO order.

Overflow refers to a situation where a buffer is full and can not therefore accept incoming data. In switches this causes packet loss. Back-pressure is needed to inform the sender about the condition and to hold the sender from transmitting the packets as long as the buffer is full.

Shared-memory queuing

In shared-memory buffering, Figure 2.15a, the incoming packets are buffered in a shared memory that contains logical packet queues. It is almost solely used in shared-memory switching fabrics (see Figure 2.14a). It provides the optimal memory usage because a single memory



A) Shared-memory queuing B) Output queuing C) Input queuing

D) Virtual output queuing E) Combined input and output queuing F) Crosspoint queuing

Figure 2.15: Buffering strategies for switching fabrics.

is used for all ports and not partitioned into segments. The shared-memory buffering requires fast memory as N write and M read operations must be possible at any given moment. Typically the memory accesses are serialized. This allows to use standard dual-port memories. The bandwidth requirement is still valid, though.

Output queuing

In output queuing (OQ) the buffers are placed on each output of the switch as shown in Figure 2.15b. This structure enables the switch fabric to operate at faster rate than wire-speed, thus accepting (sequentially) packets from multiple input ports in a normal time slot of a packet. Therefore, the output queuing with speed-up resolves the output contention problem. It provides the optimal delay-throughput performance for all traffic distributions [Hluchyj and Karol, 1988]. The major downside is that the memory speed requirement increases as the port count increases. Consequently, this limits the switch size (i.e. the port count). Also with variable-length packets, the reassembly logic becomes troublesome as it has to search through the whole buffer while combining the pieces which are interleaved among packets from other ports.

Input queuing

The buffers can also be placed in front of the switching fabric as shown in Figure 2.15c. This structure is called the input queuing (IQ). Compared to OQ, here the buffer's speed is related to wire-speed, not to the port count, and as such, is more suitable for larger switches. The major problem here is the head-of-line (HOL) blocking, which restricts the throughput. The problem is depicted in Figure 2.16 where input port 1 cannot send the head-of-line packet to output port 3 as it is already reserved by input port 2. The next packet in the queue, marked in red, cannot be sent either, even though the output port 2 is free, because the HOL packet is blocking the port. Due to the HOL problem, the throughput of an input buffered switch with uniform traffic is limited to 58.6% [Karol et al., 1987].

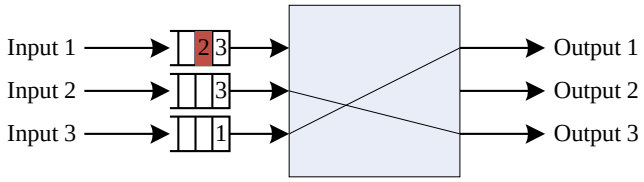


Figure 2.16: Head-of-line (HOL) packet blocking at input port 1.

Virtual output queuing

Virtual output queuing (VOQ), in Figure 2.15d, does not suffer from HOL blocking even though the buffers are at the input ports similarly to IQ [McKeown et al., 1996]. Here each port's buffer is divided into M logical queues that correspond one of the output ports. So, incoming packets destined to the same output port are placed into the same queue, thus eliminating the whole HOL problem. The downsides are the poorer buffer utilization, due to the splitted memory architecture, and the increased complexity in scheduling.

Combined input and output queuing

Combined input and output queuing (CIOQ) uses a different strategy in overcoming the HOL problem. The buffers are placed on both sides of the switching fabric as shown in Figure 2.15e. This allows the switching fabric to operate faster than the wire-speed, and therefore copes better with HOL packets. It is proved that a CIOQ switch with a speed-up of 4 can achieve 99% throughput under independent uniform traffic [Chen and Stern, 1991]. If the input buffers

are replaced with VOQs, the CIOQ switch can emulate an OQ switch with a speed-up of 2 under slightly constrained traffic [Iyer and McKeown, 2003]. As said, an OQ switch provides the optimal performance.

Crosspoint queuing

In a crossbar switch fabric, the buffers can locate at crosspoints (Figure 2.15f). This requires as many as NxM discrete buffers and as such does not utilize the memory very efficiently. However, the performance is similar as with an OQ switch due to non-existing HOL problem. The buffers are typically quite small because they have to be implemented using the memory cells available on the switch IC. Consequently, some designs use VOQs at each input to provide additional buffering capacity.

It is worth noting again, that a crossbar switch with crosspoint queuing is topologically and performance-wise equivalent to a fully interconnected switch with dedicated buffers for each input port at output ports.

2.5.3 Scheduler

The main task of a port scheduler is to provide arbitration to shared resources (e.g. a buffer or a shared-medium) and to offer fair scheduling among the users (e.g. input port). The arbitration ensures that only one user at a time is granted to access the resource. Without fairness, a user might not get its turn to access the resource, thus leading to starvation. The scheduling also has a major effect on the overall performance of the switch. Prioritization (e.g. for QoS) is also handled by the scheduler.

Chao and Liu [2007] present a good selection of different scheduling algorithms and their analysis. An algorithm may be targeted to a specific switching architecture or may be for generic use. Typically the latter ones are more simple and provide less features. For example, a simple algorithm could grant access in a round-robin fashion, while the more complex algorithm could use buffer statistics and the grant history when making the decision.

Most commonly the switch architectures have separate schedulers for each port that are operated independently from the other ports. This allows the schedulers to be run in parallel, but on the other hand, they cannot determine the optimal scheduling because of their limited view.

2.6 Network security

Network security tries to defend the system from various vulnerabilities and weaknesses by monitoring and filtering the traffic. The security is created in layers, so that a compromise in one layer does not compromise the whole system. This is called “defense in depth” [Northcutt et al., 2005]. The security perimeter begins from the outmost router, continues through the intranet, and ends to the host and its user. The following subsections describe several different tactics and devices that increase security and limit exposure.

2.6.1 Router

A router can act as a simple packet filter while doing the normal IP routing. For example, it can verify that the source IP address matches the packet’s ingress port. It can also block traffic to black-listed IP addresses. This can be useful for example to deny a virus or trojan to contact its central server.

Network address translation (NAT) and network address port translation (NAPT) are address translation schemes that can also protect the intranet so that connections can only be originated from the intranet. Other benefit is that NATs hide the internal network structure thus making attacking harder. Although “security by obscurity” is not a tactic to trust in, it still plays a subtle role in the defense in depth scheme.

2.6.2 Firewall

A firewall is a networking device that is dedicated for filtering network traffic. Software firewalls are used in protecting single hosts while hardware firewalls can protect whole networks. A hardware firewall typically has two network ports and is placed in-line so that all traffic flows through it.

There are three different outcomes of the filtering operation. A packet can be accepted and therefore allowed to pass the firewall. If the packet is not qualified to pass the firewall, then the firewall may either drop or reject the packet. The difference is that the drop operation simply discards the packet without informing the source host while the reject operation notifies the source host by sending an ICMP message to it.

In the simplest form of packet filtering, stateless filtering, the firewall does not keep any record of the traffic and filters packets based solely on the current packet. Therefore, it can not track connections per se. However, because TCP header always contain the internal state of the connection, a stateless firewall may control the establishment procedure by allowing SYN packets only coming from the intranet, for example. This guarantees that the connections are only established from the intranet.

Stateful filtering covers Layer 4 and below. It tracks the connections and is able to make dynamic filtering rules. For example, when a TCP SYN packet is sent from the intranet, the firewall would create a dynamic rule that allows the SYN-ACK packet to come back. For UDP traffic, which is connectionless by nature, the firewall could act similarly. When a host in the intranet sends a UDP packet, the firewall would create a rule that accepts the returning UDP packet with swapped source and destination addresses and ports.

Stateful inspection examines the packets in more detail and uses the knowledge of the application protocol (Layer 7). This allows problematic protocols, such as FTP, to pass securely through the firewall. In FTP, the control connection is established as any other TCP connection but the data connections are created in reverse direction (i.e. the server contacts the client). Therefore, the firewall must create a dynamic exception for the incoming data connection.

Deep Packet Inspection is the latest trend in firewalls [Northcutt et al., 2005]. The packets are inspected and compared against predefined signatures, similarly to any antivirus software. This can detect malicious packets, such as viruses, before they even get to the computer. Due to its complex nature, Deep Packet Inspection requires a lot of processing power to be feasible.

2.6.3 Proxy

A proxy is a device that acts as a middle-man between a client and a server. The client connects to a proxy and requests for a certain service, such as a web page. The proxy then connects to a server, that can provide the requested service, and passes the data back to the client. Therefore, the client and the server do not have a direct connection but an indirect connection through the proxy.

By using a proxy, the information of the internal infrastructure (such as IP addresses, operating systems) does not leak outside the proxy. It is also evident that the proxies itself must have been secured properly, as they are visible to the outside but have also access to the inside.

Some proxies have the knowledge of the application specific protocol (Layer 7), such as FTP. They can inspect the application protocol, check for anomalies, and possibly even modify requests and responses. Generic proxies, providing only TCP connections for example, do not offer this advantage. Another benefit of application specific proxies is that they can cache requests and responses, thus reducing the amount of traffic going outside and through other network devices.

2.6.4 Intrusion detection system

An intrusion detection system (IDS) tries to detect any signs of anomalies in the system. A host IDS analyzes the log files, checks the services, and verifies the file-system integrity. A network IDS inspects the flowing traffic passively. A human analyst goes through the reports and the alarms from the IDSs, and decides whether to take action or not. The general problem of IDS is that they generate a lot of false positives, which must be checked manually by the analyst.

The detection is based on Deep Packet Inspection. The software running on the IDS has the knowledge of a wide range of different application protocols and therefore has the ability to detect abnormal events. This naturally requires a lot of memory, storage, and general computational resources, and as such, IDS is suitable to run on a normal computer hardware.

2.6.5 Intrusion prevention system

The next step from IDS was the development of intrusion prevention systems (IPS). In IPS, the manual labor of reacting to the alarms from IDS was replaced with automated actions. An IPS can autonomously kill a connection, drop a packet, or modify a packet. An advanced IPS can co-operate with other network devices, such as firewalls and routers. It could instruct the firewall to block an IP address or a port, or quarantine a host by reconfiguring the router. Because of the automated actions, the false positives in IPS may lead to self-imposed denial of service.

Network intrusion prevention systems (NIPS) are categorized into two groups [Northcutt et al., 2005]: chokepoint devices and intelligent switches. A chokepoint NIPS is similar to a firewall but has a deeper knowledge of the network protocols. It is placed in-line so that all network traffic flows through it. The intelligent switch NIPS functions as a normal high-end router but

also has the IPS functionality. The benefit is that this type of NIPS has a better view point of the network and thus can prevent the intrusions with precise control (e.g. close down a switch port rather than closing down the outside network connection).

The requirements of IPS are very similar to IDS [Northcutt et al., 2005]. IPS needs to operate at least at the aggregated wire-speed, therefore requiring a lot of processing power. Some protocols are stateful so IPS needs to keep track of the state. The prevention depends on good detection mechanisms, that require good knowledge of the traffic. To reduce the amount of false positives, the signatures and parameters must be accurate and up to date. The last point, which differentiates an IPS from an IDS, is that the IPS must be able to extinguish a possible attack.

2.6.6 Host

A host is the last device in the chain of network security. Typically the host is also the ultimate target of the attack as it contains or controls the asset that the attacker is interested in. This means that also the host's security must be increased. This process, called hardening, typically contains steps such as closing unnecessary services, keeping all software up-to-date, running software firewall, and running antivirus software.

2.7 Method to process high-speed network traffic

This thesis concentrates on full-duplex wire-speed network packet processing at gigabit speeds. The processing system is placed in-line of an Ethernet network as illustrated in Figure 2.17. The processing itself is distributed over several computing nodes inside the system. The system should be transparent from the outside perspective. The packet processing, however, can create, modify, or delete packets, so it is not a pure passive system.

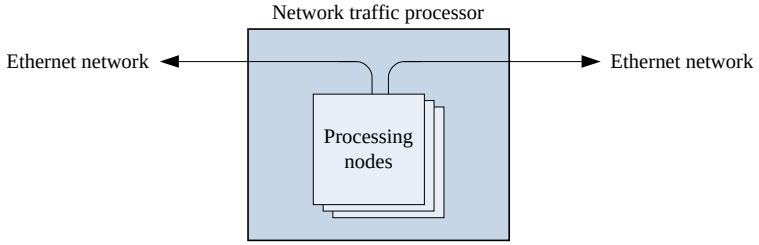


Figure 2.17: Distribution of passing network traffic to several processing nodes.

The problem resembles load balancing, except the traffic here is flowing in two directions. The next sections describe the traffic distribution in detail and show possible architectures and system topologies. Because it is assumed that a node is unable to process the traffic at the incoming rate, a special hardware, here called the controller, is needed to control the traffic flow to the nodes.

2.7.1 Controller's tasks

The controller's processing flow for one direction is depicted in Figure 2.18. When a packet is received from an external port, the controller chooses a node which will get the packet for processing. After the node has finished processing, the controller selects the correct external port where to sent the processed packet. The correct port is the external port that did not receive the original frame. This means that there has to be information about the direction of each packet so that the final dispatching happens correctly. Due to the transparency requirement, the system must preserve the original Ethernet frame, including the MAC addresses. Because there can be countless different MAC addresses, hashing or storing the MAC addresses are not viable options in identifying a frame. Therefore the only option is to encapsulate the original frames at the ingress processing of the controller and then decapsulate at the egress processing.

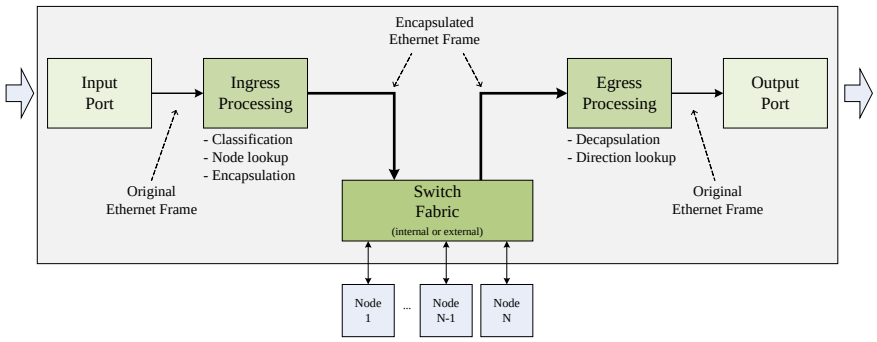


Figure 2.18: The processing flow of the system for one direction.

Because of encapsulation, the effective bandwidth between the controller and a node is decreased. This is not a problem, though, if the processing capability, not the network bandwidth, is the limiting factor of the nodes. A real problem is that the encapsulation may produce frames that are longer than the maximum allowed length of the Ethernet specification. Luckily, the jumbo frames are commonly supported in the network devices and hosts. The contents of the encapsulation header depends on the system architecture as described later.

2.7.2 System topologies

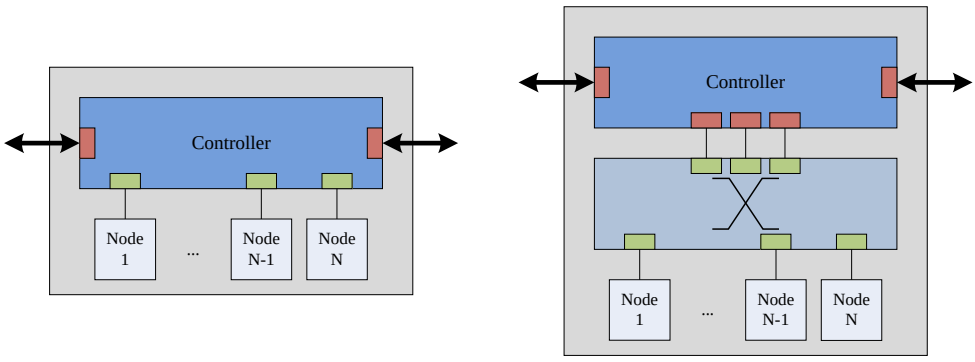
The nodes are connected to the controller either directly or through an external Ethernet switch as shown in Figure 2.19. In the direct connection option, the controller has a dedicated port for each connected node. This requires that the controller has a minimal internal switch. The nodes are operating in promiscuous mode, i.e. accepting all incoming Ethernet frames. This option requires a very minimal encapsulation as it only contains the direction information.

The advantage of the other option is that it uses an off-the-shelf switch to provide the connections to the multiple nodes. This means that the nodes can be added without adding more ports to the controller itself. The downside is that the encapsulation header must contain a full MAC header, so that the switch can forward the packets correctly.

The previously described processing systems can be chained together (i.e. connected in series) so that all the traffic flows through the combined system (Figure 2.20a). While this division allows to use several controllers with smaller port count, it does not reduce the processing load as the same traffic is flown through. Also, the overall latency is increased as the same packet goes through multiple controllers. The controller's latency is assumed to be negligible, though, when compared to the processing node's latency. The major point is, however, that this kind of "scaling by chaining" does not require any modifications to the system and is therefore a viable option for increasing the amount of processing nodes.

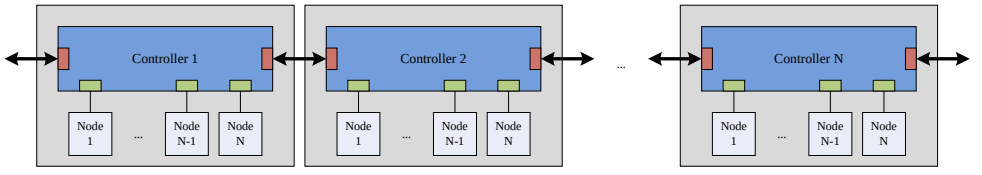
A controller and a processing node form the smallest possible configuration. The controller in this case could be integrated with the node's network interface card (NIC), as shown in Figure 2.20b, so that it would have two external ports (one per direction) and a connection to the node's internal bus (e.g. PCI Express). The downside is that this scheme requires a custom software driver for the new network interface and hardware modifications to the processing nodes. Other characteristics follow the described chaining topology.

A new topology is obtained by splitting the controller in Figure 2.19b into two identical halves A and B, as depicted in Figure 2.20c. Here, A will do the ingress processing and B the egress processing for traffic traveling from left to right, and vice versa for the other direction. As a result, the complexity and the load of the original controller is halved without increasing the latency. The downside is that this setup consumes one additional port from the switch and requires two identical hardwares. Naturally, the functionality of both halves could be implemented within a single device by logically partitioning the hardware.

