POZNAŃ UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS AND TELECOMMUNICATIONS
CHAIR OF MULTIMEDIA TELECOMMUNICATION AND
MICROELECTRONICS

DOCTORAL DISSERTATION

Advanced video codecs implementation using Networks-on-Chip in FPGA devices

Author:
Marta STĘPNIEWSKA

Supervisor:
Prof. dr hab. inż. Marek DOMAŃSKI

Poznań, 2012

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC Application Specific Integrated Circuit
AVC Advanced Video Coding
AVS Audio Video Standard
BD Bidirectional
CABAC Context-based Adaptive Binary Arithmetic Coding
CAVLC Context-adaptive Variable Length Coding
CMP Chip MultiProcessor
DSP Digital Signal Processor
DOR Dimension Order Routing
FF Flip-Flop
FIFO First-In First-Out
FPGA Field Programmable Gate Array
GOP Group of Pictures
HDL Hardware Description Language
HEVC High Efficiency Video Coding
IP Internet Protocol
IP core Intellectual Property core
LUT Look-up table
MPEG Motion Pictures Experts Group
mPE modeled Processing Element
mNI modelled Network Interface
NI Network Interface
NoC Network-on-Chip
OCP-IP Open Core Protocol International Partnership
PE Processing Element
QoS Quality of Service
QP Quantization Parameter
RS-232 Recommended standard no. 232
SiP System-in-Package
SoC System-on-Chip
SDTV Standard Definition Television
TCP Transmission Control Protocol
UD Unidirectional
UVLC Universal Variable Length Coding
VC-1 Video Coding Standard-1

ABSTRACT

The dissertation deals with the design of an Network-on-Chip (NoC) for hardware implementations of advanced video codecs, especially those based on the AVC compression technique. The work is focused on the implementation problems of the AVC codecs on FPGA chips. The dissertation also covers issues related to the modeling of codecs, as a tool used in the design of network connection structures. An optimal network should meet the demands on throughput and consume little hardware resources.

In chapter II a review of connection strategies on a chip is presented.

Chapter III presents the AVC compression standard and its implications for hardware implementations.

In chapter IV an original proposal of an NoC-based AVC codec implementation is presented. The original proposal of the research team in which the author actively participated is then described. The original network structure is also considered. Further, the chapter discusses the benefits and limitations of codec implementation with the proposed Network-on-Chip.

Chapter V discusses the problem of codec simulation and presents an original proposal for the modeling of such applications. The chapter also describes the original simulator proposal developed by the author. The accuracy of the simulation has been verified by comparison with the hardware implementation results.

In chapter VI a very simplified analysis technique for codec-based implementations is proposed, which uses queue modeling. The chapter also presents an exemplary application of the proposal.

Chapter VII contains the analysis of the optimal structure of an NoC performed for basic NoC structures intended for an AVC decoder. The chapter also discusses the choice of the optimal structure.

The presented research results point to the competitiveness of NoC as an interconnection architecture for advanced video codecs. The proposed modeling of the application facilitates the design and the choice of the optimal structure. The modeling methods can be used in the

design of any application, and especially in wide a spectrum of hardware video processing implementations.

STRESZCZENIE

Praca dotyczy projektowania sieci w układzie (ang. Network-on-Chip (NoC)) dla sprzętowych realizacji zaawansowanych kodeków wizyjnych, w szczególności tych opartych na technice kompresji AVC. Autor skupił się na zagadnieniach realizacyjnych techniki AVC w układach FPGA. W pracy zostały także omówione zagadnienia związane z modelowaniem sprzętowych kodeków wizyjnych, jako narzędziem w projektowaniu struktury połączeń sieciowych. Sieć optymalna powinna spełniać wymagania dotyczące przepustowości oraz zużywać niewielkie ilości zasobów logiki programowalnej.

Rozdział II zawiera przegląd rozwiązań dotyczących połączeń w układzie.

W rozdziale III została przedstawiona technika kompresji AVC, i jej wymagania dla realizacji sprzętowych.

Rozdział IV zawiera propozycję sprzętowej realizacji kodeka AVC opartej na sieci w układzie (NoC). Przedstawiona realizacja jest wynikiem prac zespołu, którego członkiem był autor. Rozdział omawia oryginalną konstrukcję sieci. Zawarto w nim również omówienie zalet i wad implementacji kodeka z użyciem zaproponowanej sieci.

W rozdziale V zostało zawarte omówienie problemu symulacji kodeków oraz została przedstawiona autorska propozycja dotycząca modelowania takich aplikacji. Opisano również własną propozycję symulatora. Dokładność symulacji zweryfikowano przez porównanie z wynikami otrzymanymi dla realizacji sprzętowych.

Rozdział VI zawiera propozycję uproszczonej metody analizy realizacji kodeków, która wykorzystuje modelowanie kolejkowe. Rozdział ten wskazuje także przykładowe zastosowanie takiego modelowania.

W rozdziale VII przedstawiono analizę struktury optymalnej sieci w układzie, wykonaną dla podstawowych struktur NoC przeznaczonych dla dekoderów AVC. Szczegółowo omówiono w rozdziale wybór optymalnego rozwiązania.

Przeprowadzone badania wskazują na użyteczność zastosowania sieci w układzie dla zaawansowanych kodeków wizyjnych. Zaproponowane modelowanie aplikacji ułatwia projektowanie

i wybór optymalnej struktury połączeń. Przedstawione w pracy rozwiązania mogą znaleźć zastosowanie w każdym rodzaju aplikacji, a w szczególności w szerokim spektrum sprzętowych realizacji przetwarzających dane wizyjne.

CHAPTER I

Introduction

1.1 Integrated digital circuit target platforms

For digital systems, hardware applications can be designed on one of two integrated-circuit target platforms: Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA). ASIC is a customized integrated circuit destined for some specific applications. On the other hand, FPGAs functionality can be programmed after production of a chip. Each of the mentioned platforms has own advantages and specific application areas. Advantages of both platforms are discussed in e.g., [**?**]. FPGAs are characterized with a shorter time-to-market and simpler design cycle than ASICs [**? ? ?**].

Although a functional specification and Hardware Description Language (HDL) description might be the same for ASICs and as well FPGAs, the following stages of the design flow, in the case of ASICs, require much more complicated issues to be resolved, i.e. the ones concerning equivalency checking, timing analysis and verification of layout dependent effects. These are absent in FPGA design flow. Moreover, FPGAs are widely used in ASIC prototyping and in specialized devices such as video processing cards e.g., [**? ? ?**].

The problems and solutions presented hereafter, concern functional specification and HDL description stage, unless stated otherwise. Such a restriction allows for drawing general conclusions that are applicable for both platforms. Some solutions were designed for FPGAs (presented in chapter IV), however, these are suitable as well for ASIC designs, since the same issues are present in both platforms (i.e., connectivity problem between modules).

1.2 Integrated circuit design

According to Moore's law, the number of transistors on a chip doubles every 18 to 24 months[**?**]. Such an intense development of the integrated circuit industry results in new challenges of ASIC and FPGA design, due to the increased complexity [**?**]. One of such challenges is a productivity gap which is a difference between the annual growth rate of the number of transistors on a chip, and productivity growth of the designer in terms of number of transistors per month [**?**]. In order to tackle these problems, System-on-Chip (SoC) and System-in-Package (SiP) technologies were introduced [**?**]. SoC technology uses predefined and preverified Intellectual Property cores (IP cores), that are implemented in a single chip [**?**]. In cases when it is not feasible to design a SoC for an application, SiP technology is used, that comprises a few chips in a single package. Complex SiP products may contain a few SoC components [**?**].

Reusability of system components provides significant cost reduction due to reduced design and verification time [**?**]. System which consists of IP cores is assembled relatively quickly, as compared to a design made from scratch. Nevertheless, such approach yields connectivity problems, since the complexity of an interconnection structure is increased with the growing number of blocks [**?**]. The current SoC design became connection-oriented, since the design of efficient interconnect structure is not trivial task [**?**].

In order to perform design verification, a hardware and/or software simulation is performed. Hardware simulation software accepts the HDL description of modules and estimates signal values of hardware implementation. Such a simulation is very accurate, however it is time consuming especially in the case of large applications. In order to estimate the general application and interconnect performance, a software simulation is performed. Software simulation accepts a model of processing blocks with a model of interconnect system. Such approach is less accurate, however the results are obtained in a shorter time.

Typical design flow of hardware application is shown in figure 1.1. At first IP cores are collected and interconnect architecture is roughly defined e.g., topology. Then software and/or hardware simulation is performed. After results are collected, their assessment is performed. If the designer decides to introduce some changes in the communication infrastructure, the whole process is repeated, until satisfactory interconnection is obtained.

This process is long and laborious because of the time spent on simulation. Moreover, the simulation might not give straightforeward results e.g., that describe which data path influences

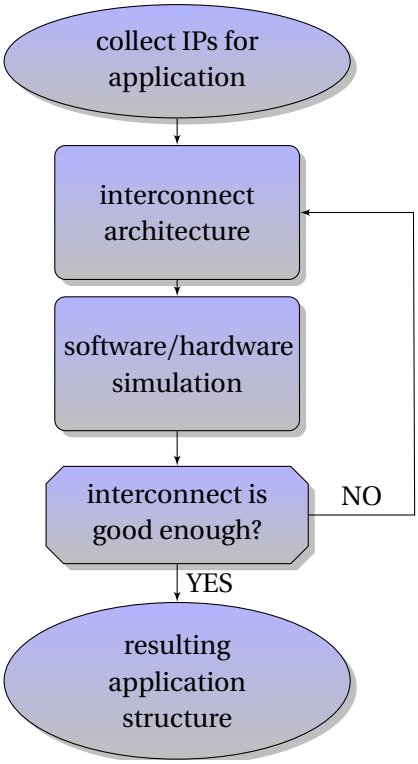


Figure 1.1: Typical flow diagram for hardware application design

the overall latency the most; and further analysis of the simulation results needs to be performed.

1.3 Interconnection techniques for on-chip applications

As SoCs became more and more complex and containing many Processing Elements (PEs), the interconnection architecture has to meet new requirements: scalability, reusability and high performance. Scalability is an ability of the interconnect system to expand or shrink easily with the minimal loss of performance as the application changes (namely, module functionality, number of modules and their placement). Easiness means that with the application change mentioned, interconnect system does not impose a redesign of communication structure or PE functionality. Such an approach requires a standard interface for modules and a well-defined functionality at PE, which in turn enables reusability of once designed and verified IP core. At the interconnect site, there is a requirement of defining a communication protocol that offers the data exchange speed that is required.

There are three basic interconnect approaches: direct connection, shared bus and Network-on-Chip (NoC), which are described in detail in chapter II. Direct connections (or peer-to-peer

connections) offer communication between two PEs over a direct link. Such a link may be designed to transmit one or a few data types at high speed (there is no bandwidth competition). Nevertheless, this technique is poorly scalable. A change in the transmission protocol is followed by changes in the design of both communicating PEs. Also, in order to develop data exchange with another module, a new connection needs to be established. As the number of links grows, a problem arises of routing them in the target platform (especially in FPGA). In the case of FPGAs, the number of links between logic cells is limited, and when all are exploited, synthesis tool uses other logic cells to provide connectivity. This results in increased hardware consumption [**? ? ? ?**].

A shared bus offers a single interconnect medium, to which all communicating modules are attached. Such an approach solves the problem of a large number of links between modules, however, it introduces bandwidth competition. Shared buses require a standard interface and offer better scalability than direct connections.

Medium competition is a major disadvantage of the shared bus approach [**?**]. In order to tackle this problem, a NoC was proposed [**?**]. This technique, similarly to the Transmission Control Protocol (TCP)/Internet Protocol (IP) networks, uses packets for communication, and routers to find a path in the network. NoC becomes a mature solution for on-chip communication, but it still needs improvement in design methods [**?**]. A detailed description of NoC and its comparison to other connection techiques is given in section 2.3.

1.4 Hardware for video processing

A compressed video is used in a growing number of devices, not only television sets and computers, but also mobile devices such as smartphones and tablets. This impose the requirements on wireless links performance, and on the efficiency of other video compression standards. This dissertation concerns the state-of-the-art video compression technoogies and the respective standards, such as Advanced Video Coding (AVC)[**?**], which is widely used e.g., in television systems. There are also other recently proposed standards such as VC-1[**?**] and Audio Video Standard (AVS)[**?**]). In parallel to this work a new compression technique is being developed which is High Efficiency Video Coding (HEVC)[**? ?**], however, it is not an accepted compression standard yet.

Stereoscopic and multiview video is another trend imposing high requirements on storage

devices or on channel capabilities (i.e., size of available bandwidth to send a video content). Such video require sending a much greater amount of data, than in the case of single view systems [**?**]. In order to cope with such requirements a video codec needs to be paralellized, namely some parts of the codec need to be multiplied to extend computational capabilities. Another approach, presented in [**?**], assumes a multiplication of the whole codec to gain computational power.

The hardware implementations of video processing techniques mentioned contain many modules that realize algorithms with a high demand on bandwidth at the input and the output of the modules they consist of. Those techniques are also characterized with variable data paths (i.e., different sets of modules that process data in a set of modes). A growing number of modules and their bandwidth demands entail connectivity problems due to the required number of links and bandwidth. A large number of links causes difficulties in placing and routing during synthesis on the final chip. In the case of a large number of links between modules in FPGAs devices, connections can be routed through Look-up tables (LUTs), causing hardware consumption.

The answer to the problems mentioned are Network-on-Chip (NoC), as presented in section 1.3. However, as described in section 3.2 video applications are considered as relatively small to be implemented with the use of NoC. Network-on-Chip implementation involves hardware costs, which in the case of video applications are significant. Nevertheless, the growing complexity of video processing techniques requires a new connection architecture that would offer acceptable bandwidth rate and scalability at low hardware cost [**? ?**].

Another problem is the modeling of video applications (see section 5.2), since they have a complicated traffic pattern, characterized with many data types, which reflects a number of encoding/decoding modes. A model of hardware application is exploited to perform a software simulation. The encoding and decoding process is described in detail in section 3.1. There are two encoding/decoding data paths, with different data patterns in each. This means that using simplified description of modules, as presented in [**?**] is not feasible, and is followed by a need to construct a mechanism in a simulator to provide such capability. Although software simulation is less time consuming than the hardware one, it still requires time to perform experiments, which may be repeated a few times in order to obtain statistically viable results [**?**]. In some cases, running a simulation is needed to observe what effect on the overall performance is introduced by some changes in the interconnect structure, and how great this influence is. In such a situation, a tool for estimating simulation results, with the use of simple calculations, should

be provided. Moreover, such a tool could also be used in rough performance analysis of PEs, and modules placement in the interconnect structure, as it would give estimates on the resulting application capabilities.

1.5 Goals and thesis of the work

The work focuses on the adoption of NoC as a connection architecture for video codecs which, as mentioned before, offers good scalability and efficient exploitation of bandwidth. Although such an architecture yields a cost in terms of hardware consumption. NoC proposal needs to balance between the offered functionality and hardware consumption.

The goal of the work is to discuss the problem of defining the connection architecture suitable for video codecs, especially for AVC compression technology. The design of connection infrastructure requires a preliminary analysis of demands for the modules. Such an analysis can be performed with the use of a simulation, that accepts a model of an application and traffic specification for modules. As it has been mentioned before, video applications and especially video codecs are characterized with a complicated traffic model that reflects different coding modes. This requires proposing a description of a traffic model that includes these issues.

The dissertation also aims to approximate simulation results with use of simple calculations, by exploiting e.g. a queue model. Such computations would significantly reduce time needed for a rough comparison of simulation results. Approximation of simulation result would allow for using it to arrange modules in a communication infrastructure and to estimate the influence of different changes in the proposed architecture on the overall application performance without performing a simulation. Such a model should also allow for the estimation of a delay introduced by each module on the data path.

The thesis of the dissertation is formulated as follows:

For hardware implementations of advanced video codecs, Networks-on-Chip are competitive relative to other connection structures realized on a chip. The architecture of such Network-on-Chip designs may be easily assessed with use of simple simulators or even analytical calculations.

The author will show the benefits of the implementation of a codec with the use of NoC and discuss the costs of such an approach. The basis for the analysis is the physical implementation of a codec. The author actively participated in the design, implementation and debug of this

proposal.

The author will also show the methods for the modeling of hardware applications and will propose a new description method covering specifics of traffic in the hardware video decoder. The method will be implemented in a simulator and verified in terms of accuracy of the simulation results. The accuracy will be assessed by a comparison of the simulation outcomes to the real performance results obtained from hardware implementation.

The author will show that simulator results can be roughly approximated with the use of simple calculations, described as a queue model. Moreover, the applicability of such modeling for a video decoder will be presented with the analysis of results accuracy. Analysis of the accuracy of the simulation will be carried out in accordance with [**?**]. The authors present research practices to obtain reproducible results in the field of signal processing.

The methods of modeling of a video codec mentioned should also be helpful during the design of communication infrastructure for other applications, especially in the area of video processing, which characterize with a traffic model similar to the codec. This model can be applied for the design of any software simulation that would estimate hardware performance results. Also, the application of a queue model to approximate simulation results can be adopted to any hardware application of similar traffic characteristics. Although the dissertation discusses the usage of a specific queue model, in the case of a different traffic scenario, another model can be applied. The author will show the benefits of such modeling.

1.6 Dissertation overview

The organization of thesis is as follows: chapter II presents a review of references on interconnection techniques that are currently used in SoC design with their performance and applicability analysis.

Chapter III presents an overview of AVC compression standard and its implementation in hardware. The compression technique description is focused on encoding and decoding process as it has a major influence on the further design. Also in this chapter a few proposals are presented regarding the implementation of AVC encoders and decoders in the hardware focused on communication issues between modules.

In chapter IV an implementation of AVC is presented in which the author has contributed a few modules. Also the author participated in the debugging of the major PEs of the application

presented.

Chapter V presents the author's proposal on NoC simulator adjusted to the hardware requirements of video codecs, however, able to simulate any type of hardware interconnect of any application. Simulator results were confronted with hardware simulation results and the accuracy was estimated. Moreover, based on these results hardware operating frequency was estimated.

Queue modeling is proposed in chapter VI and compared with the simulation results. Also the benefits and limitations of the proposal are discussed.

Chapter VII presents a few NoC proposals for AVC decoder and discusses their applicability with respect to hardware consumption and system performance.

Finally, in chapter VIII conclusions of the dissertation are presented and a list of the original results is given.

Acknowledgment

The dissertation was supported by the public founds as a research project nr "N N517 386436".

CHAPTER II

Techniques for interconnections on chip

System-on-Chip (SoC) is a system of Processing Elements (PEs) connected with each other to perform a set of tasks [**?**]. From the functional perspective this is an application, that performs a specific activity, which is implemented in hardware. Authors in [**?**], define a PE as a part of an application that is connected to the interconnect medium independently from other parts. However, from the functional perspective these parts perform a particular set of operations enclosed in a single module with intellectual rights to its design and are called Intellectual Property core (IP core)[**?**]. Nevertheless, in [**?**], the authors use the term IP core to describe the interconnect organization in SoC. Hereafter, the term PE refers to a unit of logic performing an undefined function which is connected to the interconnect medium (similarly to [**?**]). On the other hand, IP core refers to a unit of logic defined in e.g., Hardware Description Language (HDL), which performs a certain function.

There are 3 basic interconnection techniques for on chip applications: direct connections, shared bus and Networks-on-Chip (NoCs). Each method is characterized with a different number of links, connection speed and scalability. Direct connections (point-to-point) are established with a single link for each communication. A shared bus uses a single link that is connected to all PEs communicating with each other. Sending PE reserves a link and transmits data, during which no other module is allowed to send. It means bandwidth competition. On the other hand NoC introduces packet transmission in a network that consists of routers.

Direct connections offer a high throughput, but require a separate link technique, and consequently a high number of connections (compared to other interconnection techniques), as well as low scalability of a SoC. Low scalability at the architectural level means that it is difficult to change (add or remove) any functionality in the application without redesigning it. If a change in

the application functionality occurs, the less redesigning it requires, the more scalable it is. On the other hand, from the performance perspective scalability refers to interconnect efficiency changes, in the case of modifications in an application (adding or removing a module). If the interconnect performance changes significantly, while new PEs are added, such an architecture is not scalable [**?**].

In order to manage these problems, shared buses were introduced, in which PEs are connected to a single bus to exchange information between each other. Using a shared bus as an interconnect significantly reduces the number of links in an application, and forces the designer to use a standard interface for communication, which in turn improves the scalability of the application. However, the access to interconnect medium is competition-based, which reduces the available bandwidth. This problem grows with the increasing number of PEs[**?**], and for large applications shared bus communication becomes a bottleneck[**?**].

In order to reduce bandwidth competition, Network-on-Chip (NoC) was introduced as an interconnect technique for PEs. As in Transmission Control Protocol (TCP)/Internet Protocol (IP) networks, NoC uses a packet-based data exchange between PEs, with routers to determine the data path in NoC. PEs use Network Interfaces (NIs) to access the interconnect medium, and to make NoC transparent for the PEs. This means that NI translates data format between the one that is used by PE and the one that is used in NoC. Using NoC requires employing a standard interface for PEs to connect to the network. Since the effort put in the design of a NoC may be higher than in the case of other interconnect techniques, NoC design is oriented on the future reuse of both: NoC architecture and PEs.