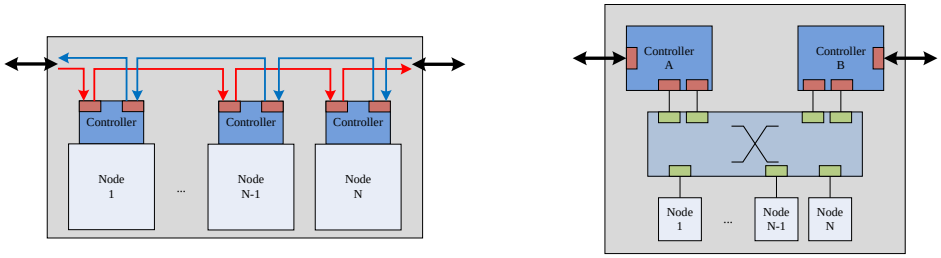


A) Direct connection B) Connection through a switch

Figure 2.19: System architectures for connecting the controller to the processing nodes.



A) Chaining



B) Integrating controller to NIC C) Pre- and post-processing

Figure 2.20: Variations of system topologies.

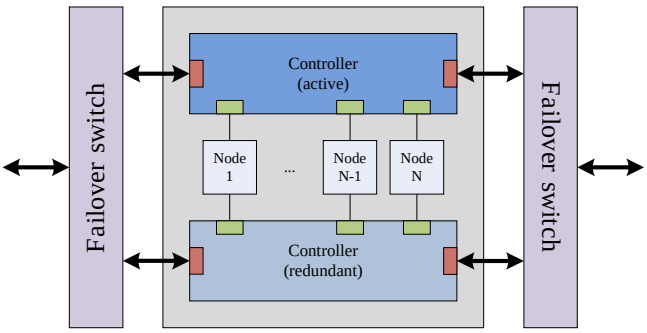


Figure 2.21: Increasing availability by adding redundancy.

In-line processing systems are critical points of the network as all the traffic flows through them and a single error in the system can paralyze the whole network. Furthermore, the chaining will increase the error probability. The availability of a critical resource is typically secured by duplicating the resource and the connections as shown in Figure 2.21. In normal state, the traffic flows through the resource. When an error is detected on the resource, the traffic is automatically routed to a redundant unit of the resource. This is called a failover mechanism and it is available in commercial network switches.

2.7.3 Encapsulation alternative

VLAN tagging may be used as an alternative to the full MAC header encapsulation. There the external switch is forced to forwared frames based on the VLAN tag and ignore the MAC addresses. This resembles closely the Ethernet VLAN Cross Connect scheme described in [Sprecher et al., 2006]. The direction field and the destination node can be encoded to the 12-bit VLAN tag (see 2.1.3) by, for instance, using the two highest bits to represent the packet's direction and the lower 10 bits the node ID. The switch should be configured with the following rules, assuming the controller has three ports and port 1 is used for direction A-B, port 2 for direction B-A, and port 3 for both directions (overflow port):

- Controller port 1 and node port i belong to VLAN group $(i + 0 * 1024)$. (direction A-B)
- Controller port 2 and node port i belong to VLAN group $(i + 1 * 1024)$. (direction B-A)
- Controller port 3 and node port i belong to VLAN group $(i + 2 * 1024)$. (direction A-B)
- Controller port 3 and node port i belong to VLAN group $(i + 3 * 1024)$. (direction B-A)

These rules exploit the VLAN switching algorithm. Because each defined VLAN will contain exactly two switch ports, and the frame's ingress port is one of those, the switch floods the frame to the other port as it has no other alternatives. The learning algorithm is basically bypassed here, and the switching is controlled solely by the VLAN tag information.

If the network links between the controller and the switch are combined into a LAG (see 2.3.3), the rules can be reduced as shown below. The controller does a similar load-balancing as above and divides the traffic over the links in the LAG.

- LAG and node port i belong to VLAN group $(i + 0 * 2048)$. (direction A-B)
- LAG and node port i belong to VLAN group $(i + 1 * 2048)$. (direction B-A)

Chapter 3

Designing Network Hardware on Reconfigurable Platform

In this chapter designing with an FPGA and the technology involved is studied further. The chapter describes the basics of digital logic designing and gives an overview of FPGA technology. The relevant external connections of an FPGA and their electrical characteristics are described. The suitable memory options are analyzed from the application's view point. The high-level view of Ethernet given in Chapter 2 is elaborated here to cover the physical layer and the available interfacing options.

3.1 Digital logic designing

Hardware programming is fundamentally digital logic designing. The logic is designed in methodology that uses synchronous clocking. Figure 3.1 illustrates the conventional synchronous system with register elements that are clocked with the same clock signal [Dally and Poulton, 1998]. The combinational logic between the register stages implements a function f(x). The register element samples the input signal at the rising edge of the clock and sets the output to reflect this after t_{CO}. After t_d the function f(x) is ready and the input signal of register 2 has settled. t_d includes the logic delay as well as the wire propagation delay. The logic path that has the largest $t_{CO} + t_d$ is called the critical path, because it determines the maximum operating frequency F_{max} of the design. Register pipelining can be used to reduce t_d, and thus increase F_{max}, as was shown in 2.4.2.

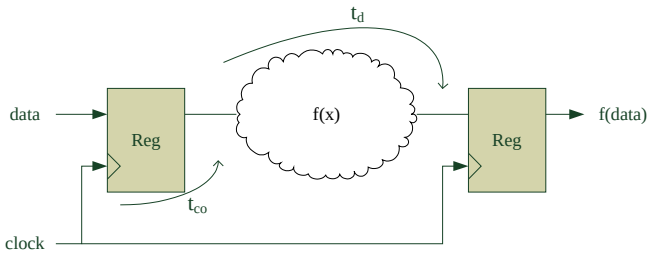


Figure 3.1: Synchronous system with register elements and combinatorial logic.

The input signal should be stable before and after the time of the sampling at the register. The setup time t_s and the hold time t_h specify the needed boundaries for successful sampling, and they should be always guaranteed [Dally and Poulton, 1998]. However, a typical design contains multiple clocks that may not be synchronous with each other. An area of logic running with the same clock is called a clock domain, and the border between two domains is called a clock boundary. When a signal crosses a clock boundary, there is a probability that the setup and hold times are violated. A synchronization failure happens when the sampling device can not decide which of the two events (e.g. the clock or the input signal) came first. In this case, the decision process takes longer than normally and may finally produce the wrong result to the output. An even worse scenario is when the two events occur at exactly the same time. There the device could end up in a metastable state where the output would hover between logical one and zero. The metastable state can last forever in theory, but in practice the noise will end the state in a certain time. The risk of metastability can be greatly reduced by using two or more registers in series when crossing clock domains [Altera, 1999].

3.2 FPGA overview

A field programmable gate array (FPGA) is a device that can be programmed after the manufacturing process. The device contains logic blocks that are interconnected through a routing architecture. The external connections are handled by the IO (Input/Output) blocks. This is illustrated in Figure 3.2. Both the logic and the routing are configurable to some degree to allow the functions to be programmed.

The target application of this thesis requires high performance and a reprogrammable FPGA device, and therefore a static RAM (SRAM) based FPGA was a natural choice. Other FPGA technologies are available but they lack performance (flash-based devices) or are one-time pro-

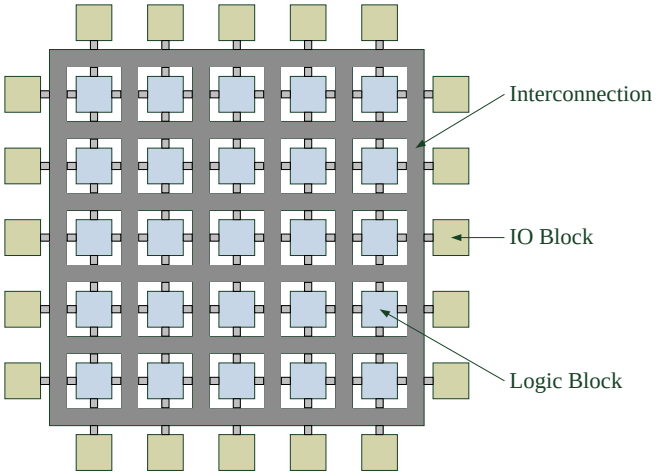


Figure 3.2: General FPGA architecture.

grammable (antifuse devices) [Rose et al., 1993]. A non-SRAM FPGA is more suited for applications that require low power consumption, immunity against radiation, security against reverse engineering, or instant power-up. The SRAM-based FPGAs are faster and have better logic density because they use the standard CMOS process and are therefore eligible for the latest advancements in IC technology. The flash-based FPGAs use an older CMOS process because their manufacturing requires additional fabrication steps on top of the standard CMOS process. Even though a cell in a flash-based FPGA requires less transistors than in SRAM-based FPGA, an SRAM-based FPGA device has better cell density due to the newer IC technology with smaller transistors. An SRAM-based FPGA must be programmed every time the device is powered up because of its volatile memory. The configuration is normally loaded from an external device, such as a flash memory. To prevent copying and reverse-engineering, the latest FPGA devices contain decryption logic on-chip so that the configuration stream may be stored and transferred in encrypted form.

The IC power consumption is typically divided into two terms. The static part, also called the leakage power, is consumed whenever the device is powered on. The dynamic part depends on the operating speed (i.e. frequency) and the activity rate. In other words, it depends on the design. Contrary to ASICs, the static part dominates the total power consumption in SRAM-based FPGAs. The increase is caused by the programmability. The normal power saving tricks used in ASICs, such as clock gating, are inefficient in FPGAs because firstly, they mainly reduce the dynamic power and secondly, they cannot be implemented properly with programmable logic. For instance, in FPGA the clock gating is done by disabling logic, not by disabling the clock, thus the clock trees are still active and consume power. According to

[Zhang et al., 2006], dynamic power savings from clock gating is about 50% to 80% and total power savings about 6% to 30% when compared to the ASIC technology. The static power consumption is a problematic issue as it increases when moving to newer process technologies and to smaller transistor sizes [Shang et al., 2002; Altera, 2007d]. Some of the newer FPGA devices have power saving tactics that completely shut down the unused regions of an FPGA (reducing static power), set some regions to low-power mode (reducing dynamic power), and set some regions to high-speed mode [Altera, 2007d]. The low-power and high-speed modes are done by scaling the voltage appropriately.

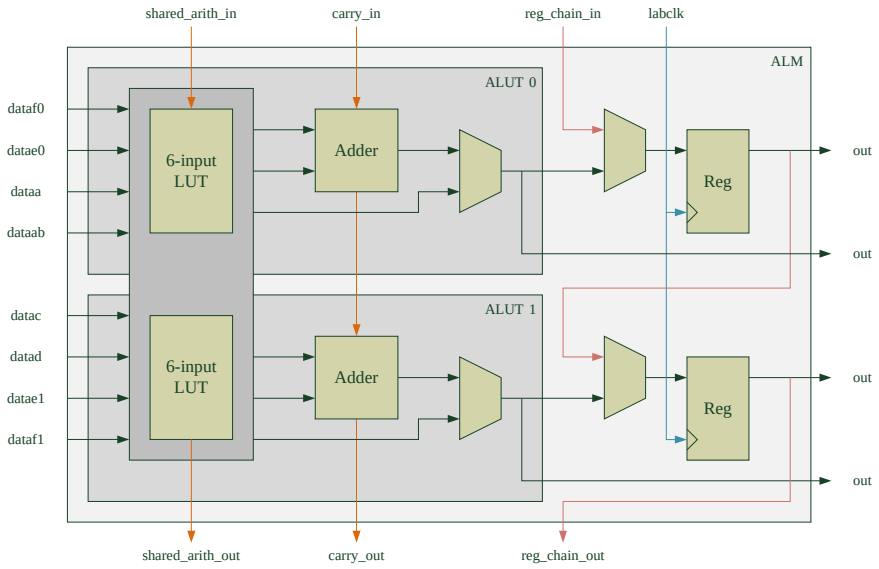


Figure 3.3: Adaptive logic module, the basic logic block in Altera Stratix III.

The logic blocks implement the programmed combinatorial and sequential logic functions. The actual implementations vary from device to device but typically they are based on one or more of the following: look-up tables (LUTs), multiplexers (MUXs), normal logic gates, or registers. The basic logic block in Altera's Stratix III device is called the adaptive logic module (ALM) and is shown in Figure 3.3 [Altera, 2007c]. A logic array block (LAB) contains ten ALMs, each having two 6-input LUTs that share the eight input signals [Hutton et al., 2004]. The LUTs in the ALM can be divided into several configurations depending on the need. This includes a configuration with two 4-input LUTs, which can efficiently replace two traditional 4-input logic elements (LE). This is important as 4-input LUTs are considered to provide the best area-delay product [Rose et al., 1990]. Other configurations are, for instance, 5-LUT + 3-LUT and 6-LUT. The ALM includes also two full-adders that can be chained with the neighboring ALMs within the LAB to form up to 20-bit wide adder. The results from the LUTs or from the adders are fed to the interconnect directly or through a pair of registers. The registers can

be chained to offer an efficient shift-register implementation. Furthermore, if the LUTs don't use the registers, the registers can be used for other purposes (e.g. for shift-registers). The logic blocks are connected to the programmable interconnection fabric. In Stratix III's case, each LAB has connections to a local interconnect and to the device-wide row and column interconnects.

Beside logic blocks, an FPGA fabric usually has embedded SRAM memory blocks. The total capacity ranges from a few kilobits to several megabits. In Stratix III, there are three different types of memory, all running at a maximum speed of 600MHz and supporting single- or dual-port random access. MLAB is similar to the LAB but has a 640-bit memory element inside. MLABs are optimized for delay lines, small first-in-first-out (FIFO) buffers, and shift-registers. M9K blocks are pure memory elements suitable for general purpose usage with their 9-kbit capacity. M144K blocks are large, 144-kbit memories designed for packet buffering. Larger memories, or FIFOs for instance, can be formed by combining several blocks together.

Even though the logic blocks can be used to implement any combinatorial or sequential circuit, many of the current FPGAs offer hard-wired function blocks. Most commonly the blocks implement computational functions, such as multiply-and-accumulate (MAC), but even complete microprocessors can be found in some devices. The hard-wired function blocks provide better performance in a smaller area.

The synchronous hardware design is based on synchronous clocking. The reconfigurable fabric offers multiple predefined, but programmable, clock trees that are connected to the flip-flops in the LUTs and to other components that require clocking. Most of the devices have phase-locked loop (PLL) or delay-locked loop (DLL) blocks that can create new clock signals based on the input clock signals. These are critical when interacting with other peripherals at high-speed.

As with processors, memories, and other ICs, the variations in the manufacturing process limit the final product's maximum performance. After manufacturing, each device is measured to verify the maximum operating condition. The high performing device units are marked and then sold with premium. This marking, called speed grade, naturally affects the FPGA's speed, and defines the device's logic speed, F_{max}.

3.3 External connections

An FPGA interfaces to peripherals and other components via pins. The amount of user-configurable pins ranges from a hundred to over a thousand. Each I/O (Input/Output) pin has several configuration parameters that, for instance, define the used electrical standard (e.g. 1.8V LVTTL). The physical pin connects to an IO cell located on the FPGA fabric (Figure 3.2). The cell contains a register for precise timing, a tri-state driver for bi-directional communication, and resistors for signal termination without external components, among others. The termination resistors eliminate signal reflections, thus improving signal integrity.

To successfully interpret the received signals, the receiver has to know where a symbol ends and where the next one begins. This is commonly achieved by using a separate clock signal that is sent by the transmitter along with the data signals. This clock signal specifies the correct timing for the receiver. The data may be clocked on one edge (typically on the rising edge) or both edges of the clock. These are called single data rate (SDR) and double data rate (DDR), respectively. In SDR signaling, the clock operates at twice the data rate and might cause signal integrity problems when operating at high frequencies. On the other hand, the timing window in DDR signaling is tighter, thus the clock jitter and other timing anomalies become more problematic. In FPGA, the DDR signaling is encoded/decoded inside an IO cell so that the internal fabric can operate with SDR data [Altera, 2007a]. For instance, each incoming DDR data signal, clocked on both edges, is expanded to two SDR signals, clocked on the rising edge. As a result, the FPGA's internal data bus is twice the width of the external DDR bus but operates at half the frequency.

Typically the whole data bus is synchronized to one clock signal. This reduces the total signal count but makes the signal routing more difficult as all data signals and the clock signal must have the same length to minimize the clock skew. The current trend is to move from parallel signaling to serial signaling. The serial signaling reduces both the pin count and the clock skew. The downside is that it requires additional logic for the parallel-to-serial conversion at the transmitter and for the serial-to-parallel conversion at the receiver. In addition, these conversions consume power and create latency. The term SerDes (serializer and deserializer) refers to the converters, and sometimes also to the serial communication in general. Some FPGA devices have on-chip hard-wired SerDes units capable of very high signaling rates, up to 8.5Gbps [Altera, 2008c].

The clock information can also be encoded to the data signals, in which case the receiver has to do clock recovery to reconstruct the original clock signal. This eliminates the clock skew as the data signal and the clock signal are combined into one signal. The receiver has to have a reference clock running at an approximately correct frequency, which is then tuned, or locked, to the incoming data. SerDes enabled FPGAs normally contain a clock and data recovery (CDR) circuits for efficient operation.

Typically, the signals are driven in single-ended, i.e. a binary signal is referred to a common ground. The other method, favored especially in high-speed communication, is to use differential signaling where one pin is driven with the signal and a second pin with the complement of the signal. Obviously, the differential signaling doubles the pin count usage compared to the single-ended signaling but it also provides better performance in terms of signal integrity, and therefore allows higher signaling frequencies and data rates. In practice, the ratio of pin counts is not two but between 1.3 and 1.8, because in high-speed single-ended systems a return path is required for every 2-4 signals [Dally and Poulton, 1998]. IO cells in FPGAs are capable in driving and receiving both single-ended and differential signals.