2.1 Direct connection

Direct connections do not require any additional device to pass data between PEs. It means that having e.g., two designed modules exchanging data, there is no intermediate module to pass data along the path, and there is no competition in the access to the interconnect medium, as shown in figure 2.1.

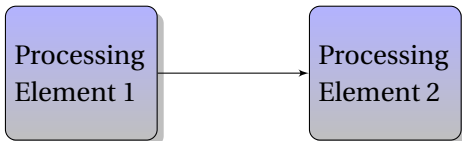


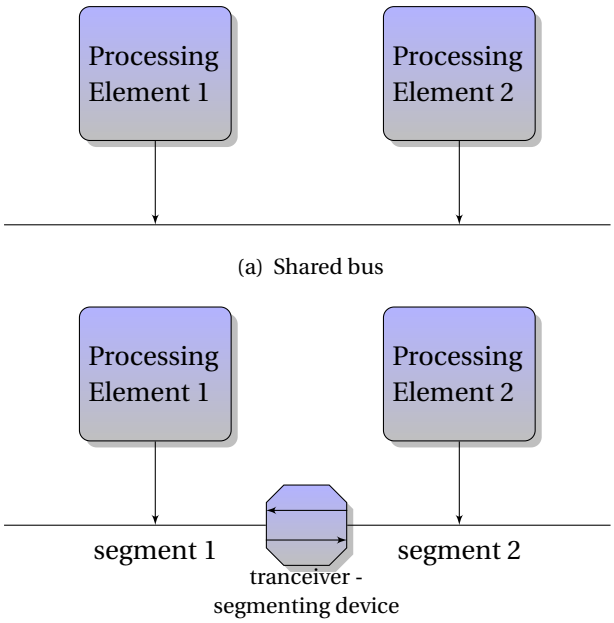
Figure 2.1: Exemplary direct connection between 2 modules

Interfaces designed for such a connection are specific to the data type that are exchanged over that link. It means that if the data format is changed, also the interface needs to be changed in both modules. This is one of the disadvantages of this technique, as it limits the reusability of PEs, because they need to be redesigned for use in another application. Another problem is low scalability, which means that in the case of a change made in one module, all other modules may require redesigning. Connecting many modules directly to each other results in a large number of links and difficulties in routing them in the final hardware design. It is especially an issue in the case of implementation in Field Programmable Gate Arrays (FPGAs), due to their limited connectivity [**?** **?**]. Direct connections offer theoretically the highest throughput of all connection techniques, which in practical implementation can be decreased by routing long links. Long links increase the delay on the critical path, which in turn decreases the operating frequency of the whole application[**?**]. In order to reduce critical path delay, an intermediate device can be introduced on such a path, which increases the operating frequency, but will extend transmission time by a few additional clock cycles[**?**].

2.2 Shared bus

PEs connected to a shared bus exchange data over a common medium, as shown in figure 2.2(a). Using a shared bus imposes the implementation of a common interface which increases the reusability of the once designed interconnect medium e.g., the shared bus proposed in [**?**] or standardized bus interfaces compared in [**?**].

Such an approach assumes competition in accessing the medium which causes a reduction of bandwidth. Moreover, in [**?**], the authors highlight the problem of inefficiency in energy dissipation of a shared bus, because switch capacitance increases with the number of nodes. Shared buses also introduce waste of energy at the system level, due to the functional congestion. The authors also draw the conclusion that because of energy consumption, the resulting usage of a shared bus as an interconnect medium in large-scale systems is "severely" limited. Using a segmented shared bus (see figure 2.2(b)) may help tackle some of these problems, however, not removing them completely. In a segmented bus a transceiver is introduced, which forwards only the messages that are destined to the other segments of the bus. Division of a shared bus into segments reduces bandwidth competition, which in turn increases bandwidth. In order to assess shared bus segmentation efficiency, the authors in [**?**] considered these two scenarios



(a) Shared bus

(b) Segmented shared bus

Figure 2.2: Shared bus architecture

of a shared bus (segmented and non-segmented) with uniform traffic. The results are gathered in table 2.1 (see first two rows) and show a significant reduction in operating frequency for the non-segmented shared bus and the same power dissipation.

2.3 Network-on-Chip (NoC)

The basic concept behind on-chip networks is to "switch packets not wires", which was proposed in [**?**]. In Networks-on-Chip (NoCs) data exchanged between PEs are divided into sets of packets, which are sent over a set of switching devices (routers). Each PE is connected to the network using NI that translates between data format of the network and the PE. Routers transfer packets over the network from the source PE to the destination, using a routing protocol for path determination, which is specific for a given network.

Such an organization of data exchange requires using standard interfaces, which, in fact, allows for PE reuse in another application. Moreover, the scope of the design process points to the connection issues, since functionality is defined at the PE level, and is independent of the interconnect level[**?**].

2.3.1 Efficiency comparison between NoC and other connection strategies

NoC is used not only due to its flexibility, but also because of its performance. NoCs offers less bandwidth competition, distributed senders and receivers, and the ease to route in the final implementation[**?**]. A shared bus connection needs to operate at a higher frequency to offer the same bandwidth as the number of PEs increases[**?**]. The more nodes exist in the system, the greater is the difference. The authors in [**?**] also considered latency in 2 scenarios: short and long messages. In the case of short messages NoC outperforms the shared bus, when comparing latency at maximum workload. For a smaller workload, the shared bus offers lower latency for less than 25 nodes in the system. However, when sending longer messages, NoC offers a comparable or lower latency than the shared bus, especially in the case of systems with a high (25+) number of nodes.

In [**?**], NoC outperform both segmented and non-segmented shared buses in terms of wire-cost, power dissipation and operating frequency. The results are shown in table 2.1, which gathers asymptotic cost functions depending on the number of nodes (n) in SoC under uniform traffic, for different connection architectures. At first, the results for NoC were calculated, and then for other architectures, with an assumption of the same effective bandwidth [**?**].

Table 2.1: Asymptotic cost functions according to [**?**]. n - number or PEs

Architecture	Total area	Power dissipation	Operating frequency
non-segmented shared bus	$O\left(n^3\sqrt{n}\right)$	$O\left(n\sqrt{n}\right)$	$O\left(\frac{1}{n^2}\right)$
segmented shared bus	$O\left(n^2\sqrt{n}\right)$	$O\left(n\sqrt{n}\right)$	$O\left(\frac{1}{n}\right)$
NoC	$O(n)$	$O(n)$	$O(1)$
direct connections	$O\left(n^2\sqrt{n}\right)$	$O\left(n\sqrt{n}\right)$	$O\left(\frac{1}{n}\right)$

Non-segmented shared bus occupies the greatest area, because its width grows to offer the same bandwidth as NoC. Bus width in the case of a segmented bus does not grow as much, hence the costs grow $O\left(\frac{1}{n}\right)$ slower than in the case of a non-segmented bus. A similar effect on the total area is produced by direct connections architecture, due to growing links length with increased number of nodes. A shared bus (segmented and non-segmented) and direct connections dissipate power at the same rate with the growing number of nodes, which is $O\left(\sqrt{n}\right)$ higher than in the case of NoC. The benefits of NoC are clear, when comparing operating frequency loss with the growing number of nodes. Only a network is able to provide the same operating frequency

irrespectively of its size. In the case of a non-segmented bus the loss is $O(n^2)$ and for others is $O(n)$. The authors in [**?**] also point out that the advantages of NoC grow in the case of more realistic scenarios (non-uniform traffic), where traffic is more localized. The results presented show that NoC implementation becomes beneficial (in terms of the total area, power dissipation and performance) with the growth of PE number.

Evaluation of shared bus configuration in a multiprocessor SoC is presented in [**?**]. The authors simulated a system in which 4 microprocessors are connected to 4 Ethernet ports using a shared bus. In this system, there are also 4 uniform-traffic generators connected directly to the Ethernet ports. The results show high performance loss due to medium access competition. The shared bus is not able to sustain its theoretical capabilities, and therefore a new interconnect approach is needed. The authors also point out, that the shared bus performance-loss problem grows with the number of PEs in SoC. The results in [**?**], also lead to the conclusion, that even in the case of small designs (up to a few nodes), but with heavy traffic, shared bus is not a solution as an interconnect medium.

2.3.2 Topology

In the general classification of NoC, topology can be direct or indirect [**?**]. In the first case switching capabilities are implemented within a PE, and consequently such a device can be directly connected to the network. On the other hand, indirect topology assumes that each module is connected to the network through a link to a switching device. Also, if the topology is constructed according to some predefined patterns, it is a regular topology. In other cases the topology is irregular.

In [**?**], there are also other high-level topology properties:

- symmetry - network is symmetrical if looks the same from a perspective of each router,
- router degree - is defined as total number of input/output ports of a router,
- homogeneity - network is homogenous if all routers have the same degree,
- bisection bandwidth - it is a bandwidth between all pairs of nodes in case of dividing topology into two equal halves,
- hop count - is defined as a maximum number of routers between two nodes on the packet path, assuming that the chosen path is minimum,
- diameter - describes the maximum distance between two PEs in cycles, and it is completely dependent on physical implementation,

- connectivity - it is a minimum number of links that need to be disconnected to isolate an PE (prevent from sending or receiving packets),
- total number of routers - which is the minimum number of routers to connect all PEs in the system,
- total number of links - is defined as the number of unidirectional network links to connect all the nodes in the network.

The authors in [**?**] do not mention the planarity of topology graph. In a planar graph, edges intersect only at the nodes. From the physical layout perspective it is an important factor, because each link crossing (that does not occur in a router) requires an implicit insertion of a switching device. In the case of Application Specific Integrated Circuits (ASICs), it also can be routed through physical layers of silicone, which in turn may increase the diameter and/or routing costs. In the case of FPGAs, intersections of non-planar topology graph mean hardware consumption (for the switching device) and operating frequency loss due to the long critical path delay.

Each of the presented properties has a different meaning to the topology and optimization of one of them can hurt the others significantly. For example, in theory, the total number of routers should be as small as possible in order to reduce hardware consumption of network devices. However, reducing the number of routers causes an increased length of links in hardware design, which has a negative effect on the diameter of the network. This example shows that optimization needs to be carefully performed.

2.3.2.1 Basic topologies overview

There are many topology proposals for NoCs. Presented below are only the basic regular topologies that are used in the NoC design and each may have a few minor modifications depending on the implementation.

Mesh topology and its modifications

According to [**?**], the mesh topology (see figure 2.3) is the most widely spread topology used for NoCs. This topology was proposed in [**?**] and is well suited for a system with many same-sized nodes (e.g. Chip MultiProcessor (CMP)). In the case of irregular sizes of PEs, some modifications may be needed as in [**?**], in which some of the routers' links located near large modules are missing. In such a case, the diameter is the same as in the original mesh.