A signal must have enough transitions for successful clock recovery. A coding method can be applied to the transmitted data to ensure a certain transition frequency. The coding method can also offer other properties. It may have out-of-band symbols to convey control information among the data symbols. The resulted signal may be DC-balanced, i.e. it contains equal number of ones and zeros over a period of time. One of the most popular coding schemes is the 8b/10b code [Widmer and Franaszek, 1983]. There 8-bit data words are encoded into 10-bit code words by using 5b/6b and 3b/4b encoders. The coding scheme has 12 control symbols of which three can be used as commas. The comma symbols are special as their bit sequences are unique and are therefore useful in synchronization. A more efficient coding scheme is the 64b/66b encoding where 64-bit data words are encoded into 66-bit code words by inserting a 2-bit prefix in front of the data word [IEEE802.3, 2005, Clause 49]. Prefix “01” defines that the following 64 bits are entirely data, prefix “10” that the bits are both control and data. Prefixes “00” and “11” are not used. The use of these prefixes guarantee a bit transition every 66 bits. The DC-balance is achieved by using a self-synchronous scrambler after the prefix encoding.

3.4 Memory options

Network processing needs memory for look-up tables (LUTs) and for buffering network traffic. The LUTs are typically small in size and are implemented with memories that have small latencies, allow random access, and possibly allow for multiple simultaneous access. The buffers, on the other hand, are large and are accessed by one reader and one writer. The accesses are more or less to sequential addresses. The read and write rates should exceed the wire-speed while the latency is typically not a problem. In addition, the maximum buffering capacity is bound by the overall latency requirement. That is, there is no point to buffer frames too long, say 10msec, as the frames will be worthless for the targeted receiver by then.

Basically there are two types of volatile memory technology in use: static RAM (SRAM) and dynamic RAM (DRAM) [Rabaey et al., 2003]. An SRAM cell requires a total of six transistors, four for storing a bit and two for control. The bit's value stays valid as long as power is fed to the cell. The active structure offers high performance with the penalty in power efficiency and area. A popular DRAM cell architecture has only one transistor and a capacitor providing a very dense layout. The read operation is more complex than the write operation and requires multiple steps. Before reading a cell, the bit lines are precharged to a known voltage. Then, the stored bit value is read by monitoring the changes in the bit line's voltage. Finally, because the readout is destructive, the original bit value is written back. The contents of a DRAM cell must be refreshed periodically, every few milliseconds, otherwise the stored bit value is lost due to leakage. The refresh operation is basically the read operation applied to all the cells. The simple internal structure of DRAM translates to lower power consumption but also to worse performance and latency compared to SRAM. In addition, a DRAM memory demands a more complicated memory controller.

Content-addressable memory (CAM) is often used in network devices to implement MAC tables (see 2.3.3) and in classification engines (2.4.1). Contrary to RAM, the memory entries are located by using data values instead of addresses [Pagiamtzis and Sheikholeslami, 2006]. A CAM memory returns an address, or sometimes the whole data row, if the given data value is found from the memory. The search operation takes a single clock cycle and thus outperforms any other search methods. The cost of this is paid with larger silicon area and higher power consumption. A CAM may be implemented by using the programmable logic of the FPGA. For better efficiency the FPGA's embedded RAM could be utilized [McLaughlin et al., 2006].

FPGAs have almost the same liberties as ASICs when choosing the memory architecture. The following subsections compare and analyze various memory technologies that are suitable for the above requirements. These options differ in latency, performance, storage capacity, cost, and power.

3.4.1 On-chip RAM

RAM located on the IC is called on-chip RAM. In FPGAs, the capacity of on-chip memory varies from a few kilobits to several megabits of SRAM. The performance is up to the fabric speed with zero latency (i.e. data is valid on the next clock edge). The memory's data bus can be as wide as required because of the programmable interconnect.

Typically on-chip memories have two ports that can be clocked with two unrelated clocks. This means that on-chip RAM can be used for CDC without fear of metastability, as long as the two ports do not access the same memory location at the same time.

From the usability point of view, an on-chip RAM is very easy to access and provides high-performance, low-latency, and deterministic behavior, although having rather small capacity. Therefore, the on-chip RAMs are very suitable for implementing LUTs and small CDC buffers.

3.4.2 SDRAM

Synchronous DRAM (SDRAM) offers a very low-cost and dense memory solution with a synchronous interface. The typical SDRAM memory chip has a bit width of 8 or 9 (8 data bits and a parity bit), a capacity of a few gigabits, and multiple banks. Multiple banks allow a bank to be accessed while the other bank is being precharged. The memory is arranged to rows and columns. SDRAM is capable of burst accesses due to its pipelined design.

The memory is controlled with four 1-bit signals: chip select (`CS`), write enable (`WE`), row address strobe (`RAS`), and column address strobe (`CAS`). Memory location is specified with bank information (`BAx`) and multiplexed row and column address (`Axx`). The data bus (`DQx`) is bi-directional.

Figure 3.4 shows a timing diagram of burst read operation [JEDEC 21C, 2003]. Active command (`RAS` asserted) opens the specified row. After tRAS clock cycles, Read command (`CAS`

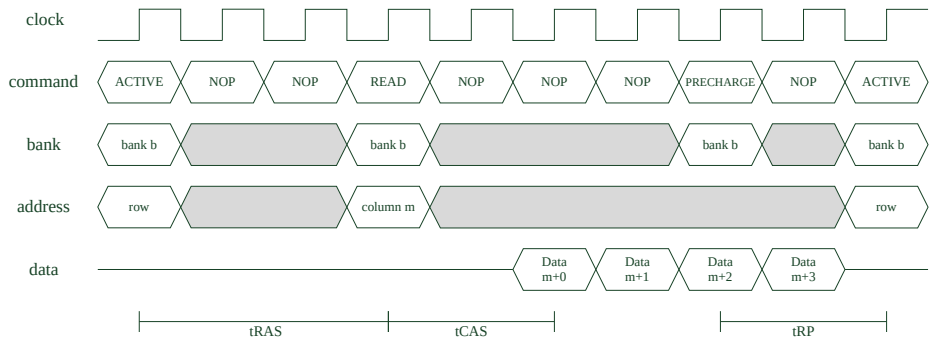


Figure 3.4: SDRAM timing.

asserted) selects the column where the burst transfer begins. The first valid data item is seen on the data bus after tCAS clock cycles. The next data items come on the following clock cycles. The burst transfer is ended with Precharge command (`RAS` and `WE` asserted). This will begin the precharge operation on the selected bank. The next operation may happen after tRP for the same bank. NOP (no operation) commands (all deasserted) are used when no command is issued. The write operation is similar except that Write command (`CAS` and `WE` asserted) is used instead of Read command. The first data to be written is given at the same clock as the command. Memory refresh is started with Refresh command (`RAS` and `CAS` asserted). The command should be called periodically to retain the memory contents.

Table 3.1: SDRAM interfaces. f refers to the used memory clock speed. Bandwidth is calculated for a 72-bit data bus.

Interface	Memory clock	Bus clock	Bitrate	Max. Bandwidth	Standard
SDR	66..133MHz	$2f$	f	9.6Gb	[JEDEC 21C, 2003]
DDR	100..200MHz	$1f$	$2f$	28.8Gb	[JEDEC 79F, 2008]
DDR2	100..266MHz	$2f$	$4f$	76.6Gb	[JEDEC 79-2E, 2008]
DDR3	100..200MHz	$4f$	$8f$	115.2Gb	[JEDEC 79-3B, 2008]

The SDRAM interface has gone through several updates. The current interface types are listed in Table 3.1. The main difference between SDR SDRAM and DDR SDRAM is that the latter uses both edges of the bus clock. Each of the DDR SDRAM evolutions increases the bus clock hence offering higher bit rates. The clock speed of the memory chips, on the other hand, remains at a similar level.

The maximum theoretical bandwidths of each interface type are also shown in Table 3.1. In practice, the achieved throughput drops due to the refresh periods and the latencies. A conservative estimate of the SDRAM efficiency is about 70% [Altera, 2008b]. This means that for buffering purposes including both reads and writes, the throughput will probably drop to about one third. Still, DDR2 SDRAM and faster should be enough for buffering a 10GbE interface.

To conclude, the SDRAM technology is a good, low-cost choice for storing large amount of data. Its suitability for buffering is questionable as it will require complex logic for handling all the internal aspects.

3.4.3 QDRII+ SRAM

The quad data rate II (QDRII) architecture provides a dual-port, high-speed interface for accessing external SRAM memory cells [Cypress, 2007]. The interface has separate data buses for read and write operations and a single, time-divided address bus. The control signals are `WPS` (write port select), `RPS` (read port select), and `BWSx` (byte write select). The interface defines two pseudo-differential (i.e. phase difference approx. 180 degrees) input clocks (`K`, `K#`) and two output clocks (`C`, `C#`) that are used for clocking the data in and out, respectively.

The standard defines two burst options. When using 2-word burst mode, the read address is provided on the rising edge of `K` and the write address on the rising edge of `K#`. When using 4-word burst mode, the addresses use the rising edge of `K` and are provided in turns. The read data is valid after 1.5-clock cycles in QDRII, and 2.0- or 2.5-clock cycles in QDRII+. The write data is provided at the same moment as the write address. A single read operation provides N words on consecutive clock edges, where N is the burst count. Similarly, a write operation must provide N words using N consecutive clock edges.

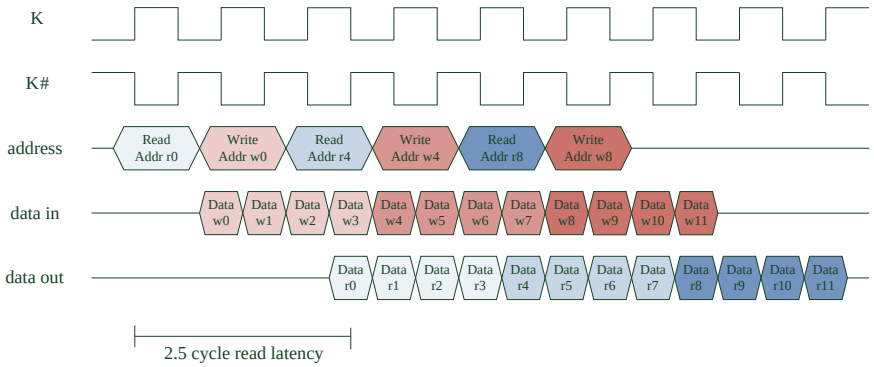


Figure 3.5: QDRII+ timing with 2.5-cycle read latency and burst length of 4 words.

A simplified version of the interface's timing diagram is illustrated in Figure 3.5. As can be noticed, the benefit of the bursting concept is that there are no idle periods on either of the data busses. QDRII+ chips are able to operate at 400MHz [Cypress, 2008] and have a bit rate of 800Mbps per direction. For instance, with 18-bit data buses the total throughput is 14.4Gbps per direction. The chips can be combined in parallel for wider data buses and higher speeds.

The QDRII+ interface shows many desirable characteristics that are critical in fast data processing. The interface's efficiency is always 100%. The latency is minimal and most of all deterministic. The data bus is full-duplex allowing concurrent read and write operations. The only limitation is the low capacity which is of course a fundamental problem of the SRAM technology. In summary, QDRII+ memories are suitable for network data buffering (with capacity restrictions) and for large LUTs that do not fit to on-chip RAM.

3.5 Design flow

The FPGA design flow resembles the one used in normal ASIC design. The FPGA vendor typically provides a suite of tools that handles all the steps in the flow. When necessary, a step can be also performed in a third-party tool.

First the system is described using a high-level hardware description language (HDL), such as VHDL or Verilog. HDL allows the system to be described at different abstraction levels, but designs normally use the register transfer level (RTL). Structures are implemented by either describing them in RTL or using intellectual property (IP). IP blocks can provide, for instance an Ethernet MAC core or a DDR memory controller. This allows designers to concentrate more on the application logic. However, the IP cores usually follow the one-size-fits-all policy and are therefore unoptimized for a specific purpose.

Next, the produced RTL description is fed to a synthesis tool that translates the design to a gate-level structural representation. This representation uses blocks that are available in the chosen FPGA device. Synthesis tools are able to detect high-level structures, such as RAMs, if they follow a certain template.

The place and route tool maps the synthesis results to physical entities on the device. This step also includes routing and pin mapping. The tool creates a netlist and a timing file. The netlist is later used to program the design to the device.

The timing information is used when analyzing whether the design meets the required timing or not. A design constraint sets boundaries for timing (e.g. "F_{max} must be over 100MHz") and informs the tools about the designer's intent. The tools use this information for instance to select which areas need to be optimized. Design constraints are typically expressed using Synopsys Design Constraint (SDC) format. Timing is especially critical when communicating

with external devices. The timing analyzer can implicitly create the timing constraints for the internal functionality but does not have any knowledge of the external components' timing.

The design is verified by using simulation. The design flow allows the simulation to happen at different stages. The behavioral simulations verify the logic on algorithm level. In RTL simulation, the behavioral models are replaced with more accurate RTL models. The gate-level simulation contains the most detailed models and incorporates also the timing information. In simulation, the design under test (DUT) is instantiated from a testbench (TB). The testbench, described in HDL, typically uses a high abstraction level to be able to express the test vectors more easily. The testing can be improved by introducing assertions to the design entries. The assertions are described in Property Specific Language (PSL).

3.6 Network interfaces

The physical layers of Ethernet are usually implemented with a dedicated physical layer device (PHY), which is connected to the MAC sublayer via the Media Independent Interface (MII). This section describes the different MIIs (GMII, RGMII, XGMII), the physical media options, and different transceiver modules. The following text covers only 1Gbps and 10Gbps full-duplex operation.

3.6.1 Reconciliation Sublayer and Media Independent Interface

The Reconciliation Sublayer (RS) is an adapter that translates data between the MAC sublayer and the Media Independent Interface (MII). The data must follow the MAC frame format including the preambles and the FCS. Table 3.2 summarizes the Media Independent Interfaces (MII) used when interfacing to 1Gbps and 10Gbps PHYs. The given bit and pin values are per direction (e.g. a bi-directional GMII has 18 pins).

Table 3.2: Media Independent Interfaces used in interfacing 1Gbps and 10Gbps PHYs.

	1Gbps			10Gbps	
Interface	GMII	RGMII	SGMII	XGMII	XAUI
Signaling	8bit SDR	4bit DDR	1xSerDes	32bit DDR	4xSerDes
Clock signaling	125MHz	125MHz	1.25GHz	156.25MHz	3.125GHz
Clock	Source	Source	Source	Source	Recovered
Pin count	9	6	4	37	8
Specification	Clause 35	HP	Cisco	Clause 46	Clause 47

GMII

The Gigabit Media Independent Interface (GMII) [IEEE802.3, 2005, Clause 35] contains the signals listed in Table 3.3. In transmit direction, all signals are driven by the MAC and are sampled on the rising edge of `GTX_CLK` by the PHY. `TXD` contains the MAC frame data when `TX_EN` is asserted and `TX_ER` is deasserted. The MAC can use `TX_ER` to propagate an error to the PHY if it notices any error after the start of the frame. In the error case, the PHY must create an error to the medium so that the receiving PHY can detect the error condition. `TX_EN` must be constantly asserted from the beginning of the frame until the end of the frame. That is, the MAC frame boundaries are derived from `TX_EN`.

The receiving direction from PHY to MAC is analogous with the transmit direction. All signals are driven by the PHY and are sampled on the rising edge of `RX_CLK` by the MAC. `TXD` contains the MAC frame data when `RX_DV` is asserted. The received data may or may not contain the preamble but must contain the SFD. Assertion of `RX_ER` identifies that an error was detected by the PHY and the frame should be discarded.

Table 3.3: GMII signals

(a) Transmit direction

Name	Width	Description
GTX_CLK	1	clock (125MHz)
TXD	8	data
TX_EN	1	enable
TX_ER	1	coding error

(b) Receive direction

Name	Width	Description
RX_CLK	1	clock (125MHz)
RXD	8	data
RX_DV	1	data valid
RX_ER	1	receive error

RGMII

The Reduced GMII (GMII) is not part of the Ethernet standard but an alternative version to the GMII specified by Hewlett-Packard, Broadcom, and Marvell [RGMII, 2002]. It carries exactly the same information as the GMII but has fewer signals, therefore making it more feasible to implement. The pin count reduction is achieved by using the both edges of the launching clock.

The RGMII signals are listed in Table 3.4. All signals are synchronous to `TXC` in transmit direction and to `RXC` in receive direction. The signals are sampled on both rising and falling edge. On the rising edge the data bus contains the bits 3..0 of the MAC frame data and on the falling edge the bits 7..4. The control signal is also clocked on both edges. On the rising edge, it functions as the GMII's enable signal and on the falling edge as GMII's error signal.

Table 3.4: RGMII signals

(a) Transmit direction

Name	Width	Description
TXC	1	clock (125MHz)
TXD	4	data (DDR)
TX_CTL	1	control (DDR)

(b) Receive direction

Name	Width	Description
RXC	1	clock (125MHz)
RXD	4	data (DDR)
RX_CTL	1	control (DDR)

SGMII

The serialized version of the GMII is called the Serial GMII (SGMII). The specification [Chu, 2001] is made by Cisco and it is not part of the IEEE 802.3 standard. The interface uses two data signals and two clock signals to transfer the same information as in the GMII. The signals are realized as differential pairs, thus the whole SGMII consumes only 8 pins. The GMII data is transferred as 8B/10B encoded. The data signals are therefore operated at 1.25GHz. The data is synchronized to the clock signal running at 625MHz and is valid on both edges (i.e. DDR operation). The receiver may recover the clock from the data signal but the PHY must always provide also the actual clock signal. The clear advantages of this interface are the reduced pin-count due to the serialization and the better signal integrity due to the use of differential signaling. The downsides are the requirements for 8B/10B encoders and the signaling rate, which is much higher than in the GMII.

XGMII

The MII for 10 Gigabit Ethernet is called the XGMII [IEEE802.3, 2005, Clause 46]. The interface's signals are shown in Table 3.5. As the transmit and receive signals share the same function, only the transmit direction is explained in more detail. By replacing the signal names, the provided information is applicable also to the receive direction.

Table 3.5: XGMII signals

(a) Transmit direction

Name	Width	Description
TX_CLK	1	clock (156.25MHz)
TXD	32	data (DDR)
TXC	4	control (DDR)

(b) Receive direction

Name	Width	Description
RX_CLK	1	clock (156.25MHz)
RXD	32	data (DDR)
RXC	4	control (DDR)

The signals `TXD` and `TXC` are synchronous to `TX_CLK` and are sampled on both the rising and falling edge. The data signals `TXD` are divided into four separate 8-bit lanes, named here as

d_3,..,d_0. The control signals `TXC` are also divided into four 1-bit control signals, named here as c_3,...,c_0. So, each data lane d_i has one associated control signal c_i. The control signal c_i tells whether d_i contains data or a control symbol. The control symbols are defined in Table 3.6. Data from the MAC sublayer is arranged to the lanes of XGMII so that the *ith* byte of the MAC frame is placed in $d_{i \bmod 4}$. The bit order inside the bytes remains unchanged.