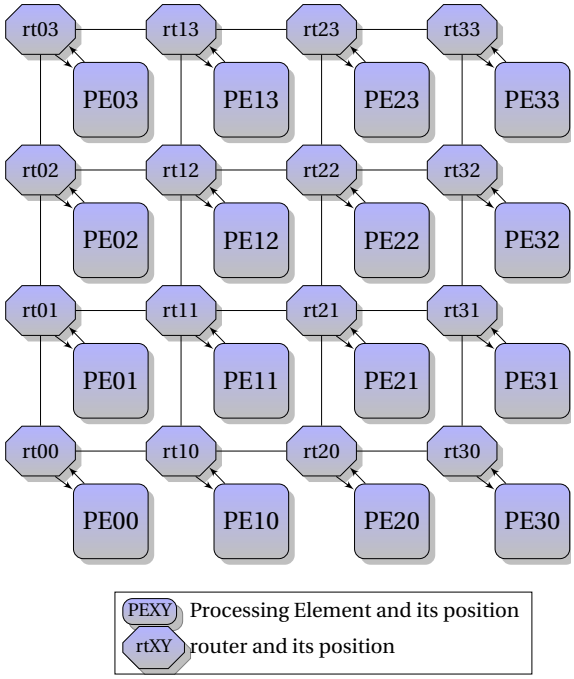


Figure 2.3: Exemplary 2D mesh topology. Each link is a two-way (⇆) connection.

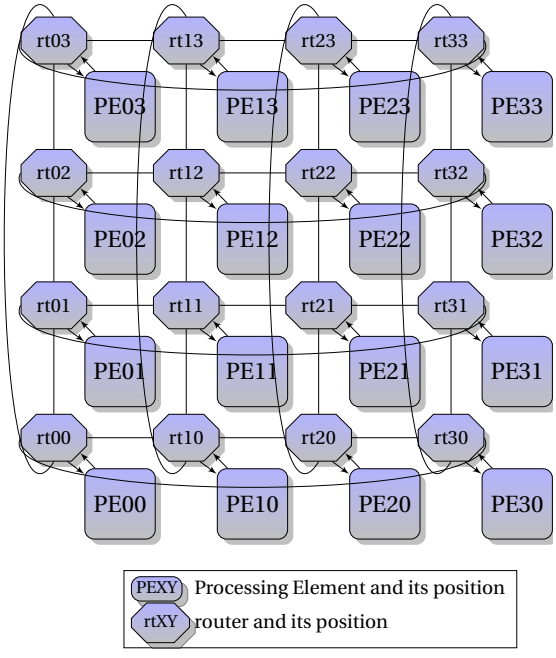


Figure 2.4: Exemplary 2D torus topology. Each undirectional link is a two-way (⇆) connection.

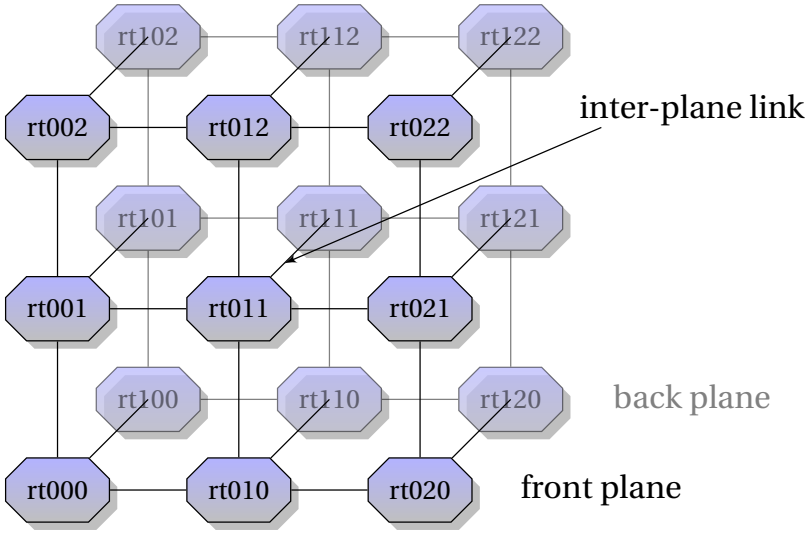


Figure 2.5: Exemplary 3D mesh topology. Each unidirectional link is a two-way (⇆) connection.

One of the major modifications of a mesh is torus (figure 2.4) which was proposed in e.g., [**?**] also mentioned in [**?**] and [**?**]. It is a mesh network with added links to connect routers at the opposite network borders. The obtained network is symmetrical with a reduced hop count, which results in decreased data delivering latency. However, such topology is not planar and may produce problems with physical layout routing. One of the other modifications of mesh topology is placing nodes in 3 logical dimensions (e.g., [**?** **?** **?** **?**]), which is shown in figure 2.5. Such a topology has a smaller hop count because each node has at least 3 neighbors (in the original mesh the minimum number of neighbors is 2), but causes routing problems such as the increased diameter in the case of converting 3D mesh into 2D plane.

Star and tree

Star/tree topology introduces an important factor of topology which is hierarchical placement of nodes in the network. Hierarchy helps to control traffic between parts of the network, so the paths of data transfers can be better managed[**?**]. This is a major advantage over mesh topology, which is poorly scalable. The exploitation of the tree topology as the basis of hierarchical network was discussed in [**?**]. Star topology was used e.g., in [**?**] for systems of up to 63 nodes.

Figures 2.7 and 2.6 show exemplary star and binary tree topology. Depending on the implementation, different restrictions can be applied to build a network. For example, in [**?**], each router has four ports, one for communication with the root and others connected with the leaves. The example provided shows that the PEs are connected to the lowest level router in the hierar-

chy, however, it is not a restriction. In [**?**], PEs are connected to a single central router, without any hierarchical routers alignment. Such an approach leads to routing problems especially in FPGAs, which was pointed out by the authors.

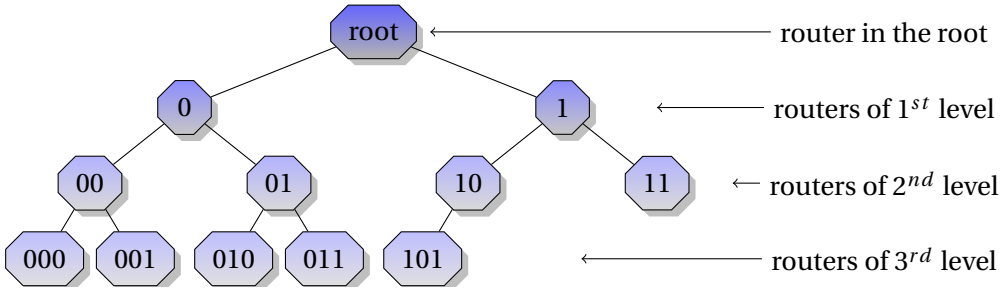


Figure 2.6: Exemplary binary tree topology. For clarity, only the routers are depicted. PEs can be attached to any of the free ports of a router, typically, to the lowest level router in the branch. Each link is two way (⇆) connection.

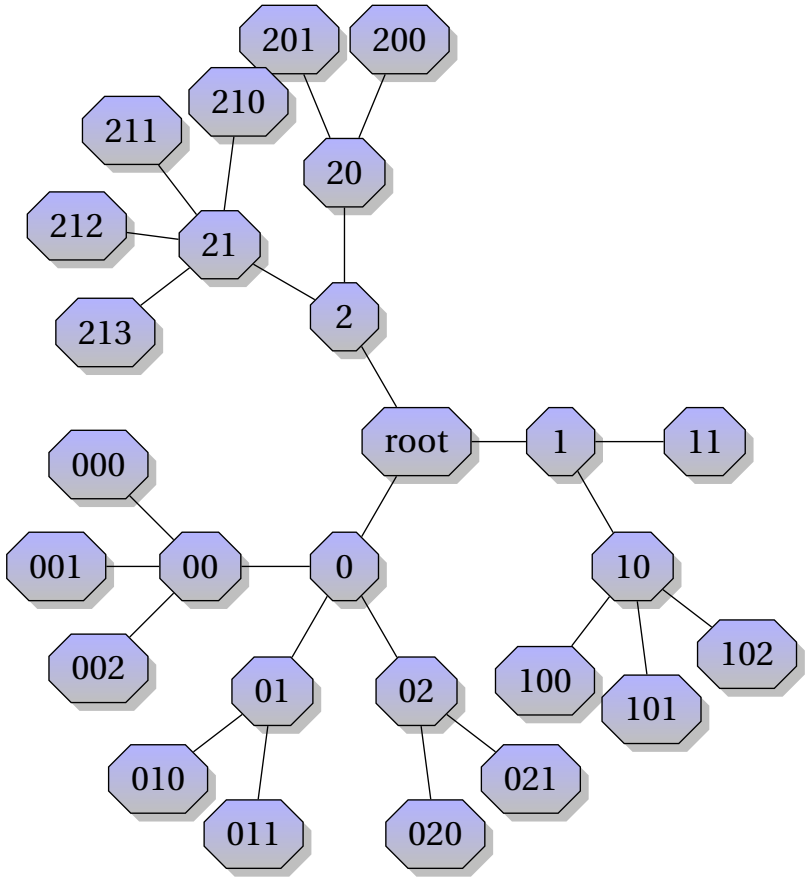
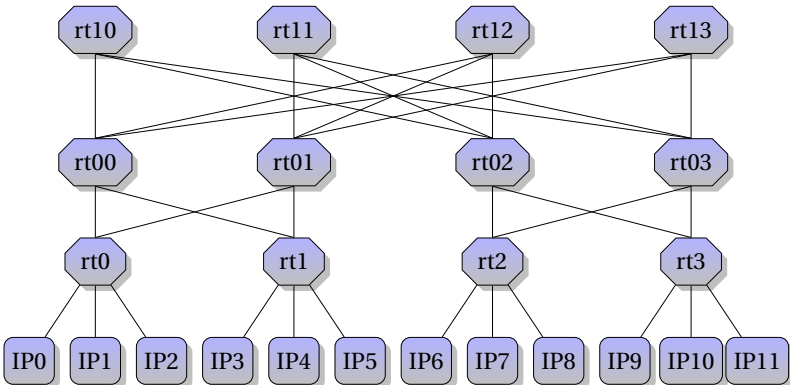


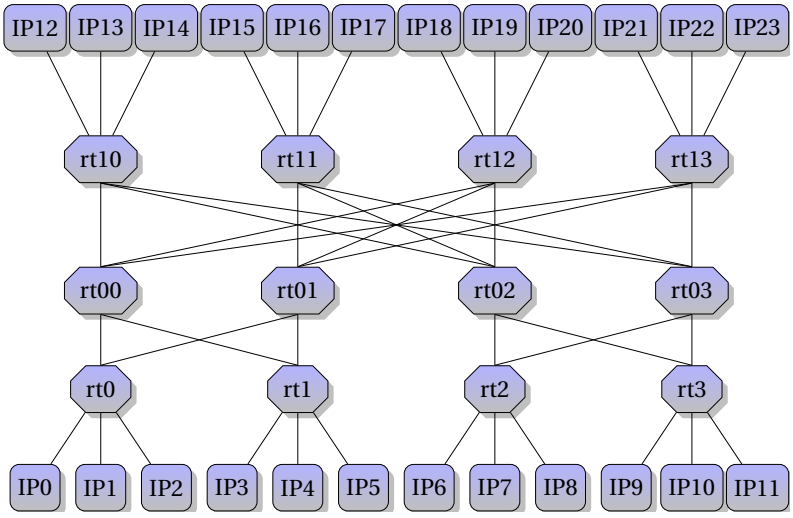
Figure 2.7: Exemplary star topology. For clarity, only the routers are depicted. PEs can be attached to any of the free ports of a router, typically, to the lowest level router in the branch. Each link is a two-way (⇆) connection.