Table 3.6: Encoding of control and data in XGMII.

c_i	d_i	Description
0	-	Normal data
1	0x07	Idle symbol
	0x9C	Error sequence (only valid in lane 0)
	0xFB	Start symbol (only valid in lane 0)
	0xFD	Terminate symbol
	0xFE	Error symbol
	others	Reserved

The Idle symbol is sent on all lanes when there is no MAC frames to be sent. At the beginning of a frame, lane 0 will contain the Start symbol and lanes 1..3 the normal MAC preamble from the MAC header. On the next clock edge (note, this can be rising or falling), lanes 0..2 will contain the preamble and lane 3 the normal SFD symbol. After this, the data from the MAC frame is transmitted on both edges of the clock, except that the preamble and SFD are skipped as they were sent already. The MAC frame data is placed accordingly to the rule described above. This continues until the end of the MAC frame. The Termination symbol is placed after the last byte of the frame. If the frame's length was a multiple of four then the Termination symbol is placed on lane 0 on the following cycle. Lanes that don't have MAC frame data or the Termination symbol are filled with the Idle symbol. At any point of the transmission, the Error symbol can be injected on any of the lanes. The Error sequence symbol may be used when the cause of the error is known. A local fault is signaled by setting (d_1, d_2, d_3) to $(0x00, 0x00, 0x01)$. A remote fault is indicated as $(0x00, 0x00, 0x02)$. All other values are reserved.

It maybe necessary to modify the length of the IFG in order to align the Start symbol to lane 0. There are two ways to do this. In the simple method, the IFG is extended by inserting additional Idle symbols to align the Start symbol correctly. This of course reduces the effective data rate. The alternative method is to use variable length IFG by maintaining a Deficit Idle Count (DIC). The DIC is incremented when an Idle symbol is removed from the IFG, and decremented when an Idle symbol is added to the IFG. The IFG shortening is constrained by the requirement that the DIC must be always between zero and three. This means that an IFG

may sometimes be three bytes shorter than the specified IFG. However, the frequency of the shortened IFG is bounded by the calculation rules. Consequently, the average IFG will not be lower than the specified IFG.

XAUI and XGMII Extender

The electrical signaling limits the usable distance of the XGMII to about 7cm [IEEE802.3, 2005, Clause 46]. For this reason, the standard specifies also the XGMII Extender Sublayer (XGXS) that is operable to distances up to 50cm [IEEE802.3, 2005, Clause 47]. Another clear benefit of using XGXS is that it reduces the number of interface signals as it uses serial communication.

The interface between two XGXS is called 10 Gigabit Attachment Unit Interface (XAUI). Figure 3.6 depicts how the XGMII signals (lanes) are converted to the XAUI signals and vice versa. The XAUI interconnect has 4 differential pairs for each direction. Each pair transmits the XGMII lane data as 8B/10B encoded, thus the bit rate on the XAUI signal pair is 3.125GHz. The source XGXS converts the Idle symbols into an 8B/10B code sequence. The destination XGXS recovers the clock and data from the XAUI lanes to the XGMII lanes.

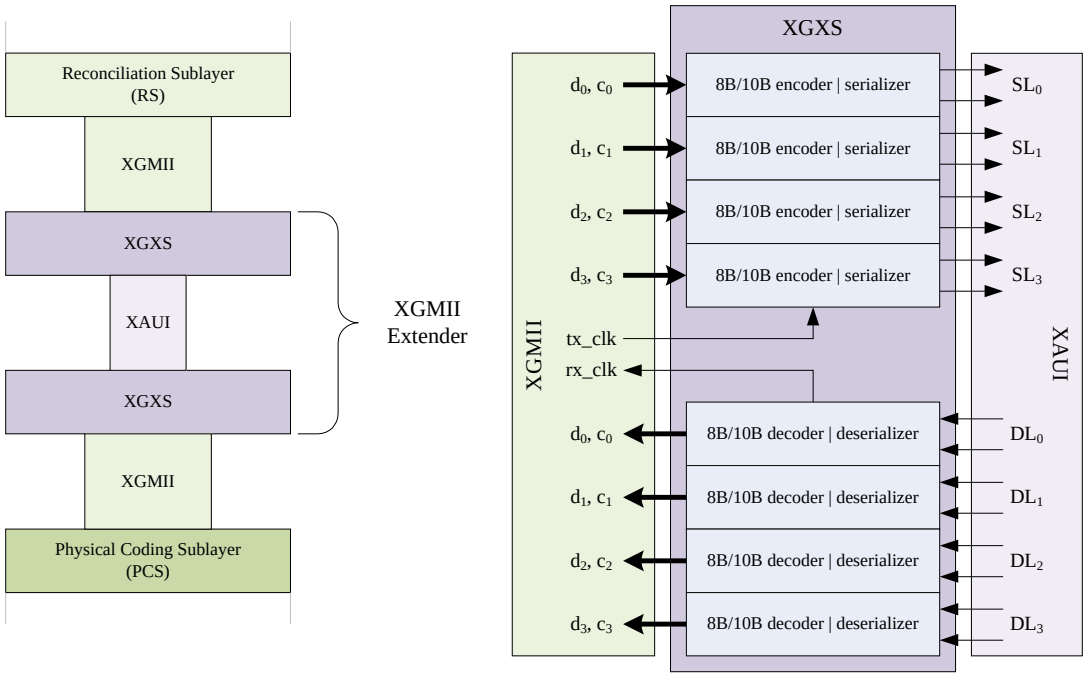


Figure 3.6: XGXS conversion between XGMII and XAUI.

3.6.2 Management interface

Management Data Input/Output (MDIO) Interface is used in controlling the PHY devices [IEEE802.3, 2005, Clause 22, 45]. The interface allows the MAC to query status of the PHYs and set control parameters. This includes, for instance, checking the status of the connection and setting the duplex mode. The interface has a master sourced clock signal `MDC` and bi-directional data signal `MDIO` as shown in Table 3.7. The interface allows up to 32 slaves be connected to the same bus. The clock `MDC` is driven only when necessary.

Table 3.7: MDIO signals.

Name	Width	Description
MDC	1	data clock (max 2.5MHz)
MDIO	1	data input/output (bi-directional)

The data transmitted on the MDIO has the frame structure shown in Figure 3.7. The preamble is a sequence of 32 logic one bits. The Start of Frame is a bit pattern 01b. The operation field may have bit pattern 01b or 10b. The first defines a read operation while the latter a write operation. The PHY and register address fields have both five bits and are used to identify the targeted PHY and the register in that PHY. The turnaround (TA) field is a 2-bit spacing to give time for turning the bus around in the read operation. In read operation the master sets its driver into tri-state on the first bit of the TA and allows the targeted slave (i.e. the PHY) to start driving the bus on the second bit of the TA to logic zero. The slave drives the 16-bit data field containing the value of the requested register before putting its driver into tri-state. In write operation the master transmits a bit pattern 10b during the TA and then sends the 16-bit data that should be written to the register address. After completing the MDIO frame transaction, `MDIO` is set to tri-state and `MDC` may be stopped. This is the idle state of the bus.

Preamble (32 bits)	Start of Frame (2 bits)	Operation (2 bits)	PHY Address (5 bits)	Register Address (5 bits)	TA (2 bits)	Data (16 bits)

Figure 3.7: Format of a Management frame.

3.6.3 Physical layer device

Physical layer device (PHY) is a device that implements the PCS, PMA, and PMD sub-layers of the Ethernet standard. The MAC sublayer uses the PHY through the Reconciliation Sub-layer via the Media Independent Interface and controls the behavior using the MDIO interface. Contrary to the MAC, the PHY is media dependent and thus changes according to the

used medium. Usually one PHY implementation can support only one medium type, although covering a range of speeds on that medium. PHY can support auto-negotiation feature that allows both ends of the link to choose the best possible operational mode between the PHYs [IEEE802.3, 2005, Clause 28, 37]. The details of the auto-negotiation function are not relevant for this work.

1000BASE

The IEEE 802.3 standard defines four different media types for Gigabit Ethernet. They are presented in Table 3.8 and Figure 3.8. The first three types are collectively known as 1000BASE-X and they share the same PCS and PMA sub-layers, and only the PMD sublayer changes depending of the medium. 1000BASE-T PHY contains the PCS and PMA sub-layers but doesn't utilize the PMD sublayer as it only supports one type of MDI, Category 5 unshielded twisted-pair (UTP) copper cabling.

Table 3.8: Media types for Gigabit Ethernet.

Name	Range	Description
1000BASE-LX	5km	Long-wavelength optical transmission on single- or multimode fiber
1000BASE-SX	550m	Short-wavelength optical transmission on multimode fiber
1000BASE-CX	25m	Shielded twisted-pair copper cable
1000BASE-T	100m	Cat. 5 unshielded twisted-pair copper wires

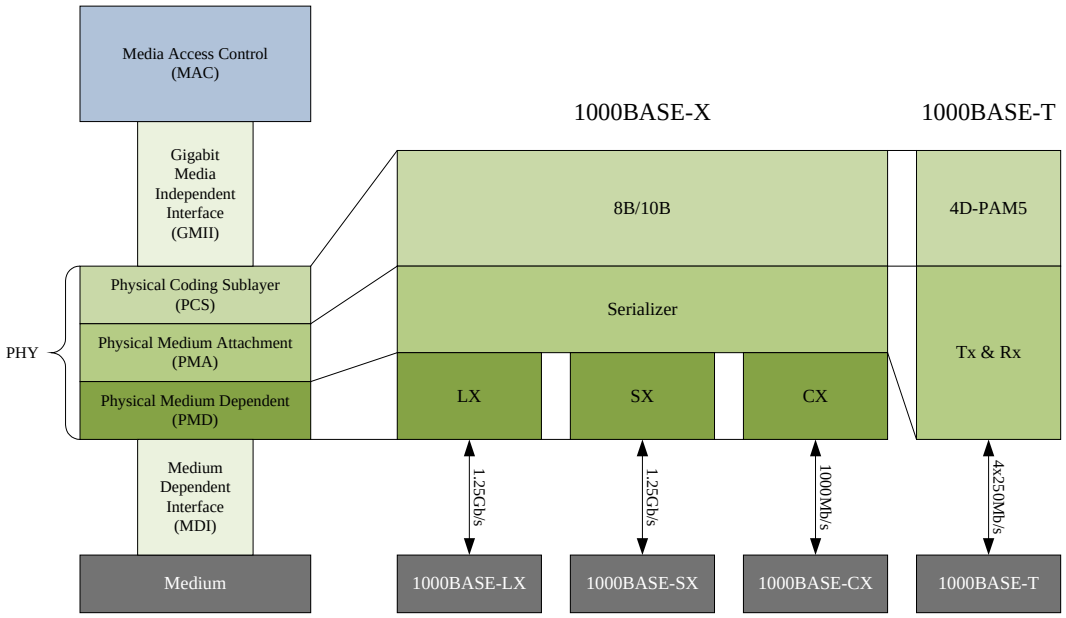


Figure 3.8: Different physical sub-layers in 1 Gigabit Ethernet.

10GBASE

The abbreviation jungle in 10 Gigabit Ethernet is even more complicated. The standard has three different PCS sub-layers, which are depicted in Figure 3.9.

- 10GBASE-R for serial 10Gbps operation (Clause 49)
- 10GBASE-W for serial 10Gbps operation, SONET/SDH compatible (Clause 50)
- 10GBASE-X for 4 lane parallel 10Gbps operation (Clause 48)

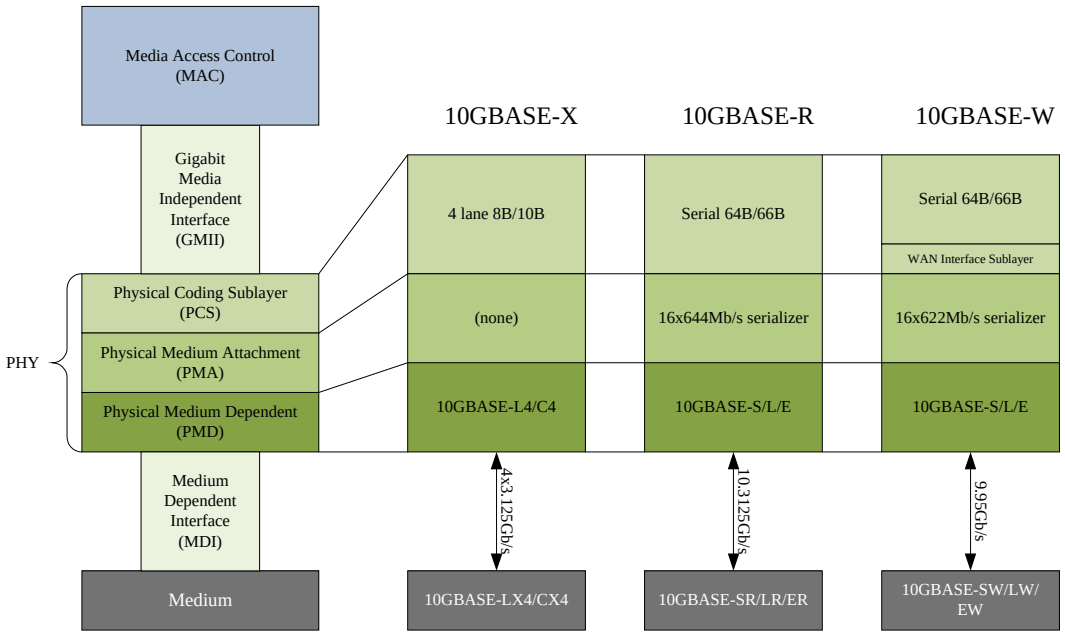


Figure 3.9: Different physical sub-layers in 10 Gigabit Ethernet.

In 10GBASE-R and W, the PCS sublayer does a 64B/66B encoding. They both use the same PMA (Clause 51) that serializes the data suitable for a PMD sublayer. The difference between these two is that the 10GBASE-W has an additional WAN interface sublayer (WIS) below PCS that allows transport of 10GbE data over traditional SONET/SDH infrastructure. SONET (Synchronous optical networking), used in North America, and SDH (Synchronous Digital Hierarchy), used in rest of the world, are protocol standards for synchronous data transmission over optical media. They are widely used between network providers and therefore an important target for interoperability. The R and W versions can operate on three PMD sub-layers: 890nm short-wavelength (“S”), 1310nm long-wavelength (“L”), and 1550nm extended reach (“E”). So, the following six different combinations are possible: 10GBASE-SR, LR, ER, SW, LW, and EW.

The PCS and PMA sub-layers of 10GBASE-X are identical to those in the XGXS (i.e. 8B/10B encoding). 10GBASE-X has two PMD sub-layers: multi-mode fiber ("L4") and 4 shielded twisted-pair copper cable ("C4"). The two possible combinations are 10GBASE-LX4 and 10GBASE-CX4.

10GBASE-CX4 is an especially interesting alternative. Because the PCS and PMA are the same as in XGXS, the signaling in CX4 complies with the XAUI interface. Therefore, provided that the transmitter and the receiver meet the specified electrical signaling, a XAUI capable MAC can directly connect to the 10GBASE-CX4 link without using any additional components. The downside of this low-lost alternative is that the cabling, which is the same as in Infiband, is impractical for most installations due to its inflexible nature.

IEEE802.3an [2006] specifies 10Gbps Ethernet over twisted-pair cabling, the 10GBASE-T standard. It operates on Category 6 UTP or shielded twisted-pair (STP) copper cabling. So far there have not been any products offering direct 10GBASE-T support. However, in April 2008, Broadcom announced the industry's first 10GBASE-T transceiver [Broadcom, 2008].

Transceiver modules

The optical transceivers are usually implemented in the form of a module. The modules are medium independent and hot-swappable, therefore allowing the end-user to choose the fiber medium freely and alter the configuration after purchase. There are several different module types standardized by the vendors with multisource agreements (MSAs). These standards define the mechanical and electrical interface of the module. The important point is that while the module types are not standardized by 802.3, they implement a 802.3 PMD sublayer and therefore provide compliance to the 802.3 standard.

XENPAK modules contain the PCS, PMA, PMD functions and provide the standard XAUI interface [XENPAK, 2002]. They offer easy interfacing and cheap host boards while being expensive and physically large modules. XPAK and X2 are physicall smaller successors of XENPAK. XFP module implements the PMA and PMD functions inside and has a XFI interface [XFP, 2005]. XFP are smaller and cheaper but require the MAC or other IC to implement the PCS functionality. SFP+ module is the smallest module type containing only the optics and the relevant drivers [SFP+, 2008]. The implementation of the PCS and PMA functions are left for other devices.

Chapter 4

Internal Architecture of the Controller

This chapter describes two different implementations of the controller that was outlined in 2.7. The first implementation is a straightforward solution targeted for 1Gbit Ethernet. The second implementation is a revised version offering a modular design that is capable of speeds up to 10Gbit.

4.1 First design

In the first version, the controller operates in 1Gbit Ethernet environment. The nodes are connected to an external switch as shown in Figure 2.19b. The controller has three ports connected to the switch in order to support wire-speed operation with encapsulation (see 2.7.1). Each of these ports have their own MAC address. The external networks are connected to the two other ports that do not have MAC addresses. The design uses full encapsulation where the original packets are prepended by a full MAC header. The header includes a node's and the controller's MAC address, a custom Ethernet Type, and the direction flag (left-to-right, or right-to-left). Actually, the direction flag is not completely necessary because the controller's MAC address could contain the direction information as well. The external switch must support jumbo frames. The controller supports VLAN tagged frames thus extending the maximum allowed frame size to 1522 bytes.

4.1.1 Overview of the operation

The top-level block diagram is shown in Figure 4.1. The external networks are connected to the ports on the left and right, and the external switch to the bottom ports. The internal data bus is 8-bit wide and all the controller's logic runs at the same 125MHz clock except the network interfaces, which are clocked by the PHY chips. The buffering is handled with FIFOs at the network interfaces. All the buffering is done with on-chip RAM.

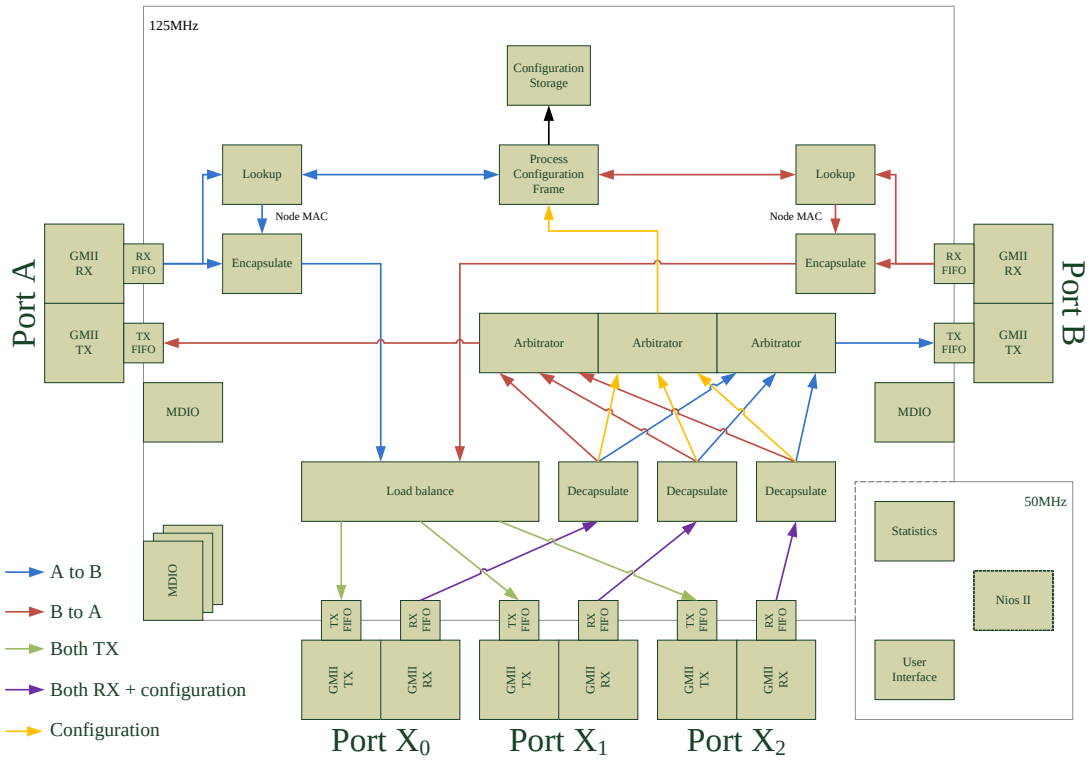


Figure 4.1: Top-level block diagram of the controller (first design).

The ingress port (e.g. Port A) verifies the received frame and puts the frame into its RX FIFO. When the Encapsulate unit notices the availability of the new frame, it begins to request an output port from the Load Balance unit. The Load Balance unit selects a free TX port and grants it to the Encapsulate unit. After receiving the grant, the Encapsulate unit begins copying the frame from the RX FIFO to the granted TX FIFO. At the same time, the Lookup unit begins to search the correct processing node that will process the current frame. Once the search is completed, the node's MAC address is handed to the Encapsulate unit. The Encapsulate unit will push the encapsulation header with the node's MAC address and one of the controller's MAC addresses to the TX FIFO. Because the TX FIFO has a separate channel for the header, the TX port can begin the transmission directly from the encapsulation header.

The external switch forwards the encapsulated frame to the processing node and then back after it has been processed. The RX port will verify the received frame and then push it into the RX FIFO. The Decapsulate unit will remove the encapsulation and verify the MAC address (non-promiscuous operation). The direction flag is now used to determine the correct output port. The Decapsulate unit will request access to the output port from the Arbitrator unit which will eventually grant the access. After receiving the grant, the Decapsulate unit will copy the rest of the frame to the TX FIFO of the output port. The TX unit will transmit the frame and generate a correct FCS field at the end. This concludes the path of a single frame.

4.1.2 Network interfaces

A network interface contains separate transmit (TX) and receive (RX) units implementing the MAC layer of Ethernet. The design uses GMII to connect to the external PHY. RGMII is supported with an adapter that translates GMII signals to RGMII signals and vice versa. The interface supports Ethernet flow control (i.e. PAUSE) on the bottom interfaces. The external interfaces are not equipped with flow control by default because then the system would not fulfill the transparency criteria.

Both the TX and RX units have their own CRC32 unit to generate and verify frame checksums (FCS). The checksum formulas are generated with the on-line CRC tool at `www.easics.be`. The checksums are calculated during transmission and reception without buffering, a 8-bit word in a clock cycle.

Each network interface has also an associated MDIO unit. The unit configures the PHYs and monitors the state of the network link periodically. This information is used by the Load Balance unit to detect link losses.

4.1.3 Lookup unit

The Lookup unit is based on a state-machine. The frame data is inspected as it flows and the critical fields are extracted to registers. Once all the required information is gathered, the unit will query for the MAC address of the selected processing node. The configuration query takes two clock cycles as the access to the configuration memory is interleaved among the two Lookup units.

4.1.4 Load balance unit

The Load Balance unit has two input ports and three output ports. Primarily, the left input port gets access to the left output port, and the right input port gets access to the right output port. The middle output port is used as an overflow port for the both inputs when their own output ports are incapable of accepting more frames. Without this arrangement, the all three Decapsulate units could have frames destined for the same output port while the other output port would be idle. In other words, the arrangement eliminates the HOL blocking. The unit is also capable of surviving a link loss from one of the network interfaces although resulting in degraded performance.

4.1.5 Arbitrator unit

The Arbitrator unit has two inputs and one output. It grants the accesses in round-robin fashion so that the last user has the lowest priority. The arbitration takes one clock cycle. A bus multiplexer is included inside the unit.

4.1.6 Configuration

The information about processing nodes, including their MAC addresses, are given by using custom configuration frames. The Decapsulate unit will detect these configuration frames by examining the Ethernet Type field. Similarly to encapsulated frames, an Arbitrator unit is used to request access to the Process Configuration Frame unit. That unit will inspect the configuration frame and update the internal memories accordingly. If the controller does not have a valid configuration, the traffic can be configured to flow directly through left to right and vice versa.

4.1.7 Control interface

An optional control interface is provided with a Nios II CPU, which is Altera's soft-core RISC-processor. The software implements a user-interface via a serial port and also allows statistics to be shown on the connected LCD display. The statistics are gathered from network interfaces and from Decapsulation units.

4.2 Revised design

The first design provided a fully functional controller that was capable of 1Gbit speeds and met the requirements. However, the design was not flexible enough for higher speeds or different port counts. That is why a more modular design was needed.

This new revised design is targeted for 10Gbit Ethernet with direct connection to the processing nodes and supports cascade topology. The design is modular so that the parts of the design can be modified and changed easily. The design is also parameterized to provide configurability at the synthesis phase. For instance, the number of ports and the switch behavior is controlled by design parameters. The buffers in the design can use on-chip or external memory. Load balancing, if required, can be implemented within a so called packet engine. Beside the configurable port count, also the roles of the ports can be chosen freely, and are not restricted to the previously defined roles of external and internal.

4.2.1 Overview of the operation

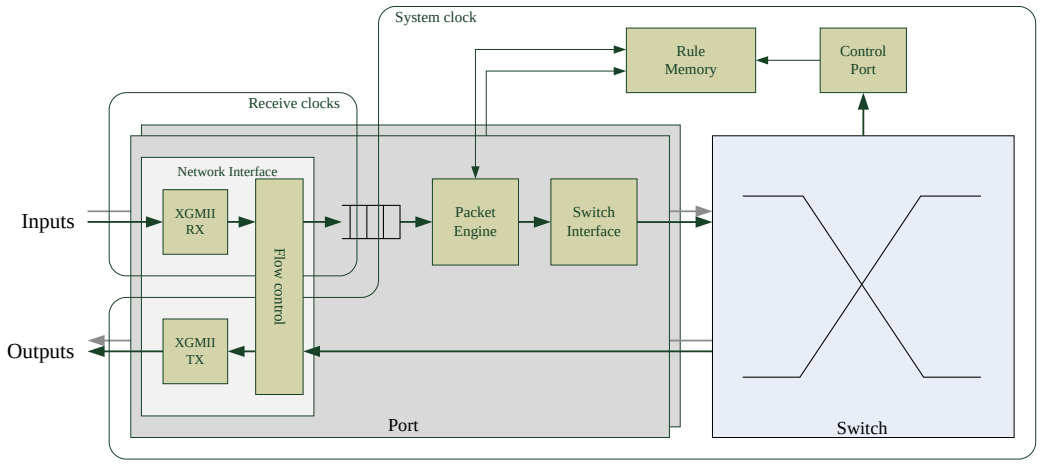


Figure 4.2: Top-level block diagram of the revised controller.

The top-level block diagram is presented in Figure 4.2. The design consists of N ports, a control port, and a (N+1)-port switch fabric. The control port can only receive frames and is used for configuring the rule memory. The rule memory contains information about the processing nodes. The system has N+1 clock domains, which are marked with a dotted line in the figure. The receiver logic is clocked by the PHY. Other logic, including all the transmitters, are clocked by the system clock. The FIFO is used as a synchronization point between a port's receive clock and the system clock.

The frame bus (bold arrows in the figure) has 64-bit data lane and 8-bit control lane. Its target speed is 156.25MHz, which provides throughput equivalent to 10GbE. The signaling in the frame bus resembles the XGMII interface.

The frame is received by the XGMII RX unit. The unit verifies the frame (FCS, runt, PAUSE) and pushes it to the FIFO. As in the first design, the FIFO is capable of discarding invalid frames.

The packet engine reads the frame data from the FIFO and processes it. This process is application dependent and can include, for instance, VLAN untagging (decapsulation), header extraction, classification, and VLAN tagging (encapsulation). Once the result, the destination port number, is known, the frame data is pushed forward to the switch interface. The switch interface requests the destination port from the switch fabric. After getting the grant, the frame data is pushed to the switch fabric.

The switch fabric forwards the frame to the wanted destination port. The switch fabric ensures a certain period between the frames for efficient IFG generation.

After the switch fabric, the frame goes directly to the XGMII TX that transmits the frame with a generated FCS. There is no buffering in the TX, thus the needed buffering must be implemented inside the switch.

The packet engine is where all the traffic management happens. For external ports, the packet engine decides which processing node will get the frame. The frame is encapsulated so that it contains at least the direction information and, in non-direct configuration also the processing node information. For internal ports, the packet engine checks the encapsulation and decides where to forward the frame. The options are one of the external ports, or the control port if the frame contains node configuration data.

4.2.2 Internal data format

The frame data is transferred internally with the format that closely resembles the XGMII data format (see 3.6.1). The bus is 64-bit wide containing eight data lanes, each having 1-bit control signal and 8-bit data. The control signal is asserted when the associated data lane has valid data. The bus is in idle state when none of the control signals are asserted. At the start of a frame (SOF), the first byte of the frame data is placed on the first data lane and then on the

following lanes. The frame data is transferred in consecutive words so that all the lanes contain valid data. At the end of the frame (EOF), depending on the frame's length, the lanes may be only partially filled. The bus also assumes that the frame's length is over 2 words, 16 bytes, which is not a restriction with Ethernet traffic.

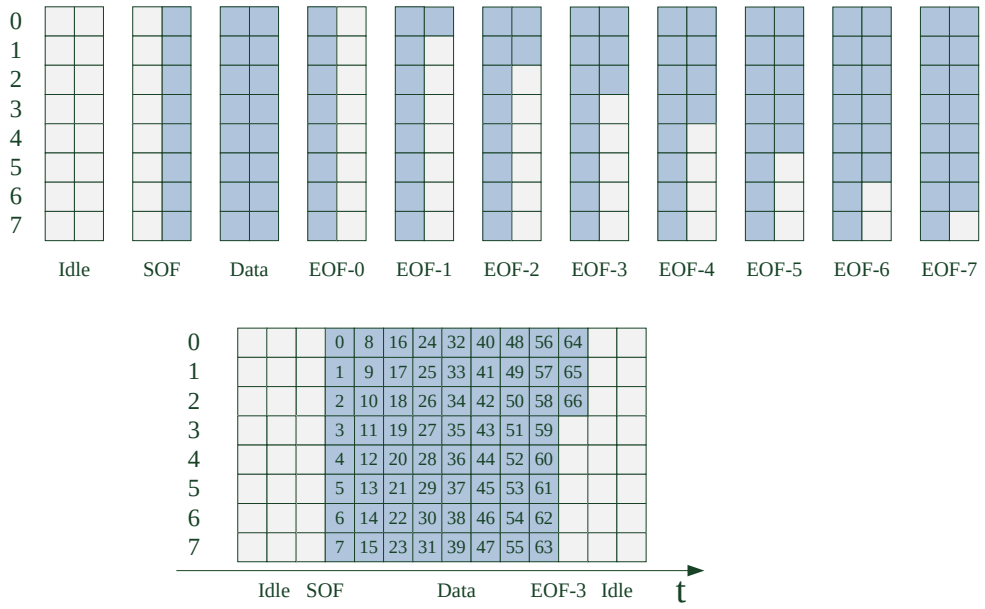


Figure 4.3: All the valid control signal sequences (top) and an example case with a 67-byte frame (bottom).

All the valid control signal sequences are shown in the top part of Figure 4.3. The blue color indicates that the control signal is asserted. On each sequence, the stripe on the left is the previous state and the right is the current state of the control signals. There is only a single pattern for SOF. Therefore the SOF can be detected from a rising edge of any control signal. There are eight different patterns for EOF case, named EOF-n where n is length $\bmod 8$. The EOF detection can be based on a falling edge of the last control signal. Idle case is detected when the control signal 0 is not asserted. The bottom part of Figure 4.3 shows an example of frame transmission. The numbers inside the boxes represent the byte order of the original frame. The frame has a length of 67 bytes, so EOF-3 pattern is seen at the end.

4.2.3 Network interface

The network interface implements the XGMII standard (see 3.6.1). The DDR-to-SDR conversion is done before the block so the incoming and outgoing XGMII signal has 8 lanes and is 64-bit wide. Other interfaces, such as GMII, are supported by using an external adapter.

The internal structure of the network interface is depicted in Figure 4.4. The block contains a transmitter, a receiver, and a flow control unit. The dashed horizontal line shows the clock boundary between the receive clock and the system clock.

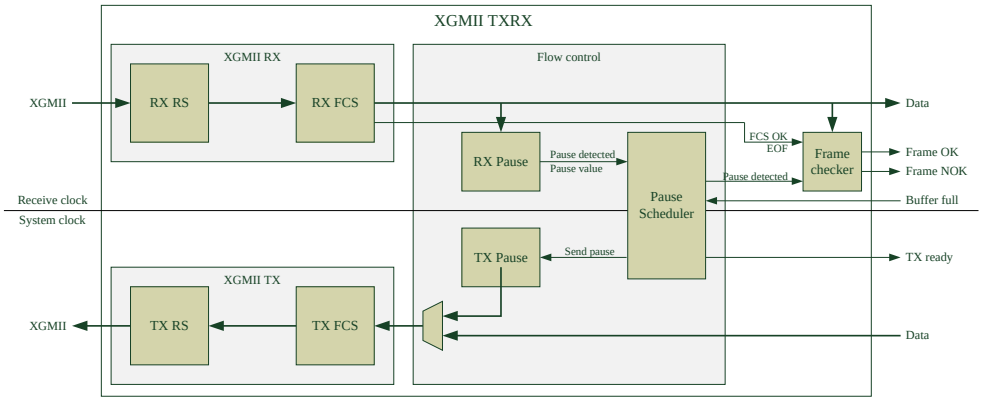


Figure 4.4: Block diagram of the XGMII transceiver.

XGMII RX

The RX RS (reconciliation sublayer) unit translates the incoming XGMII symbols to the internal data format. This procedure also removes the XGMII start symbol, the MAC preamble bytes, and the XGMII terminate symbol, thus outputting only the MAC header, payload, and the FCS field. The RX FCS unit calculates the FCS over the whole frame (including the FCS field), and compares the calculated FCS with the correct value of 0xdebb20e3 at EOF. The unit also removes the trailing FCS field.

XGMII TX

The transmit side does the opposite of the receiver. The TX FCS unit calculates a checksum and appends it to the frame. The TX RS unit translates the internal data format to the XGMII symbols. It generates the XGMII start symbol and the following seven MAC preamble bytes at SOF. At EOF, the XGMII terminate symbol is added. In idle state, the unit transmits the XGMII idle symbols.

The unit uses adaptive IFG scheme and thus implements the DIC algorithm. The port scheduler is informed about changed IFG via the `tx_ready` signal.

Flow control

The flow control unit implements the Ethernet flow control as described in 2.1.4. The incoming data is monitored by the RX pause unit that detects valid PAUSE frames and extracts the pause time parameter. The pause scheduler deasserts `TX ready` when a pause is requested, and keeps it deasserted until either the pause time has expired or a zero pause time has been received.

The pause scheduler also controls the PAUSE frame sending. When `buffer full` is asserted, the pause scheduler deasserts `TX ready` and waits until the current transmission is finished. Then it orders the TX pause unit to send a PAUSE frame with a selected pause time. After the send operation, `TX ready` is restored.

CRC-32

Both the TX FCS and the RX FCS have an instantiation of the same CRC-32 unit. The unit is capable of calculating a CRC-32 round in a cycle for variable-width data. As shown earlier, the data on the internal bus always starts at lane 0 but may take the next 1 to 7 lanes. Therefore, the data width varies from 8 to 64 bits in 8-bit increments. The internal structure is shown in Figure 4.5.

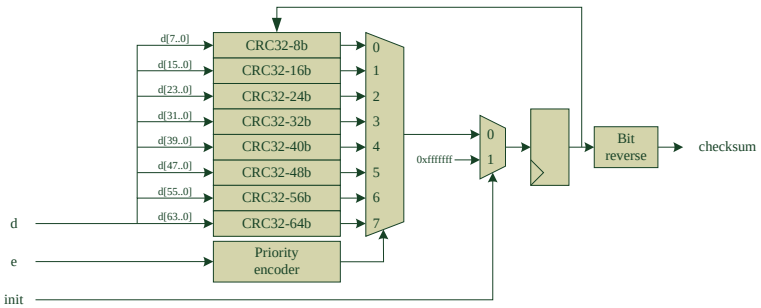


Figure 4.5: Block diagram of the variable-width CRC-32 engine.

At SOF, the CRC-32 engine is initialized with a value of 0xffffffff. The engine contains eight separate CRC-32 cores, one for each possible data-width. The correct result is then selected by using the control signals. The output of the CRC-32 unit is bit-reversed according to the Ethernet specification. The checksum XOR-equations for the CRC-32 cores are generated with the on-line CRC tool at `www.easics.be`.