Fat tree and butterfly

Another hierarchical topology is a fat tree which is shown in figure 2.8 used in [**? ? ?**]. Similarly to the star and binary tree, the lower and higher level nodes can be distinguished. However, in the case of the star and binary tree, PEs may be connected at each level of network devices, whereas in the fat tree, higher level nodes connect only switching devices from the distant parts of network.



(a) Fat-tree



(b) Butterfly

Figure 2.8: Exemplary fat tree and butterfly topology. Each link is a two-way (⇆) connection.

Butterfly topology is similar to the fat tree one. It is formed by adding routers with PEs to the highest level routers in the fat tree, as shown in figure 2.8(b). Depending on the implementation, there can be a different number of PEs connected to the low level router: in figures 2.8(a) and 2.8(b) there are three PEs whereas in [**?**] there are two nodes and in [**?**] there are four nodes.

Moreover, fat tree and butterfly topologies may have a different number of levels, namely the levels of routers used to connect switching devices [**?**].

Ring

One of the simplest, but not widely spread [**?**] topologies is the ring topology, which consists of switching devices, each connected to 2 others. The ring topology (e.g., [**? ?**]) may consist of 1 or 2 logical rings as shown in figure 2.9.

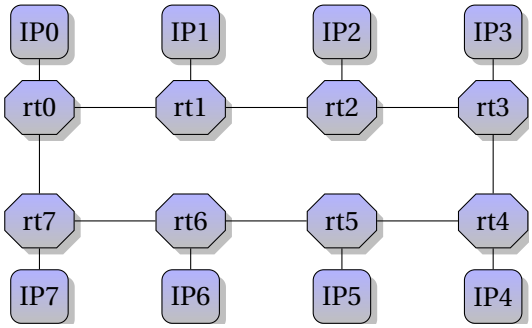


Figure 2.9: Exemplary ring topology. Each link between two routers may be a two-way (⇆) or a one-way (→) connection.

2.3.2.2 Topology comparison

Application performance is different depending on the chosen topology and the placement of elements in it. Further investigation needs to consider not only high level topology characteristics but also physical layout problems such as physical path delay. The importance of this approach is presented in [**?**], where the authors implement a set of topologies for high and low level comparison. The results obtained show significant differences between these two approaches. High level analysis proves theoretical capabilities of topologies which show that 2D mesh is outperformed by several topologies. However, when physical layout characteristics are taken into account, this topology becomes the best solution.

2.3.3 Routing protocol

There are two types of path determination: deterministic and adaptive. In the first case, there is a constant path between the source and destination. On the other hand, in adaptive routing, the path may vary during operation of the application.

Another distinction between path determination protocols is source-based and distributed routing. In the first case the whole path is known in the source-node, and the packet contains information about it. The routers read the packet header and follow orders contained in it. In the latter case, the network decides on the path in each router independently [**?**].

In TCP/IP networks there is adaptive, distributed routing. The paths are calculated using a routing protocol, which enables path determination form the source to destination node independently for each router. The routing protocol involves exchanging routing tables or sending information about link state. Such a complicated task in the case of TCP/IP networks is an operation of a hardly noticeable cost, in terms of time and hardware consumption. In the case of NoC, the addition of any functionality requires adding a significant amount of hardware. It also puts higher requirements on bandwidth because the size and frequency of the exchanged routing data may be comparable to the amount of data that are sent over NoC. It means that the designer's decision whether to implement adaptive routing needs to be well-justified in effectiveness gains of the network.

Adaptive routing also means that there is a possibility of sending sets of packets through different paths. It results in packets buffering in the receiver, and ordering them according to the sequence they were sent in. Moreover, the more complicated the routing algorithm implemented, the more hardware it consumes. Hardware is exploited not only on the functionality being implemented, but also on the necessary storage, such as memory blocks to store the routing table or to buffer a packet. Another disadvantage of adaptive routing algorithms is forwarding of the routing data (e.g., routing tables), which consumes bandwidth.

Adaptive routing in NoC may be justified in general purpose design [**? ? ?**] or to balance power consumption [**?**]. With general purpose NoC e.g., a Chip MultiProcessors (CMPs), the designer does not know the final functionality that would be loaded on a chip, therefore a more complicated routing protocol, i.e., one that enables gathering information on modules in the network and forwarding routing tables, seems to be justified [**?**]. Adaptive routing also allows for balancing bandwidth consumption, which helps to improve network performance.

2.3.3.1 Deterministic (nonadaptive) routing

Deterministic routing can be implemented in two ways: with predefined routing tables or with the use of an algorithm, that will route packets from the specific source to the destination always on the same path. The first solution assumes that routing tables are loaded at the time

of loading NoC on a chip. If a special port is provided e.g.[**?**] for off-chip configuration of the network, routing tables could be reloaded with use of special control packets, sent by an off-chip device. Another deterministic routing algorithm is e.g., Dimension Order Routing (DOR), which determines the path to the destination based on a destination address that indicates its geographic position in the topology.

The deterministic routing algorithm always offers the same path for every source – the destination pair, which means that the order of packets sent in the source is the same as the order of packets received in the destination. Hence, there is no need to send order indication in the packet header, nor buffer packets in the case of reordering. From the designer's perspective it means saving hardware and latency.

Dimension Order Routing (DOR) protocol

The example of DOR routing algorithm is XY routing, used in the mesh-type topology [**?**], in which every node address is at the same time an indication of XY position in the mesh. When the router receives a packet, it forwards it at first in X direction as far possible, and then in Y direction. Algorithms for DOR in other topologies are different, nevertheless, they depend on the same assumption which is that the PE address indicates position in the network. Such an algorithm works only with regular topologies, with a known arrangement of the nodes.

IP core registration protocol

Registration protocol (proposed in [**?**]) allows for acquiring data in routers about the network structure and PEs' functionality. Routing tables built in a logically hierarchical network provide deterministic routing. Logical hierarchy in the network means that the router contains one port, which is designed for communication with higher-level (logically) part of the network. In [**?**] the physical topology is the tree, however, this routing protocol is not restricted to it. It also allows for the routing and creating of ad-hoc networks. After power-on every node in the network sends a registration packet, in which it informs the router about its own address and functionality. This address and PE description code is sent to the router in the root of the network where it is discarded. All routers along the path update their routing tables. After the registration of all PEs the router contains information about the whole network. Also, if an PE does not know its own address or address duplicates, the root sends backward a packet with PE source address update, which updates the routing tables in the routers along the path. The root is also a host of the last

resort. If any router receives a packet of unknown destination, it passes it upward in hierarchy. If the root of this network does not know the destination, it sends a reply packet to the sender that the destination is unknown. Otherwise, the root sends PE its source address, with a packet that updates routing tables along the path.

The presented protocol results in a deterministic routing scheme, which does not require predefined routing tables and is flexible to node positioning in the network. It only imposes introducing explicit (e.g., tree topology) or implicit (indication of the port to the root in every router) hierarchical alignment of the routers.

2.3.3.2 Adaptive routing algorithms

In the networks for general-purpose functionality e.g., CMPs, a tolerance on fabric faults or congestion routing is required. Some implementations such as [**?** **?** **?**] are designed for 2D mesh topology, others, such as [**?**], allow for some irregularities in the network. Also, most of the congestion-tolerant protocol proposals, such as [**?** **?** **?**], are designed for 2D mesh. The mesh topology is the most common [**?**] topology for NoC and due to its regularity it is easy to compute and implement alternative routes. Adaptive routing introduces additional hardware cost, due to the computations involved in determining an alternative path. In low-scale networks that connect modules of e.g., single video encoder or decoder, the tolerance on faults or congestion does not justify the costs.

2.3.4 Flow control

In NoC, similarly to TCP/IP networks, there are three types of flows: unicast, multicast and broadcast. Unicast is a data flow that involves a single source and single destination. In the case of multicast and broadcast transmission there is a single source sending data to a few nodes (in the case of multicast) or to all nodes in the network i.e., broadcast transmission. The authors in [**?**] define 3 methods to implement multicast or broadcast transmission. The first is sending multiple unicast messages to a set of destined nodes in the network. The second is to create a single path that reaches all the destined nodes for transmission so all the destined nodes are able to receive the packet and resend along the path. The third method is to send a single packet and in the case of splitting routes for different destinations the packet is copied and sent in every direction. In the TCP/IP networks multicast transmission is realized using only the third method [**?**].

The need for flow control appears in the case when the source is sending data too fast, which means that the chosen destination is not able to receive them at the same pace [**?**]. Such an unmatched sending and receiving speed causes the buffering of packets, at first in the destination's receiving buffer, and after the buffer overflows, in the routers along the path. Such a situation leads to the increased latency of other packets in the network. The goal of flow control mechanisms is to prevent such situations from happening. The designer may introduce one of the three traditional flow control mechanisms [**?**]: on-off, ACK/NACK and credit-based.

On-off flow control

This is the simplest flow control mechanism based on sending control packets with a message, that turns transmission off or on [**?**]. This technique is used in RS-232 standard[**?**]. If the destination is not able to receive more packets, it sends a message to the source that turns the transmission off. When it is able to receive the data again, it sends a packet turning the transmission on. This mechanism does not allow the source to control the amount of data sent. Moreover, this technique does not prevent the occurrence of congestion, it only enables the mechanism when it already happens.

ACK/NACK flow control

In order to prevent the destination buffer from overflowing, a mechanism can be introduced that employs sending an acknowledgement (ACK) or denial of acknowledgement (NACK) messages from the destination to the source, similarly to the one implemented in Transmission Control Protocol (TCP). This technique was used in [**?**] and allows for the source to be informed whether the packet or a set of packets sent were received. The source is not allowed to send more data without a confirmation of receipt of the previously sent packets. This mechanism also may introduce some negotiations, on how many packets the source may send without receiving the ACK message.