4.2.4 Packet engine

The packet engine is the most import customization point of this design. It decides where to forward the incoming frame and can modify the frame contents. The following describes a configuration that was used in this work. Here, two different roles were identified for the packet engine: one for external traffic and one for internal traffic. Both can be implemented with the structure depicted in Figure 4.6.

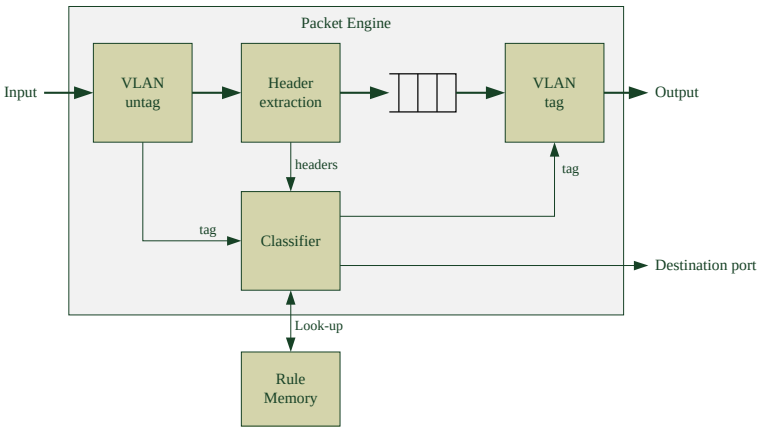


Figure 4.6: Generic packet engine with configurable cores.

The external traffic is processed in packet engines that are connected to the two external ports. The header fields are first extracted from the incoming frame. Then the classification unit analyzes the extracted header fields, makes a look-up to the rule storage, and finally decides the output port. After the classification, a VLAN tag is inserted into the frame. The tag's ID defines the ultimate output port, i.e. the other external port. The classification engine can also decide to forward the frame directly to the other external port to allow controller cascading. However, the frame is not tagged in this case.

The packet engine that is targeted for internal traffic assumes that all valid incoming frames have a VLAN tag. Therefore, before header extraction, the VLAN tag is removed from the frame. The classification unit uses the tag's ID when deciding the output port. The tag was originally inserted by the packet engine at an external interface. A special tag is reserved for configuration frames that are always forwarded to the internal command port. No new tag is inserted.

The following shows the details of the sub-entities of the packet engine.

VLAN untag and tag

When enabled, the VLAN untag unit removes a VLAN tag from the frame and extracts the tag's parameters. If the frame does not have a VLAN tag, the frame is passed through without modifications. The unit has a constant latency of one clock cycle. The MAC addresses stay in place but the data after the address fields is rotated by 32-bits due to the removal of a 16-bit Etype field and a 16-bit VLAN tag field.

The VLAN tag unit inserts a new VLAN tag when enabled. The unit does not add latency. However, the data after the MAC addresses is delayed by 32-bits due to the insertion of the new VLAN tag.

Header extraction

The header extraction unit simply stores the header fields of the incoming frame to a set of registers. These registers are valid from the moment the complete header is seen until the beginning of the next frame. The validity is signaled by asserting `hdr_valid`.

Classifier

The classifier chooses the frame's destination and decides whether to insert a new VLAN tag or not. The decision is based on the input from the VLAN untag unit (the tag, if any) and from the Header extraction unit (the frame header values). After the information has became valid, the classifier starts to process the received information. During the process, the classifier may do look-ups to the rule memory.

The total time from the SOF to the moment where the outputs are valid will be limited by the header extraction (h cycles), the tag extraction (t cycles), the maximum rule memory access time (r cycles), and the internal processing time (p cycles). Because h and t are overlapped, and $h > t$, the total will be less than $h+r+p$. If the header extraction provides the MAC header (112 bits) and the IPv4 header (160 bits), the delay h will be six cycles. The rule storage can serve one query per cycle. If there are five ports all accessing the rule storage, the worst-case delay r will be six cycles. The processing can be done in a cycle, so p is one clock cycle. Thus, the total worst-case delay will be 13 clock cycles.

Delay buffer

The frame data is delayed with a buffer to give time for the classifier. The minimum size of the buffer is defined by the classifier's latency. Based on the above calculations, a 16-slot buffer is suitable. Due to the small size, this can be implemented with registers instead of memory cells.

4.2.5 Switch interface

The integration of the packet engine to the switching fabric is handled in the switch interface unit. The unit's task is to act as an adapter in front of the switch, thus hiding the implementation details and any interface changes in the switch fabric. With the current design, the unit simply connects the signals together without any adaption. This is possible because the switching fabric and the packet engine have been designed concurrently.

4.2.6 Switch

The switch module implements a switching fabric that connects the input ports to the output ports. Because of the small number of ports and the performance requirements in this application, the designed switching fabric is based on the fully interconnected architecture (see 2.5). The modular design allows any other switching architectures to be used, though. The implemented switch is non-blocking and supports multi- and broadcast frames.

Switch fabric

The overview of the switch is shown in Figure 4.7. The switch contains N input ports and N output ports. Only one data transfer can be active per output port, but up to N data transfers can be active per input port. The switching fabric is not actually fully interconnected as some of the connections are unused and therefore optimized away during synthesization (see 4.2.7). Thus, the architecture could be called a partially interconnected architecture. The switch is operated by the system clock, i.e. there is no speed-up, and thus no requirement for output buffering.

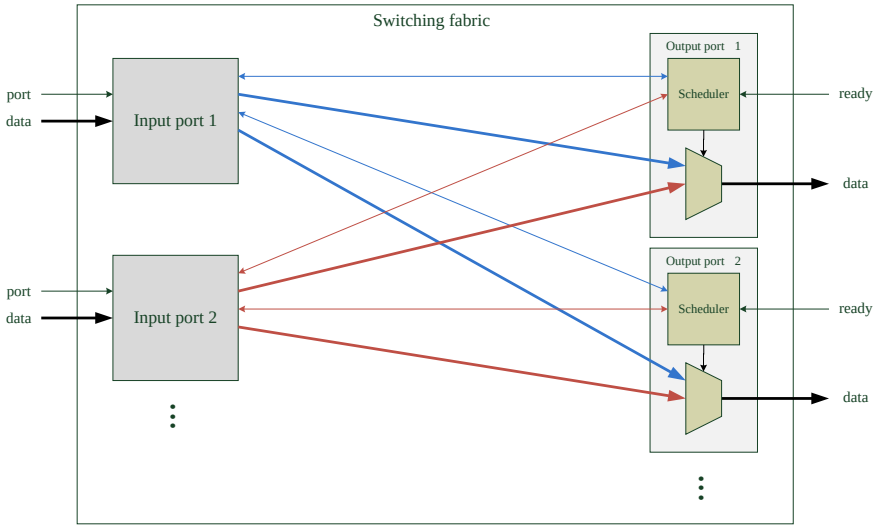


Figure 4.7: Overview of the switch fabric.

Output port

Each output port has a scheduler, that gets N request signals from the N input ports, and an N-to-1 72-bit multiplexer. The scheduler ensures a fair arbitration between the users, i.e. the input ports, and controls the associated multiplexer.

The scheduling is based on a round-robin algorithm that gives the lowest priority to the last user. The scheduler also ensures the proper IFG period between the frames by monitoring the `tx_ready` signal and delaying the grants. When the ready signal is deasserted by the TX unit, the scheduler does not grant any user. This basically enables flow control and causes back-pressure to the input buffers.

Input port

The input port receives incoming data at wire-speed. The destination port information is given with the frame data. The HOL problem was solved by using virtual output queuing (see 2.5.2). The internal architecture of the input port is shown in Figure 4.8.

The incoming data is first written to a FIFO that corresponds to the destination port. If the FIFO is full, then there is two options. Either inform the user (in this case, the port engine) to create back-pressure, or just silently ignore the frame. The latter was chosen because using back-pressure would just relocate the HOL problem to the earlier stages.

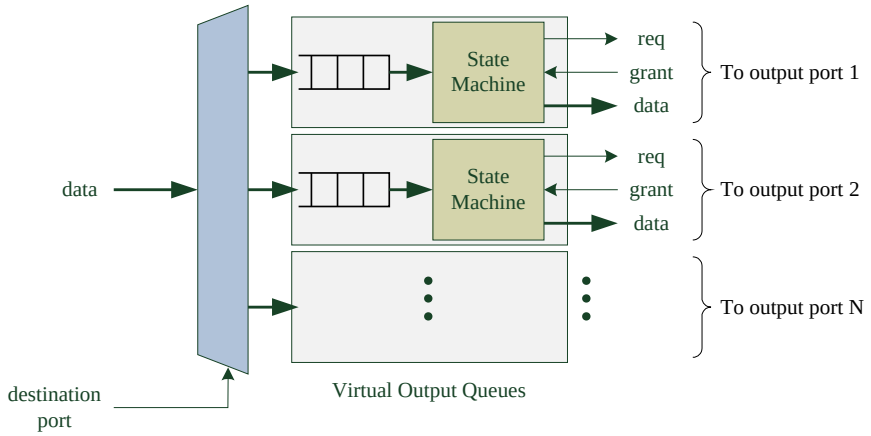


Figure 4.8: Input port with virtual output queues.

Each FIFO has a state machine that does the following. When the FIFO is not empty, a request signal is sent to the port scheduler of the corresponding output port. When a grant is received, the frame is copied from the FIFO to the output port. There is no need to wait for the FIFO to contain the full frame as the incoming data rate (i.e. write speed) matches the read speed, thus eliminating the problem of underflow. Note, that the internal data bus does not allow gaps in the middle of a frame.

Because each FIFO has its own state machine and output buses, the maximum total data rate is the port count times the wire-speed. The FIFOs are implemented with on-chip RAM and their sizes are configured during synthesization (see 4.2.7). Furthermore, a VOQ without any output connections will be removed in the synthesis stage.

Input port with external memory

To enlarge the buffering capacity, an input port with external memory was designed. Here the virtual output queues are implemented by creating logical ring-buffers in the external memory. The target was to use QDRII+ memory but any other memory type should be usable as long as the performance is enough for wire-speed read and write.

In principle, the concept is simple. Write the incoming data to a correct ring-buffer location, and on the other side, read the data from the memory and copy it to the output port. However, the read latency raises some interesting problems. With on-chip RAM, the reading was controlled by the output of the FIFO. With off-chip RAM, this is not possible as the control would be severely delayed due to the latency.

The used QDRII+ IP memory controller has a constant read latency of 15 cycles. In other words, the data will be available 15 cycles after the read request (i.e. the read address). This is equivalent to the transfer time of two minimum-length Ethernet frames. Even if the boundary information, for example the frame's length, would be at the beginning of a frame, the control would not be fast enough for small frames.

There are several strategies that can be used to circumvent the latency problem:

- Store the frame lengths or the begin/end pointers to an on-chip FIFO. Use that information to control the read operation.
- Add N small secondary on-chip buffers after the memory. When an EOF is detected, notify the memory controller and store the already requested words to the buffer. Read the buffer as in the on-chip memory case. Each buffer should have space for `read_latency - minimum_frame_length` words.
- Add N secondary on-chip buffers after the memory to act as a temporary storage. Keep the on-chip buffers as full as possible by reading data from the external memory. The buffer size should be greater than `2*maximum_frame_length + read_latency` words for efficient operation.

The first option considerably increases the amount of on-chip memory needed for book-keeping, and the amount is also proportional to the capacity of the external memory. The second option has the smallest memory overhead but requires a constant wire-speed read performance and does not utilize the property of multiple simultaneous transfers offered by the used switching fabric. The constant read-rate is achievable with SRAM memories but not with DRAM memories (see 3.4.2). The third option offers similar performance as the on-chip-only input port while being tolerant for read-rate variations. It was therefore selected for the implementation.

The whole architecture with the external memory as the primary VOQ storage and the on-chip buffers as the secondary VOQ storage is illustrated in Figure 4.9. The primary VOQs are located in arbitrary, non-overlapping, contiguous locations in the external memory. A VOQ is treated as a normal ring-buffer with a read and a write pointer. The pointers are stored in registers. Because a VOQ's size is not necessarily a power-of-two, the looping must be handled manually by reassigning the start address when the last address is accessed. Similarly, the used space counter cannot be calculated by simply subtracting the read pointer from the write pointer because it would require arbitrary modulo arithmetic. Instead, the used space counter is implemented as a step-up/down counter, that a) increments the count on a write operation,

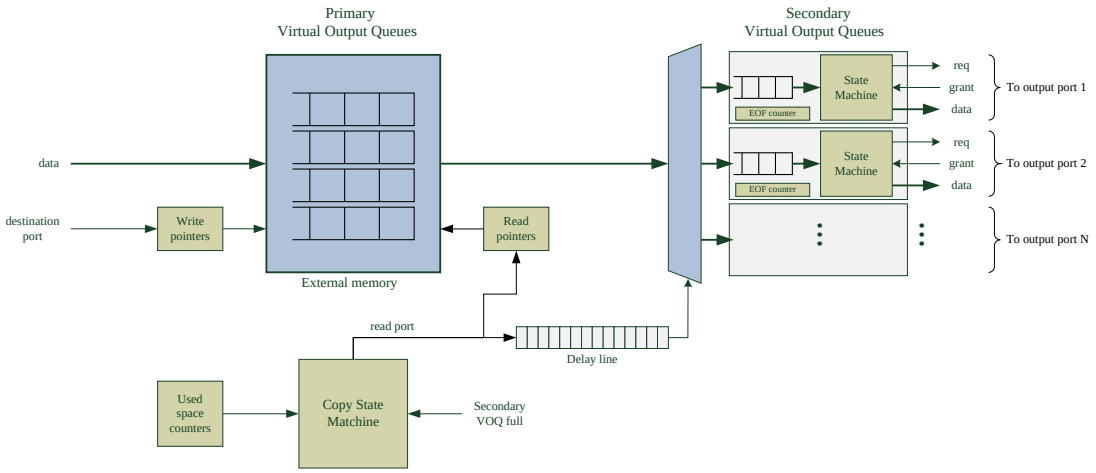


Figure 4.9: Input port with an external memory for primary VOQ storage.

b) decrements the count on a read operation, and c) does nothing when both read and write are active or inactive.

The secondary VOQs are similar to the VOQs used in the on-chip-only input port. The only exception is that the state machine monitors the amount of EOFs in the buffer. The output process is started once there is at least one EOF, i.e. a full frame, in the buffer. This ensures that there are no gaps in the outputted frame due to buffer underflow. The EOF counter is implemented as a step-up/down counter as above.

The Copy State Machine controls the reading of the external memory. A memory read is issued when there is data available in a port's primary VOQ and the port's secondary VOQ is not full. The full signal is triggered 15-words too early to take the pipeline effect of the read requests into account. Also, due to the read latency, a delay line structure constructed from registers is used to delay the port information so that when the read data is valid, the data is pushed to the correct secondary VOQ.

4.2.7 Design configuration

The implemented design includes numerous parameters that can be used to customize and tune the design to various configurations. The customization is achieved by using VHDL's `for..generate` statement [Ashenden, 2002]. For example, by adjusting the port count parameter, the synthesizer will synthesize a switching fabric of the correct size and generate enough port entities.

The internals of the switching fabric are tuned with a NxN matrix called the traffic matrix. This matrix defines the traffic probabilities between the ports. The VOQ buffer in Port *i* that outputs to Port *j* is proportional to the value of cell (*i*,*j*). Furthermore, if the value is zero, then the whole buffer is eliminated and the output signals are set to zero. For the synthesizer this indicates that the specified path will be never used and thus it can optimize the whole path away. This results in massive optimizations in the switching fabric and effectively transforms the fully interconnected architecture to a partially interconnected architecture that is tuned to the specific traffic pattern.

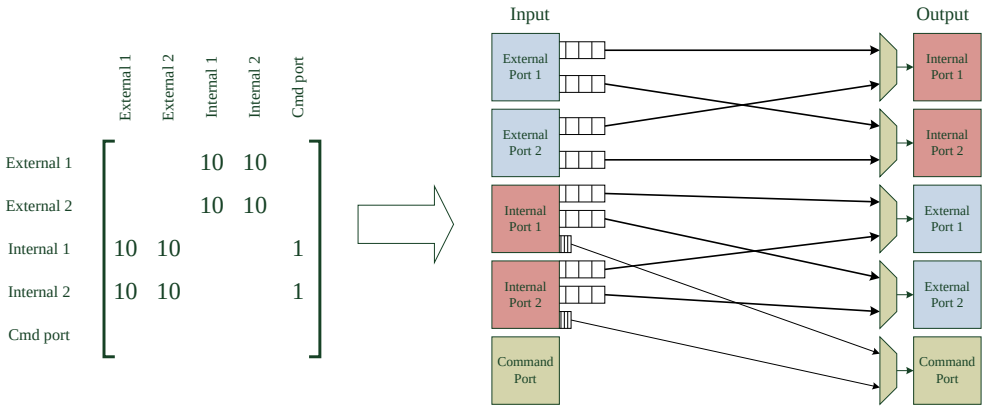


Figure 4.10: The traffic matrix and the resulting switching fabric.

The effect of the traffic matrix is illustrated in Figure 4.10. In this example case, there are five ports, two for external traffic, two for internal traffic, and one command port. The command port will not transmit anything, so all the cells on row 5 are zeros. Because no loopbacks are allowed, cells (*i*,*i*) are zeroed. The external ports will send only to the internal ports and vice versa, except that the internal ports may also send to the command port, although a small amount. The resulted switch fabric is shown on the right of Figure 4.10. In this example, the amount of interconnections was reduced from 25 to 10, multiplexers have transformed from 5-to-1 to 2-to-1, and also the memory usage is optimized.

In this design, the roles of the port engine and the flow control mode are also configured with a matrix, called the port configuration. The configuration can enable or disable VLAN tagging and untagging, define the tag's value, or set a pass-through destination if the the node configuration is unavailable.

During run-time the configuration can be altered via the command port. For instance, the information about available processing nodes can be uploaded through the command port to the rule storage.

4.2.8 Data path and buffering

The stages that a frame faces during the process inside the controller are summarizes in Figure 4.11. The figure also shows the three buffers used. Frame data is first pushed to the CDC buffer that is responsible of data synchronization to the system clock domain. The buffer's size must be at least twice the maximum supported frame, because the frame is not read from the buffer until it has been completely verified. The process delay buffer delays the frame data to give time for the frame classification. Its size is comparable to the size of the processed header. The main buffering is done in the switch at the input ports. The buffers follow the virtual-output queuing method and therefore don't suffer from HOL.