The presented technique is flexible and allows for a detailed control of the data flow, however, it introduces much hardware and bandwidth overhead, especially in NoC due to its complexity. Depending on the application implemented, the amount of data sent due to flow control may be comparable to the data that is to be sent from the source to the destination[**?**].

Credit-based flow control

Credit-based flow control is based on the restriction that data can be sent through the network only if it can be immediately consumed at the destination point. Before sending the data, the source asks the destination about the amount of credit available, and after receiving it, it sends no more than that amount of data without the confirmation that the packets were absorbed. The destination sends backward the information about the increased or decreased amount of credit. At the end the source sends the end of transmission message. Crediting has been implemented in e.g., [**?** **?** **?**], and may vary depending on the implementation: single or dedicated buffer for each source-destination communication.

Crediting is an efficient tool for flow control that requires minimal hardware for algorithm implementation and small bandwidth consumption.

2.3.4.1 Quality of Service (QoS)

In a network that connects PEs of different functionalities and requirements on network performance, there is a need to introduce QoS so that all services could get satisfactory transmission parameters. In order to implement QoS, different network parameters such as latency, bandwidth, jitter etc. need to be selected and judged on how much the different services tolerate changes in those parameters [**?**]. In [**?**] and [**?**] the authors propose a division of network traffic into classes, and assign requirements on network parameters of each traffic class. In the NoC that contains small number of the PEs (according to [**?**] it is about 25 nodes) e.g., that connects PEs in single video encoder or decoder, QoS implementation is not necessary. In such network, the traffic between the nodes can hardly be diversified into classes, due to the fact that the data transfers mostly have the same characteristics. Hence the QoS methods are not cost effective, and the traffic control can be performed with simpler techniques, such as a credit based flow control.

CHAPTER III

AVC codecs and their implementations in hardware

Advanced video compression technology is the state-of-the-art technology in multimedia industry. This technology is also the reference technology for research [**? ?**]. Advanced video compression standards such as Advanced Video Coding (AVC)[**?**], Video Coding Standard-1 (VC-1)[**?**], Audio Video Standard (AVS)[**?**] and currently developed High Efficiency Video Coding (HEVC)[**?**], are hybrid codecs that use prediction (intra- or inter- frame) and cosine transform coding to compress a prediction error. Hybrid coding scheme is the only scheme that is widely spread in practice [**?**]. Although a new video compression technology called HEVC is currently being developed, AVC is still the dominant compression standard [**? ?**]. Hence, further video compression considerations concern the AVC standard but conclusions are also valid for other advanced compression standards.

3.1 AVC standard overview

Video sequence is a set of pictures (frames). Each frame consists of one or more slices, which are the smallest parts of AVC stream that can be decoded independently. Prediction of each slice is carried on its small fragments called macroblocks - equal-sized squares (in case of AVC it is 16x16 picture points). There are two types of predictions: intra- or inter- frame. In the first case, the contents of macroblock is predicted from the samples of the same slice as being currently encoded. In the latter case the current macroblock is predicted from previously encoded frames [**? ? ? ? ?**].

In order to restrict interframe prediction range, Group of Pictures (GOP) was introduced. Each group begins with an intra-predicted frame, which is followed by a set of inter-predicted frames. A description of GOP is provided in 3.1.5.

The following standard overview concerns the main profile of AVC.

3.1.1 Intraframe prediction

In the AVC coding scheme, during the intraframe prediction process, the contents of the current macroblock is predicted based on the previously processed samples of the same frame [**?**]. This process can be performed using one of the two partitioning schemes:

- 4x4 - macroblock is divided into 16 equal-sized squares (4x4 picture samples). Each part of the macroblock is predicted independently,
- 16x16 - intraframe prediction process is performed for the whole macroblock.

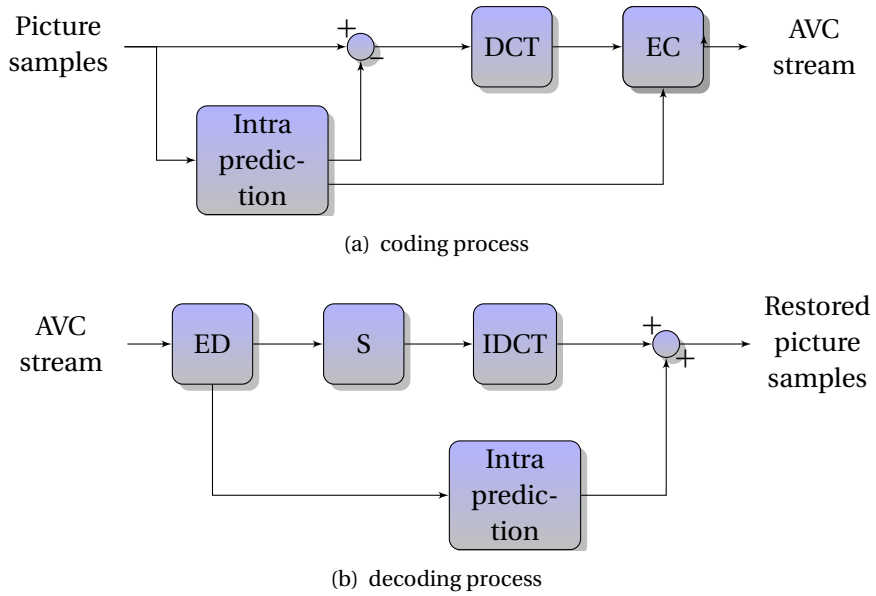
Both partitioning schemes use samples from the partition border to build the context used in the prediction. AVC coding standard offers 9 or 4 (for 4x4 and 16x16 partitioning schemes, respectively) prediction schemes that can be chosen in the intraframe prediction process. Figure 3.1(a) shows an intraframe encoding scheme. The output of the process is a prediction error, which is encoded using cosine transform and quantized (in order to remove irrelevant information).

In the decoding process (see figure 3.1(b)) the macroblock is predicted upon the context (partition border samples). Also, the prediction error is scaled (due to quantization) and inverse transform is calculated. At the end, the predicted macroblock and the belonging error are added together.

An intrapredicted macroblock is called macroblock I. Frames that consist only of these macroblocks are called intra frames or frames I.

3.1.2 Interframe prediction

In interframe prediction process, currently encoded macroblock is predicted from a macroblock in previously encoded frame [**?** **?**]. It is based on the assumption that frames which are not distant from each other, in the sense of time, share similar content. The difference between the frames comes from the fact that objects in the picture may move. Consequently, the contents are mostly the same, with some correction on the object movement. In the AVC interframe prediction process, the encoder looks for the most similar macroblock (not an object) in the previously processed frames. This process is called motion estimation. As an output of the encoder, it provides a reference frame indicator, motion vector and prediction error. The reference frame contains a macroblock, based on which, current macroblock samples are being predicted. A motion vector gives a relative to the currently encoded macroblock position of the reference ma-



(a) coding process

(b) decoding process

Figure 3.1: Intra frame prediction scheme. DCT, IDCT — Discrete Cosine Transform, forward and inverse, respectively; Q,S — quantization and scaling (dequantization), respectively; EC, ED — entropy coding and decoding, respectively.

croblock. The value of the motion vector is predicted, based on motion vector values of the surrounding partitions and/or macroblocks, and in the bitstream the prediction error is encoded. Interframe prediction error, as in the intraframe prediction scheme, is encoded using cosine transform and quantized in order to remove insignificant information. The interframe encoding process is shown in figure 3.2(a).

Decoder (see figure 3.2(b)) calculates the predicted value of the motion vector. Then it is used (with the frame indicator) to obtain the value of macroblock samples in the previously decoded frame. Next, the interframe prediction process is evaluated and the error (after scaling and inverse transform) is added.

Motion vectors may be estimated and transmitted with the accuracy of the whole, or½, or ¼ of the sampling period. In order to obtain sub-sample accuracy interpolation on reference macroblock samples is performed.

Macroblock partitioning

In order to achieve better compression results, each macroblock can be divided into partitions, as presented in figure 3.3(a)[**?**]. Partition 8x8 samples can be divided further, according to the scheme presented in figure 3.3(b)[**?**]. A motion compensation process is performed for each

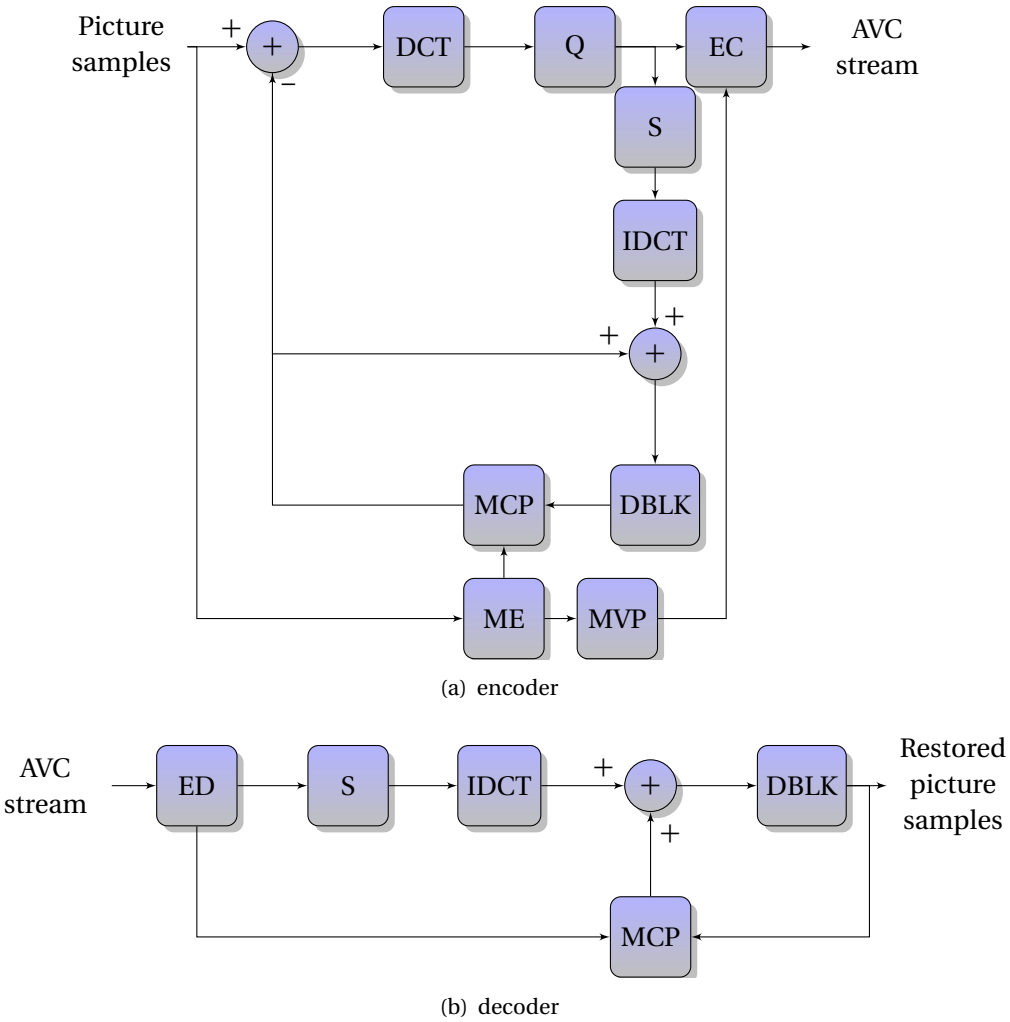
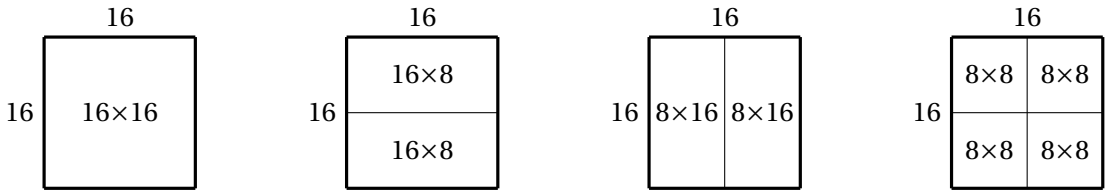


Figure 3.2: Interframe prediction scheme[**?**]. DCT, IDCT — Discrete Cosine Transform, forward and inverse, respectively; Q,S — quantization and scaling (dequantization), respectively; EC, ED — entropy coding and decoding, respectively; ME — motion estimation; MCP — motion compensated prediction; MVP — motion vector prediction.

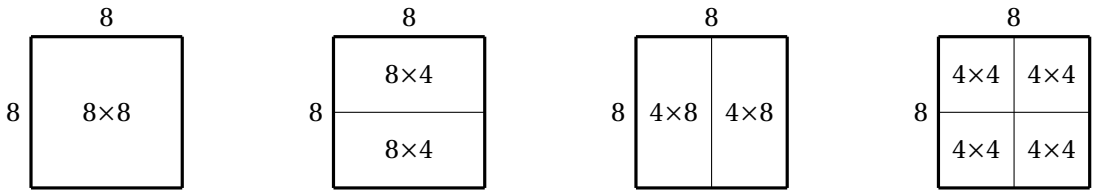
partition independently. Large partitions (16x16, 16x8, 8x16, and 8x8) of a macroblock, may have different reference frames. However, small partitions i.e., 4x8, 8x4 and 4x4 share the same reference frame. Presented partitioning scheme allows for accurate motion compensation, nevertheless, it is computationally complex and time-consuming [**?**].

Types of interpredicted frames

There are 2 types of interpredicted frames: P and B. P frames consist of macroblocks whose reference frame is the previous frame in the video sequence as shown in figure 3.4. Interpredicted macroblocks in frames P are called macroblocks P[**?**].



(a) big partitions



(b) small partitions

Figure 3.3: Macroblock partitioning for interframe prediction[**?**]

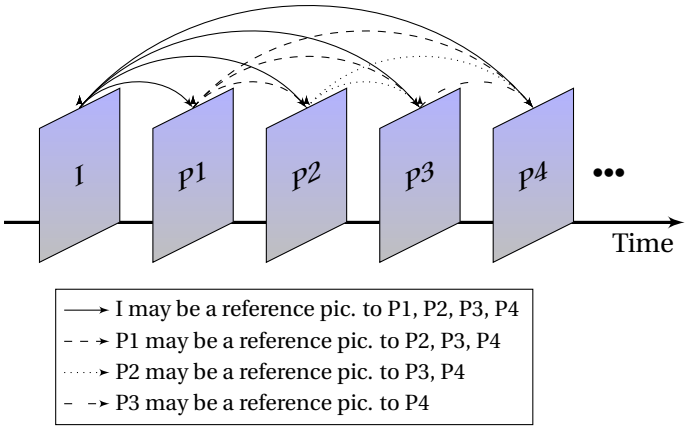


Figure 3.4: Reference frames for P frames

However, the prediction is more accurate if the information used in the prediction process can be obtained from the future frames as shown in figure 3.5. Frames, in which interpredicted macroblocks may refer to the future frames, are called frames B (as an abbreviation from bidirectional), and interpredicted macroblocks in such a frame are called macroblocks B[**?**]. In this coding scheme the order of encoding pictures is not the same as in the original video sequence (see figure 3.5). Macroblocks B may have more than 1 reference frame. In this case, the encoder needs to calculate 2 motion vectors, and compute the prediction, based on 2 reference macroblocks.

Figure 3.6 illustrates a visual comparison between compression efficiency of three coding schemes: I, P and B. Interframe prediction, especially bidirectional, is very efficient in terms of

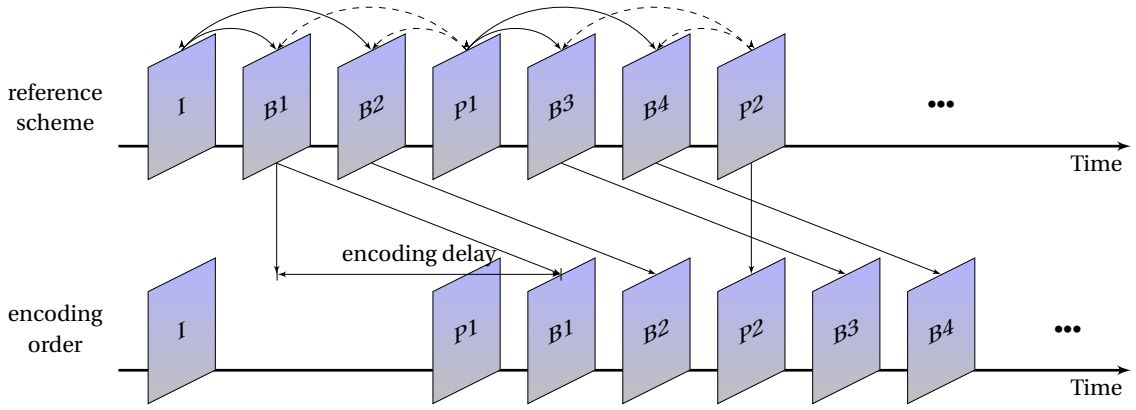


Figure 3.5: B frames reference scheme and encoding order

compression rate of a video sequence, however, it consumes about 40–70% of computational power of an encoder [**?**]. On the other hand, frames I are the fastest in encoding but the compression rate is much lower than in the case of interpredicted frames.

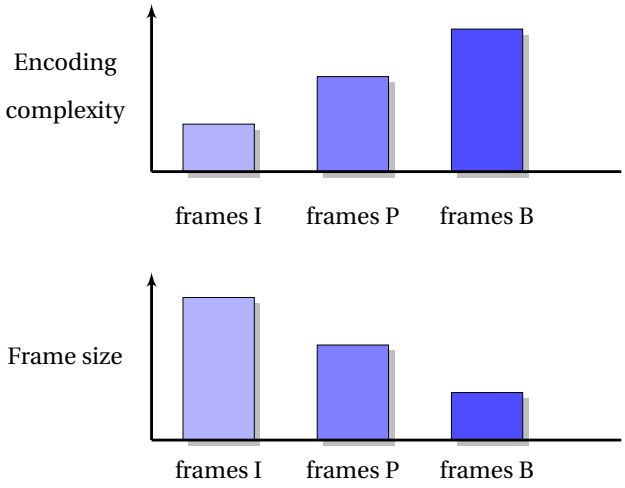


Figure 3.6: Comparison between encoding complexity and efficiency of different frame types [**?**]

Table 3.1 presents which macroblocks types may be chosen by the encoder in the course of the picture encoding process. In frames I only macroblocks I are allowed. In P frames, however, both macroblock I and interpredicted macroblocks that refer only to one of previous frames are allowed. Similarly in frames B, macroblocks I are allowed and interpredicted macroblocks that refer to the past and/or future frames.

Table 3.1: Macroblocks allowed in individual frame types

Frame type	Macroblocks allowed
I	only macroblocks I (partitioning: 4x4 or 16x16)
P	macroblocks I (partitioning: 4x4 or 16x16) macroblocks P (partitioning: as shown in figure 3.3)
B	macroblocks I (partitioning: 4x4 or 16x16) macroblocks B (partitioning: as shown in figure 3.3) each big-partition reference to the past and/or future frame

3.1.3 Transform, quantization and deblocking filter

There are two types of transform in AVC: 4x4 and Hadamard transform [**?** **?**]. Hadamard transform is available only in the case of macroblocks I encoded with the use of a 16x16 prediction scheme. At first, 4x4 cosine transform for 16 blocks in macroblock is performed. Then all the 16 DC coefficients (i.e. average signal values for each 4x4 block) from the whole macroblock are gathered, and Hadamard transform is calculated.

Quantization is performed on cosine- or Hadamard- transformed prediction error i.e., on transform coefficients. In AVC, quantization step is configured indirectly through Quantization Parameter (QP), which may take values between 0 and 51. The higher the QP value is, the greater compression occurs, and at some value, a blocking effect is visible. The blocking effect is an image distortion, which is an artificial border at the edge of the transform block. In order to reduce this effect, deblocking filtration can be performed as described in 3.1.3

In order to enhance picture quality, filtration is done on the edges of blocks that conform to the transform edges[**?**]. This operation is performed by deblocking the filter in the encoder and decoder on restored frames. Since the output of the filter is used in further encoding/coding process, as a reference, it is an in-loop filtration (i.e., it is in the coding/encoding loop), and slows down the speed of the AVC codec. Deblocking filtration is optional in the encoding process, and is performed in the decoder only if the AVC stream was produced with this option.

3.1.4 Entropy coding

Video encoder produces the following symbols:

- set of parameters which describe how to decode a macroblock,

- motion data (in case of macroblocks P and B) i.e., reference frame indices and motion vectors,
- transform coefficients.

Motion vectors are further encoded, based on the fact that the neighboring macroblocks often have similar motion vector values. Based on the context, a prediction of the motion vectors is performed, and the difference is sent to the bitstream encoder.

Symbols in AVC are encoded using variable length coding (Universal Variable Length Coding (UVLC)/Context-adaptive Variable Length Coding (CAVLC)) or arithmetic coding (Context-based Adaptive Binary Arithmetic Coding (CABAC))[**? ? ? ?**]. The UVLC/CAVLC encoder uses a dictionary to define which value is more or less probable. More often symbols obtain shorter bit codes, whereas symbols with lower probability are coded using longer bit codes. CAVLC exploits context-adaptive codes and is used only to encode transform coefficients. In the case of UVLC each parameter or motion vector has its own, predefined dictionary of bitstream codes, that is not adapted to the previously coded content. Entropy codec in the AVC compression standard encodes or decodes the following:

- binary word of variable length of 1–31 bits,
- exp-Golomb codes (signed or unsigned),
- Huffman codes stored in a memory block.