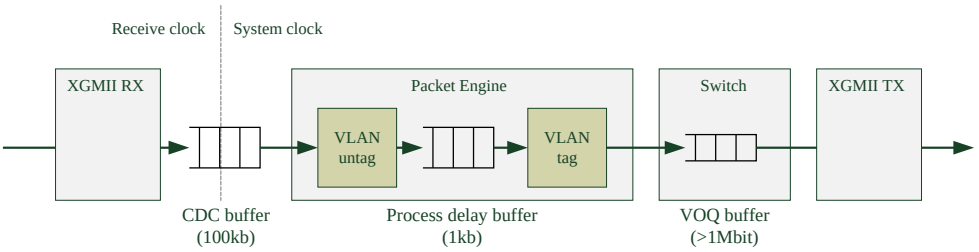


Figure 4.11: Buffering and processing stages on the frame's data path.

Data congestion happens when a) there are two or more ingress ports trying to access the same egress port, or b) the egress port is paused due to the reception of an Ethernet PAUSE frame. If the congestion is prolonged and there are more incoming frames, the VOQ buffer will eventually become full. The switch's ingress port may then either silently ignore the incoming frames for the full egress port or request that a PAUSE frame is sent to the originating port.

While the pause tactic reduces the possibility of a frame drop, it will effectively create a case which resembles HOL blocking: the ingress port will not receive any frames during the time the PAUSE is active. This is very problematic as typically only a single egress port is causing the congestions while the other ports are functioning fine. On the other hand, the controller's buffering capacity is very limited compared to the amount of memory available to the processing node. This speaks up for the PAUSE tactic, i.e. to use the node's memory for additional buffering. Then again, the silent frame dropping tactic can greatly increase the aggregated throughput by allowing the RX ports to operate continuously. If the congestion is only temporary, the PAUSE tactic should be the best alternative, because the frame dropping generally causes serious degradation to the connections. The implementation supports both alternatives.

4.2.9 Latency analysis

Latencies from every design unit in the data path are depicted in Figure 4.12. The buffering delay, caused by the store-and-forward operation, depends on the frame length and thus dominates the total latency with larger frames. The other significant part of latency is caused by the delay buffer, which is needed for classification and for rule memory arbitration.

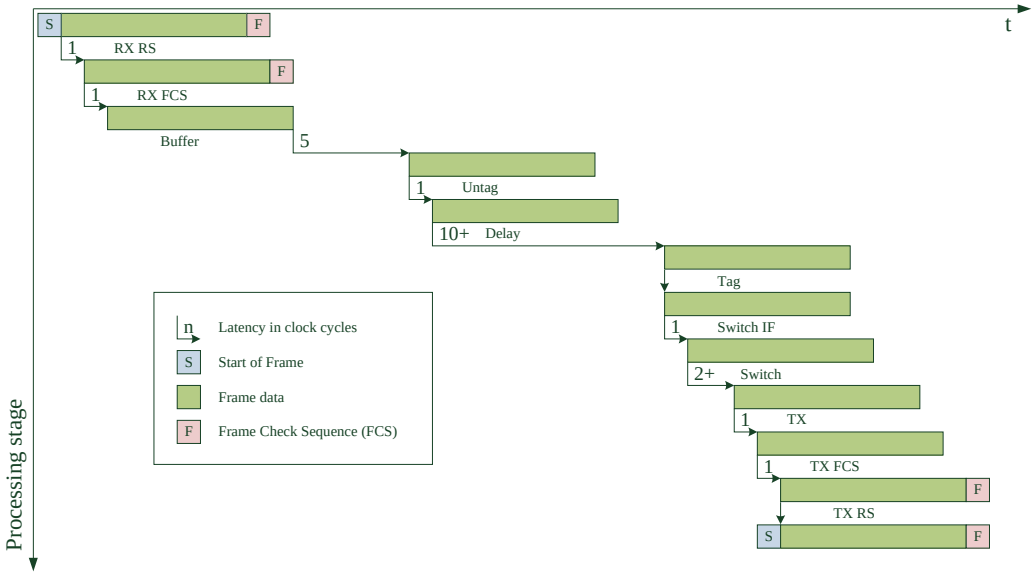


Figure 4.12: Diagram of the data path latency.

The total minimum latency in terms of clock cycles is

$$\text{latency} \geq 28 + \frac{\text{frame\_length}}{8} \tag{4.1}$$

where `frame_length` is given in bytes. The latency may be longer because of four reasons. Firstly, the classifier can not make decision in the minimum time due to a congestion in the rule memory. Second, the VOQ buffer is not empty. Third, there is a congestion to the output port. Fourth, the transceiver must temporarily increase IFG due to the DIC algorithm.

Chapter 5

Implementation Verification and Performance

This chapter describes the prototyping environment, including the FPGA platform, and the used hardware and software tools. The design verification steps are explained. The synthesis results are analyzed and suitable target FPGA devices are listed. Finally, the controller running on the prototyping platform is measured in terms of latency and throughput.

5.1 Prototyping environment

For the development platform, a generic development kit from Altera was chosen. The kit contains a board that has a Altera Stratix III (EP3SL150F1152C2) FPGA and a 1GbE PHY. With a 4-port 1GbE extension card, the total port count is increased to five, which was enough for all planned test scenarios. Figure 5.1 shows the complete platform. Stanford's NetFPGA [Gibb et al., 2008] is a popular prototyping platform in academia. Unfortunately, having only four 1-gigabit ports makes it unsuitable for this thesis' application.

The on-board PHY (Marvell 88E1111) is connected to the FPGA and supports SGMII, GMII, and RGMII interfaces [Altera, 2007b]. The extension card has a quad-PHY (Marvell 881145) and supports either SGMII or RGMII [MorethanIP, 2008]. RGMII was chosen here for all PHYs as it is the simplest one to set-up while still having the same performance. The board also has two displays that are used for showing the traffic rates.

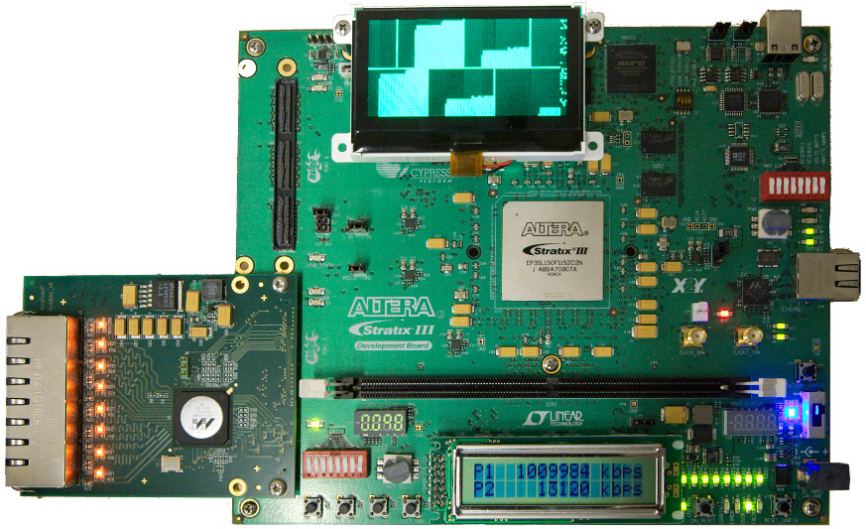


Figure 5.1: Stratix III development kit with 4-port extension board.

5.2 Verification

Verification is performed on three levels. First, the logic design is tested with RTL simulation. Then, the whole design is run within a synthesizable testbench on a FPGA. Finally, the design is tested in a real environment with real PHYs and traffic. The final test is limited to 1Gbps operation speed because of the PHYs.

5.2.1 Simulations

The top-level testbench is depicted in Figure 5.2. A node model sends configuration frames to the controller and loopbacks incoming data. A network model generates random frames, transmits these frames, and verifies the incoming frames. A generated frame contains a PRNG seed of the transmitter, which is used in verifying the received frames.

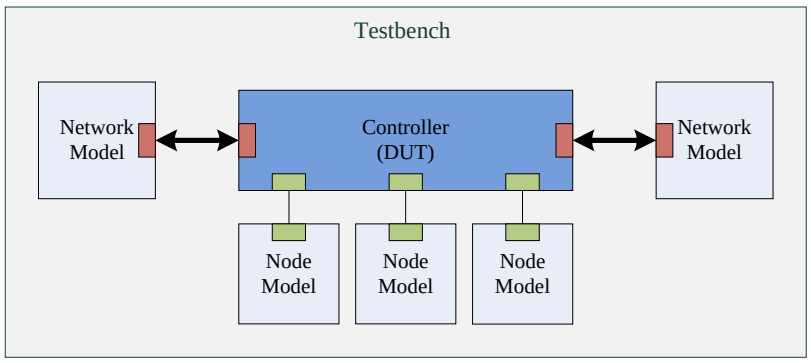


Figure 5.2: Testbench setup with the controller and the node and network models.

In addition to the standard VHDL assertions of the testbench, the design units have PSL statements to verify critical assumptions and conditions, such as buffer under- and overflows. These conditions are invaluable when locating logical or interfacing errors. To get real Ethernet traffic to and from the simulation environment, a VHDL code was written to record and replay frame captures in libpcap format. This file format allows for exchange of data with tcpdump and Wireshark network tools.

5.2.2 Synthesizable testbench

Instead of the traditional approach of using behavioral level for testbench components, the node and the network models were written at the RTL level. Although this is more difficult, it allows the testbench to be synthesized and run on the FPGA at full-speed, hence the name "synthesizable testbench" (STB). The speed is a significant advantage over a standard testbench simulation, which is relatively slow (e.g. 1ms of simulation takes 100s of computation time). Because the STB lacks the debugging capabilities that the simulator has built-in, any error conditions detected in STB must be replicated in the simulation environment.

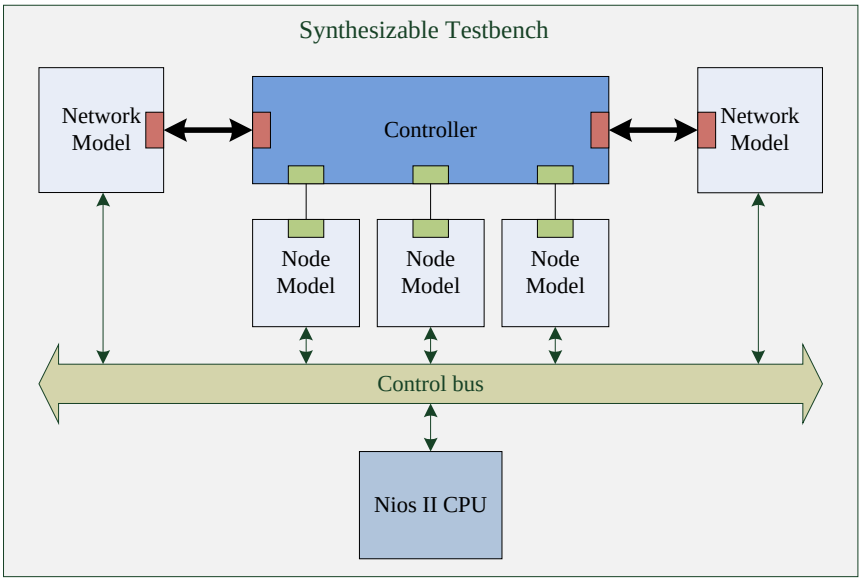


Figure 5.3: Verification of the controller with a synthesizable testbench.

The STB setup is illustrated in Figure 5.3. Models and the controller are running at the target speed (i.e. 10Gbps). The Nios II CPU is running at 50MHz as it merely controls the test and gathers results. The models are connected to the Nios II CPU via a control bus implemented as an Altera Avalon bus [Altera, 2008a]. The testing is monitored through a serial port console.

5.2.3 Testing in real environment

Because there is no off-the-shelf FPGA platforms with 10GbE available, the testing is limited to 1-gigabit speed. However, the 10GbE design can be fully tested, excluding the physical XGMII connectivity, by reducing the internal clock of the controller so that the operating speed corresponds the 1-gigabit traffic rate. That is, instead of running the controller at 156.25MHz, it is run at 15.625MHz. Unfortunately, the Altera's QDRII+ memory controller is not able to operate under 50MHz on Stratix III. Therefore, the tested design here utilizes only the on-chip memory even though the design supports the use of external memory for VOQ buffering.

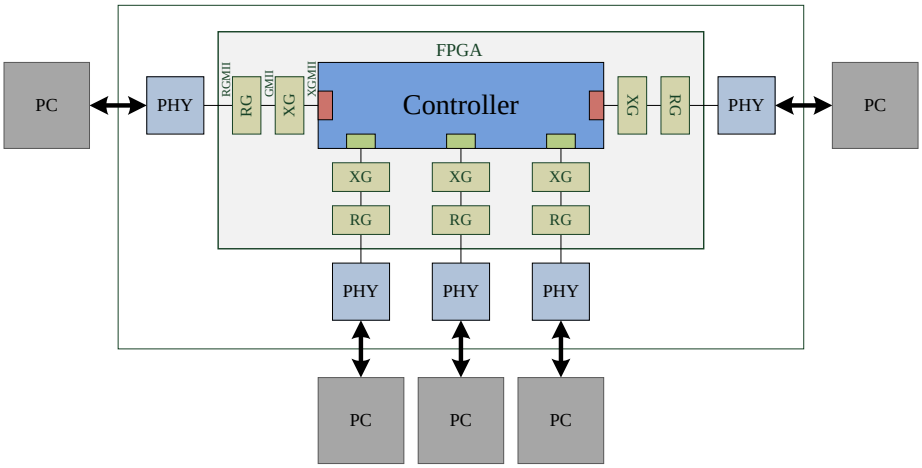


Figure 5.4: The testing environment with the controller and 1GbE PHYs.

The testing environment is shown in Figure 5.4. The XG blocks translate data between GMII (8b@125MHz) and XGMII (64b@15.625MHz). The synchronization is implemented with two CDC FIFOS. The depth of the RX buffer depends on the RX clock jitter. The RG blocks convert data between GMII and RGMII, and handle conversion from SDR to DDR and back.

5.3 Synthesis results

The design was synthesized using Altera Quartus 8.1 with variable amount of 10GbE and 1GbE ports and targeted to different Altera FPGAs. It was noticed that the critical paths were mostly in the combinational logic of the CRC computation in the FCS units. The switching fabric was challenging for older generation devices, although the complexity of the switching fabric does not increase as steeply as with fully interconnected switches, which have a N^2 relation. The synthesis results are analyzed further in the following sections.

5.3.1 Logic utilization

It was noticed during the synthesis that the logic utilization grows almost linearly as a function of the port count. Therefore, the logic utilization is given here in ALMs per port measured on Altera Stratix III with 10GbE ports. Due to the ALM sharing and other optimizations, the given values are averages and may vary considerably between synthesis rounds.

The rule memory and the command port are the only entities that are independent of the port count. They consume 195 ALM and 100 ALM, respectively, which are negligible when compared to the rest of the design.

The switch and the port entities consume 8% and 92%, respectively, of the total 2420 ALM/port. Table 5.1 shows the logic consumption in the sub-entities. As seen, the XGMII interface consumes almost 75% of a port's logic resources. Closer inspection reveals that the CRC calculation inside the XGMII is almost solely responsible for the large use of resources. Consequently on the device scale, three quarters of the device's resources are used for calculating CRC. This makes the CRC calculation a perfect candidate for future optimization since the other parts are marginal in terms of logic consumption.

Table 5.1: Logic utilization per design entity.

Entity	Subentity	ALM	%
Switch	Scheduler	14	1
	VOQ	179	7
Port	Frame buffer	110	5
	Packet engine	252	10
	Switch i/f	85	4
	XGMII	1778	74

5.3.2 FPGA performance

Because the design may be operated at a slower speed, the design's performance was analyzed on 1GbE in addition to the design's target speed. In 1GbE operation, the controller runs at 15.625MHz, which is one tenth of the operation speed of the 10GbE design. This is a relatively low speed for today's FPGA technology, hence the available on-chip memory and the external connectivity will become the limiting factor. The serial SGMII interface would be the best alternative, especially for a controller with higher port count. An SGMII interface requires one transceiver running at 1.25GHz. A parallel interface, such as GMII or RGMII, offers a valid alternative, although with increased PCB complexity.

10GbE operation requires F_{max} of 156.25MHz from the FPGA. In addition, the FPGA should have enough transceivers to implement XAUI interface as the parallel XGMII interface is very demanding to route on a PCB from the signal integrity point of view. A XAUI interface needs four transceivers, each running at 3.125GHz. In Altera's device family, this limits the selection to the GX versions.

The controller design was synthesized for all major Altera FPGA families with variable port counts (3 to 10) and target speeds (1GbE and 10GbE). Table 5.2 lists the final results which are given as a function of maximum achievable port count.

Table 5.2: Synthesis results for different Altera FPGA families. An asterisk means that the test was not done as it was assumed to be a success. The unit prices are rough estimates for comparison.

					Max. port count	
FPGA	Process	Transceivers	Speed-grade	Price	1GbE	10GbE
Arria GX	90nm	max. 12	6	$500	6	-
Stratix II GX	90nm	max. 20	5	$1000	*	3
			4	$1500	*	4
			3	$2000	*	5
Stratix IV GX	40nm	max. 48	2..4	unknown	*	10
Cyclone III	65nm	-	6..8	$200	10	N/A
Stratix III	65nm	-	2..4	$2000	10	N/A
Stratix IV	40nm	-	2..4	unknown	10	N/A

As expected, the design with 1GbE ports was easy for the selected FPGA devices. Only the Arria GX failed to synthesize ten ports. Other devices, including the low-cost Cyclone III, were able to hold ten 1GbE ports and still meet the timing. It is bit unfortunate that Arria GX, the low-cost device with desirable integrated transceivers, suffers from low memory granularity, which in turn limits the maximum port count to six.

With 10GbE ports, Arria GX device was not able to meet the timing requirements, even after several optimization steps. This was due to the low speed-grade. On Stratix II GX devices, the speed-grade had a big impact on the achieved port count. The slowest speed-grade was able to support three 10GbE ports while the fastest device was limited to five ports only due to the finite number of transceivers. The newest device, Stratix IV GX had no problems to meet the timing with ten ports.

5.4 Measurements

For the measurements, the controller was synthesized with 5 ports, each having a XGMII-GMII adapter and a GMII-RGMII adapter. The processing nodes were left out from the measurements by setting the PHYs to loopback mode. That is, the traffic to a node is looped back inside a PHY. Consequently, the measurements reflect better the controllers performance. The two networks were modelled by two identical computers (Intel Xeon Quad Core X3210, 4GB RAM, Intel PRO/1000 PT, Ubuntu 8.04 64-bit).

The five measurement setups are listed below. The first three are reference measurements and are used for giving perspective to the measurements with the controller.

1. **Direct**: Two computers connected directly together.
2. **Switch**: Two computers connected through a 1GbE switch (Dell PowerConnect 2716).
3. **Switch x2**: Two computers connected through a 1GbE switch twice.
4. **Passthrough**: Two computers connected to the controller in passthrough mode.
5. **Indirect**: Two computers connected to the controller in load-balance mode.

5.4.1 Latency

Latency was measured as a round-trip time (RTT) by using a standard *ping* command with arguments "`-f -w 2 -n -q`" with different packet sizes. The linear trend in Table 5.5 shows that the latency is proportional to the packet length in all setups. The measurements from *Passthrough* and *Switch* setups are comparable, which is expected due to the similar store-and-forward operation. In the *Indirect* setup, RTT is increased because the traffic is forwarded to a node and then again through the controller thus adding another store-and-forward step. This makes the *Indirect* comparable to the *Switch x2* setup. Based on the measurements, the controller's latency is negligible when compared to the latency added by the network processing in the processing nodes.

5.4.2 Throughput

Throughput was measured with *netpipe-tcp* application. The application sends a message through a TCP connection, waits for a reply, and then measures RTT and calculates the through-

put. Figure 5.6 shows the results with different message sizes. The controller does not affect the maximum achievable throughput as the result is same in all setups. However, the increased latency lowers the throughput with smaller message sizes. This was expected as the measurement is done in the described send-receive fashion and is therefore affected by the increased latency.

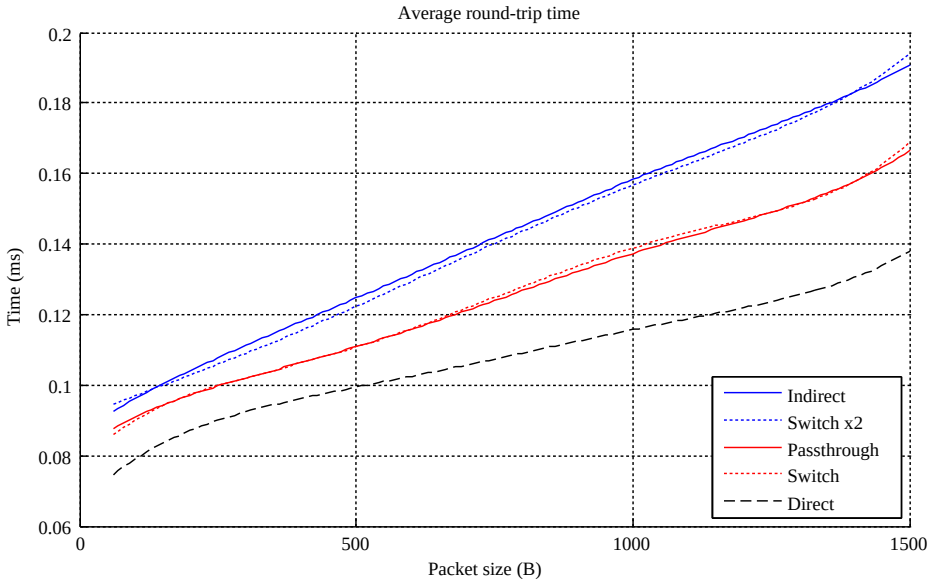


Figure 5.5: Measured round-trip time with variable packet size.

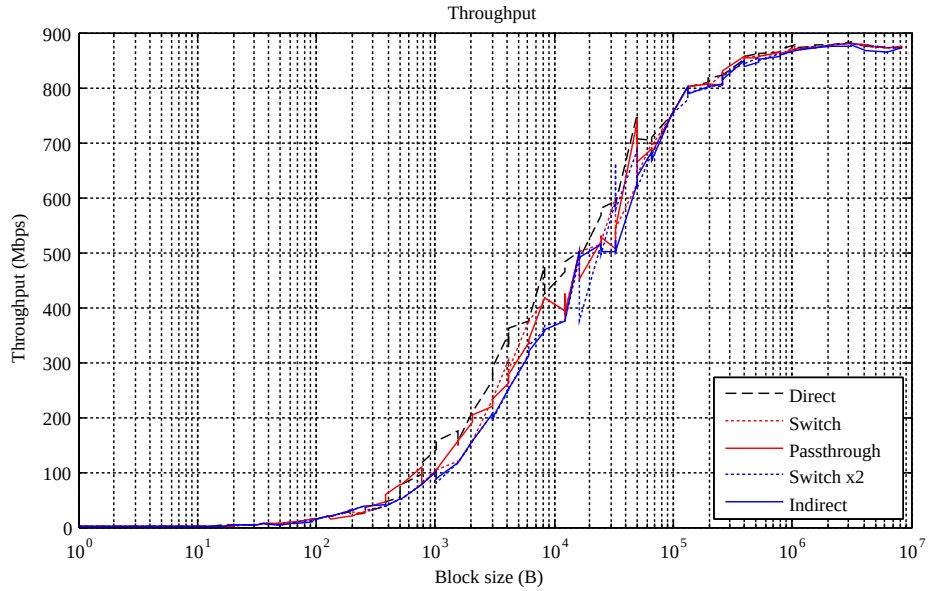


Figure 5.6: Measured throughput with variable packet size.

Chapter 6

Conclusions

The objective of this thesis was to develop a distribution controller that distributes the incoming network traffic to processing nodes and after the processing sends the resulting data from the nodes forward. The parallel processing scheme helps to process the network traffic using multiple units operating in parallel, and thus reduces the work load of a single unit. This processing style can be used, for instance, in network firewalls, network traffic inspection, and VPN gateways.

Two basic system topologies were identified. The processing nodes may be connected either directly to the controller or through an external network switch. The direct connection minimizes the encapsulation overhead but increases the required network port count of the controller. In the external switch case, at least three ports are required for the switch connection because of the increased traffic due to the encapsulation.

Two architectures were developed. The first one was a straightforward implementation that gave good practical experience to Ethernet and to the processing phases of the controller. The second revision was a modular controller architecture targeted for 10-gigabit Ethernet. The implementation is highly configurable. For instance, the port count and the roles of the ports are configurable through parameters. The design includes an internal switch that has an adaptive switch fabric, thus it is suitable for all identified system topologies.

The controller's design was verified on three levels. The design was simulated to verify the operation. A synthesizable testbench (STB) was used to accelerate the simulation performance to the real operation speed. The final test was in a real Ethernet network environment. Unfor-

tunately, the prototyping platform was not able to support 10-gigabit Ethernet. As a result, the tests were performed with 1-gigabit Ethernet. The design was slowed down by clocking it at one tenth of the target operating frequency. The latency and the throughput of the controller were measured with a simple computer-based setup. The latency is equivalent to the operation of a standard store-and-forward switch and the throughput is not affected due to the wire-speed operation.

The performance of the implementation was as high as expected from an FPGA-based implementation. The modular hardware architecture does not only increase the adaptability but also improves performance by implicitly enforcing pipeline and parallel sub-architecture in the design. It was a surprise how well the different FPGA devices performed in this application. At 1-gigabit speed all the FPGA families of Altera were able to meet the timing and area requirements. At 10-gigabit speed, the FPGA device's suitability depends more on the supported external connectivity. The latest high-speed device family with 40nm process technology (Altera Stratix IV GX) was able to handle all tested port counts. The previous family with 90nm technology (Altera Stratix II GX) had performance problems when the design contained more than three ports.

The CRC checksum calculation consumes almost 75% of the total logic space and forms the most timing critical part of the design. Therefore it is the best target for optimization and further research. One solution is to process the last word of a frame, which currently requires eight parallel CRC-units of different bit-widths, in multiple steps. This would add latency but should also result in better overall performance. Another point of optimization is the placement of the tagging unit. It could be moved to after the switching fabric to allow for larger encapsulation headers. Furthermore, control plane processing would be possible by adding a synthesizable CPU, such as Altera's Nios II, and connecting it to the internal command port. Due to the project time constraints, the measurements were performed only briefly and should be rerun using proper network measurement equipment.

Overall, the thesis project has met and exceeded the goals set initially. As the results show, the FPGA is very suitable for low-level network processing where the performance comes before other features. The reconfigurable hardware allows configurability that is not available when using other implementation alternatives, such as network processors or ASICs. For instance, the adaptive switching fabric is not possible with any other technique. The resulted architecture is versatile and adaptable to applications requiring similar characteristics.

Bibliography

Altera (1999, May). Metastability in Altera devices. Application Note 42.

Altera (2007a). altddio: Megafunction user guide.

Altera (2007b, December). Stratix III development board. Reference Manual.

Altera (2007c, November). Stratix III device handbook.

Altera (2007d, May). Stratix III programmable power. White Paper.

Altera (2008a, October). Altera Avalon interface specifications.

Altera (2008b, August). Selecting the right high-speed memory technology for your system. White Paper.

Altera (2008c, November). Stratix IV device handbook, I/O features.

Amdahl, G. (1967). The validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conf. Proc. 30*, 483–485.

Ashenden, P. J. (2002). *The Designer's Guide to VHDL* (second ed.). Morgan Kaufmann.

Baker, F. (1995, June). RFC 1812: Requirements for IP version 4 routers.

Bielby, R. (2002). Digital consumer convergence demands reprogrammability. *Xcell Journal*.

Braden, R. T. (1989, October). RFC 1122: Requirements for Internet hosts — communication layers.

Broadcom (2008, April). Broadcom drives transition to 10 gigabit Ethernet with industry's first quad speed 65nm 10GBASE-T transceiver to operate over 100 meters of UTP cable. press release.

Cerf, V. G., Y. K. Dalal, and C. A. Sunshine (1974, December). RFC 675: Specification of Internet Transmission Control Program.

Chao, H. (2002, Sep). Next generation routers. *Proceedings of the IEEE 90*(9), 1518–1558.

Chao, H. J. and B. Liu (2007). *High Performance Switches and Routers*. Wiley-IEEE Press.

Chen, J. and T. Stern (1991, Apr). Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch. *Selected Areas in Communications, IEEE Journal on 9*(3), 439–449.

Chu, Y.-C. (2001, July). Serial-GMII specification. Technical report, Cisco Systems.

Cisco (2008, June). Cisco visual networking index - forecast and methodology, 2007-2012. White Paper.

Comer, D. (1988). *Internetworking with TCP/IP*. Prentice Hall.

Comer, D. E. (2004). *Network Systems Design using Network Processors*. Prentice Hall.

Culler, D. E., J. P. Singh, and A. Gupta (1999). *Parallel Computer Architecture: A Hardware/Software Approach* (first ed.). Morgan Kaufmann.

Cypress (2007, November). QDR-II, QDR-II+, DDR-II, and DDR-II+ design guide. Cypress Application Note.

Cypress (2008, March). CY7C1263V18: 36-Mbit QDR-II+ SRAM 4-word burst architecture. Datasheet.

Dally, W. J. and J. W. Poulton (1998). *Digital Systems Engineering*. Cambridge University Press.

Deering, S. and R. Hinden (1995, December). RFC 1883: Internet Protocol, version 6 (IPv6) specification.

Deering, S. and R. Hinden (1998, December). RFC 2460: Internet Protocol, Version 6 (IPv6) specification.

Doyle, J. (1998). *Routing TCP/IP Volume I*. Cisco Press.

Gibb, G., J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown (2008, August). NetFPGA - an open platform for teaching how to build gigabit-rate network switches and routers. *Education, IEEE Transactions on 51*(3), 364–369.

Giladi, R. (2008). *Network Processors: Architecture, Programming, and Implementation*. Morgan Kaufmann.

Gupta, P. and N. McKeown (2001, Mar/Apr). Algorithms for packet classification. *Network, IEEE 15*(2), 24–32.

Hall, E. A. (2000). *Internet Core Protocols: The Definitive Guide*. O'Reilly.

Hill, M. D. and M. R. Marty (2008, July). Amdahl's law in the multicore era. *Computer 41*(7), 33–38.

Hluchyj, M. and M. Karol (1988, Dec). Queueing in high-performance packet switching. *Selected Areas in Communications, IEEE Journal on 6*(9), 1587–1597.

Hornig, C. (1984, April). RFC 894: Standard for the transmission of IP datagrams over Ethernet networks.

Hutton, M., J. Schleicher, D. M. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault, A. Lee, H. Kim, and R. Saini (2004). Improving FPGA performance and area using an adaptive logic module. In *FPL*, pp. 135–144.

IEEE802.1ad (2006, May). Virtual bridged local area networks. Amendment 4: Provider bridges. IEEE Standard 802.1ad-2005.

IEEE802.1AX (2008, November). Link aggregation. IEEE Standard 802.1AX-2008.

IEEE802.1D (2004, June). Media access control (MAC) bridges. IEEE Standard 802.1D-2004.

IEEE802.1Q (2005, May). Virtual bridged local area networks. IEEE Standard 802.1Q-2005.

IEEE802.3 (2005, December). Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. IEEE Standard 802.3-2005.

IEEE802.3an (2006, September). Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Amendment 1: Physical layer and management parameters for 10 Gb/s operation, type 10GBASE-T. IEEE Standard 802.3an-2006.

ITU-T94 (1994, July). Open systems interconnection - model and notation. ITU-T Recommendation X.200.

Iyer, S. and N. McKeown (2003, June). Using constraint sets to achieve delay bounds in cioq switches. *Communications Letters, IEEE 7*(6), 275–277.

JEDEC 21C (2003, May). Configurations for solid state memories. JEDEC Standard.

JEDEC 79-2E (2008, April). DDR2 SDRAM specification. JEDEC Standard.

JEDEC 79-3B (2008, April). DDR3 SDRAM specification. JEDEC Standard.

JEDEC 79F (2008, February). Double data rate (DDR) SDRAM specification. JEDEC Standard.

Karol, M., M. Hluchyj, and S. Morgan (1987, Dec). Input versus output queueing on a space-division packet switch. *Communications, IEEE Transactions on 35*(12), 1347–1356.

Kuon, I. and J. Rose (2007, Feb.). Measuring the gap between FPGAs and ASICs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26*(2), 203–215.

McKeown, N., V. Anantharam, and J. Walrand (1996, Mar). Achieving 100 *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE 1*, 296–302 vol.1.

McLaughlin, K., N. O'Connor, and S. Sezer (2006). Exploring CAM design for network processing using FPGA technology. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, pp. 84. IEEE Computer Society.

MorethanIP (2008, February). HSMC Ethernet quad-PHY daughter board. Reference Guide.

Northcutt, S., L. Zeltser, S. Winters, K. Kent, and R. W. Ritchey (2005). *Inside Network Perimeter Security* (second ed.). Sams.

Pagiamtzis, K. and A. Sheikholeslami (2006, March). Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits 41*(3).

Patterson, D. A. and J. L. Hennessy (1998). *Computer Organization & Design: The hardware / software interface* (second ed.). Morgan Kaufmann.

Paulin, P. G., C. Pilkington, and E. Bensoudane (2003). Network processing challenges and an experimental NPU platform. *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 64–69 suppl.

Peterson, W. and D. Brown (1961, Jan.). Cyclic codes for error detection. *Proceedings of the IRE 49*(1), 228–235.

Postel, J. (1980a, January). RFC 761: DoD standard transmission control protocol.

Postel, J. (1980b, August). RFC 768: User datagram protocol.

Postel, J. (1981a, September). RFC 791: Internet Protocol.

Postel, J. (1981b, September). RFC 793: Transmission control protocol.

Rabaey, J. M., A. Chandrakasan, and B. Nikolic (2003). *Digital Integrated Circuits* (second ed.). Prenctice Hall.

RGMII (2002, January). Reduced gigabit media independent interface (RGMII). Technical report, Hewlett-Packard Company.

Rose, J., R. Francis, D. Lewis, and P. Chow (1990, Oct). Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency. *Solid-State Circuits, IEEE Journal of 25*(5), 1217–1225.

Rose, J., A. E. Gamal, and A. Sangiovanni-Vincentelli (1993). Architecture of field-programmable gate arrays. *Proceedings of the IEEE 81*(7), 1013–1029.

Ruiz-Sanchez, M., E. Biersack, and W. Dabbous (2001, Mar/Apr). Survey and taxonomy of IP address lookup algorithms. *Network, IEEE 15*(2), 8–23.

Seifert, R. (1998). *Gigabit Ethernet*. Addison-Wesley.

Sezer, S., M. McLoone, and J. McCanny (2005, April). Reconfigurable architectures for network processing. *VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT). 2005 IEEE VLSI-TSA International Symposium on*, 75–83.

SFP+ (2008, May). Enhanced 8.5 and 10 gigabit small form factor pluggable module (SFP+), revision 3.0. SFF Specification.

Shah, N. (2001, September). Understanding network processors. Master's thesis, University of California, Berkeley.

Shang, L., A. S. Kaviani, and K. Bathala (2002). Dynamic power consumption in Virtex-II FPGA family. In *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, New York, NY, USA, pp. 157–164. ACM.

Sprecher, N., P. Klein, D. Berechya, and F. Lingyuan (2006, February). GMPLS control of Ethernet VLAN cross connect switches.

Spurgeon, C. E. (2000). *Ethernet: The Definitive Guide*. O'Reilly.

Srisuresh, P. and K. Egevang (2001, January). RFC 3022: Traditional IP network address translator (Traditional NAT).

Waldvogel, M. (2000, May). *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*. Ph. D. thesis, Swiss Federal Institute of Technology.

Widmer, A. X. and P. A. Franaszek (1983, September). A DC-balanced, partioned-block, 8b/10b transmission code. *IBM Journal of Research and Development 27*, 440–451.

XENPAK (2002, September). A cooperation agreement for 10 gigabit Ethernet transceiver package, revision 3.0.

XFP (2005, August). 10 gigabit small form factor pluggable module (XFP), revision 4.5. SFF Specification.

Zhang, Y., J. Roivainen, and A. Mammela (2006). Clock-gating in FPGAs: A novel and comparative evaluation. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, Washington, DC, USA, pp. 584–590. IEEE Computer Society.