UVLC/CAVLC bitstream codes conform to the specific AVC symbols or parameters, which is not true in the case of CABAC. Arithmetic coding used in AVC, utilizes a set of statistic models that are dependent on the coded symbol type and its context[**? ? ?**]. Using CABAC, instead of CAVLC, results in 8-14%[**?**] bitstream reduction. However, it requires more computational power, e.g., in the case of decoding CABAC on personal computers, stream decoding time increases by about 40-70%[**?**] when compared to the UVLC/CAVLC coding.

In AVC compression standard macroblocks are processed sequentially, in lines. It means that at first all macroblocks in the first line are processed, then in the second, etc. Moreover, the only possibility of parallelization is to introduce a few slices within a single frame, which can be decoded independently. Such an approach has two drawbacks. It causes duplications of parser modules and leads to a reduction in compression efficiency, due to context loss at the border of the slices[**?**]. In addition, slice bitstream can only be processed sequentially.

3.1.5 Group of Pictures

The first frame in a coded video sequence has to be an I frame, because it cannot refer to any other frame in the video sequence. Other frames may be coded as P or B frames [**? ? ?**]. In order to decode a P or B frame, all previous frames (in the sense of the encoding order) need to be decoded, so the video sequence coded with a scheme of only one I frame and other P and B frames, has a hardly accessible content. This difficulty grows with the number of frames. The solution is to introduce Group of Pictures (GOP), which is a set of pictures that can be decoded alone, i.e., without the need to decode all the previous frames. Such an GOP needs to start with an I frame and all the references required by P and B pictures cannot exceed the frames in GOP. An exemplary GOP is shown in figure 3.4, and contains one intra frame and four frames P, which is denoted as *I4P*.

3.2 Implementations of video codecs in hardware

There are three important aspects of implementation of video codecs in hardware, that differentiate applications between each other: the division of an application into a set of Processing Elements (PEs), a memory access scheme and a connection strategy of PEs. These aspects have a mutual influence on each other, because defining one restricts the range of solutions for the others. In particular, the definition of PE's functionality determines the traffic pattern and influences the choice of the interconnection architecture. For example, if quantization and a transform block are placed in two separate blocks, there are two PEs with one-way traffic from transform to quantization. If these are placed in a single PE, in a pipeline, from the interconnect perspective, there is less data to send. In the first case quantization and transform modules should be placed close to each other (i.e., with as few hops between as possible). In the latter case, such restriction no longer exists.

Moreover, hardware codecs characterize with a high exploitation of stored data during the encoding and decoding process, especially in the case of encoding frames P and B. In this case memory organization needs to be divided into levels e.g., [**?**]. Some data can be stored locally, for example context information for the intra prediction process in which only the neighboring samples are needed. Nevertheless, the inter prediction process involves a frequent usage of memory with the reference frame buffer. Such characteristics significantly slow down the decoding process and put high requirements on the to/from memory connection.

3.2.1 Division into PE

The division of an application into a set of PEs has a significant influence on the interconnect implementation, because it defines the type and amount of data that is exchanged between modules. Defining the functionality of each module in an application is affected largely by the compression standard implemented. It means that different parts of an application have a defined functionality. Also, the compression standard defines data flows between the particular parts. Each of the data flows means that data need to be sent over some communication infrastructure. Since the wiring is always limited, especially in Field Programmable Gate Arrays (FPGAs)[**?**], hardware implementations are designed with minimization of data flow between modules. Each PE should contain a functionality that accepts a set of data different from other PEs and performs a specific set of calculations e.g., inter- and intra- prediction process[**? ? ? ? ? ? ? ? ? ?**]. Codec implementations presented in the literature propose a division into PEs, which is based on a functional division of the coding process. The division into PEs presented below is used further to define the functionality of modules of a hardware codec described in chapter IV:

- intra prediction,
- inter prediction,
- deblocking filter,
- transform/inverse transform,
- dequant/quant block,
- symbol decoder,
- entropy decoder/encoder.

Each of the listed modules performs a different type of operation, for example intra and inter prediction. Moreover, interframe prediction downloads previously decoded samples to perform calculations. Results of both modules are different. Intraframe prediction produces a prediction mode and prediction error. Conversely, interframe prediction produces a motion vector and prediction error. Similarly, other modules require and produce different data types.

Major differences concern the joining or splitting of some modules e.g. dequantization (scaling) and inverse transform block, e.g., in [**?**] and [**?**] the scaling and inverse transform block are split, whereas in [**?**] these are joined and form single PE. The dequantization module sends data only to the transform, and the transform acquires data only from the dequantization. These

two modules can placed in a single PE in the pipeline. Also integer- and fractional- motion estimation/compensation are split e.g. in [**?**] and [**?**]. In contrast, in [**?**] the authors placed motion compensation in a single block. Moreover, symbol predictors (motion vector predictors, transform coefficients decoder, etc.) and bitstream parser or encoder block are not explicitly mentioned in e.g., [**? ? ?**] and are included in UVLC/CAVLC PE. Also a block that performs the final macroblock reconstruction is not mentioned explicitly, which means that this functionality is embedded in another block, or it is performed at the buffering stage.

Other differences between codec implementations may concern memory architecture, which is not always explicitly mentioned in a proposal. AVC standard exploits many data that need to be stored in the memory for further use, both in the encoding and decoding process. Such a requirement entails the usage of big, external, memory blocks such as SRAM e.g., [**?**]. In order to reduce the number of memory calls, some of these data (e.g., currently coded context) need to be stored in a smaller (i.e., containing fewer video frames) memory block, but located closer to the PE, to reduce data download delay. An example of such an architecture is presented in [**?**]. Since the encoding and decoding process can be divided into a set of stages, some implementations exploit that property [**?**], however the buffer's architecture is not mentioned. Solutions presented in [**?**] and [**?**] use buffers between each stage.

The approaches presented impose different requirements on the interconnection architecture due to varying traffic characteristics generated by modules.

3.2.2 Interconnect architecture for AVC

Basically, AVC is considered as a relatively small design for which a shared bus is suitable. There are few implementations with image processing applications that employ Network-on-Chip (NoC) as an interconnect approach: e.g., [**? ?**]. In [**?**] the authors propose a 2D mesh topology in which an MPEG-2 decoder is part of a greater System-on-Chip (SoC). In the paper mentioned, the authors present only the tile architecture of SoC that contains a multiprocessor, a memory and a Network Interface (NI) connected to a network. There are also implementations of AVC on a predefined architecture e.g.,[**?**], where the authors propose a combination of Digital Signal Processor (DSP) and FPGA devices, whereas in [**?**] the authors propose a combination of DSPs, memories and in-/out-put interfaces.

Other implementations employ one or more shared buses and direct links to the connect processing modules. In [**?**] an AVC decoder is presented, which exploits direct connections

between decoding modules. Shared bus (i.e., 128bit DRAM and 32bit system bus) are used to connect the decoder with cache, DRAM and entropy decoder. [**?**] presents an AVS encoder divided into the processing stages with direct connections between modules. As an interconnect medium it employs DDR SDRAM Bus Interface at the input, which connects DDR SDRAM, video input and system control blocks. Other connections between PEs are direct.

The AVC decoder presented in [**?**] has a pipelined architecture, with direct connections between the main control block and all the processing blocks of the decoder. A shared bus is used to connect the decoder with the memory and the microprocessor. In [**?**] an AVC decoder of scalable video was implemented on shared buses: local and system. The local bus connects the decoding modules with an external memory and a decoded picture buffer. The system bus connects the decoder with external devices such as a bitstream input. The blocks inside the decoder are connected directly. The AVC encoder presented in [**?**] and the decoder in [**?**], contain 2 buses: external and system bus. The system bus is used as an interconnect between the memory and the processing block, whereas the external bus provides connectivity for the whole encoder. Connections between the processing modules inside a decoder are direct. In [**?**] a pipelined architecture of an AVC decoder is presented. Each processing block is connected directly to the next PE in the pipeline, and to the system bus that provides connectivity with the local and external memory blocks.

Although the majority of recent implementations is based on a combination of direct links and shared buses, such an architecture may cause a few problems:

- poor reusability of the processing blocks (because of direct connections) which also limits the possibility of expanding of the application e.g., from the decoder to the encoder. Only the system presented in [**?**] is expanded in such a way (from a system decoder in [**?**]),
- difficulties in debugging of an application based on direct connections,
- potential big loss of bandwidth on a shared bus due to the competition in the access to the medium.

These problems lead to a conclusion that a new NoC architecture, suitable for AVC codecs, is needed. It should offer optional debugging capabilities, as well as a small size of the device compared to the size of the application. This in turn imposes simple routing and flow control protocol that does not consume much hardware.

CHAPTER IV

AVC codec implementation in NoC architecture

4.1 Introduction

In this chapter, an original implementation is described for a codec compliant with the Advanced Video Coding (AVC) technology (see chapter III). The presented codec was being developed from 2004 to 2007, and it was an innovatory proposal. The system proposed was designed for Field Programmable Gate Array (FPGA) devices, however it can be implemented on Application Specific Integrated Circuit (ASIC) devices as well. The author actively participated in the codec's implementation (the author's contribution is presented in section 4.5). The problems the design team needed to tackle remained the same and are concentrated on communication issues. The experience obtained during the design and implementation inspired the author to further study the problems related to the Networks-on-Chip (NoCs). Targeting the design to FPGAs adds restrictions on the design which concern mainly limited resources of FPGA devices [**?**]. From the communication perspective these are a limited number and length of connections. The codec proposed uses NoC because the design team, including the author, believed that it will:

- reduce the number of connections required,
- help overcome problems related to the multichip FPGA design,
- reduce the designing and debugging time.

The codec proposed is divided into 2 parts: the stream parser/formatter and image predictor (see figure 4.1) and was launched on Xilinx Virtex-4[**?**] devices, with the use of evaluation boards ML-402[**?**]. Such a division is a result of implementing the codec on two separate boards due to the ease of debugging of the application.

The architecture proposed is characterized with hybrid connection architecture that contains a segmented shared bus, NoC and peer-to-peer (direct) connections. It could also be implemented on an NoC entirely, however, due to technical considerations discussed in section 4.4, the designing team decided to implement it with the use of different connection strategies. The division into Processing Elements (PEs) is based on the functional division of the coding process (see section 3.2.1). The modules implement different functionalities defined in the AVC compression standard. Moreover, several modules were added to control the encoding and decoding process (i.e., microprocessor) and to store the context during prediction (i.e., picture memory, picture context, and write and read cache).

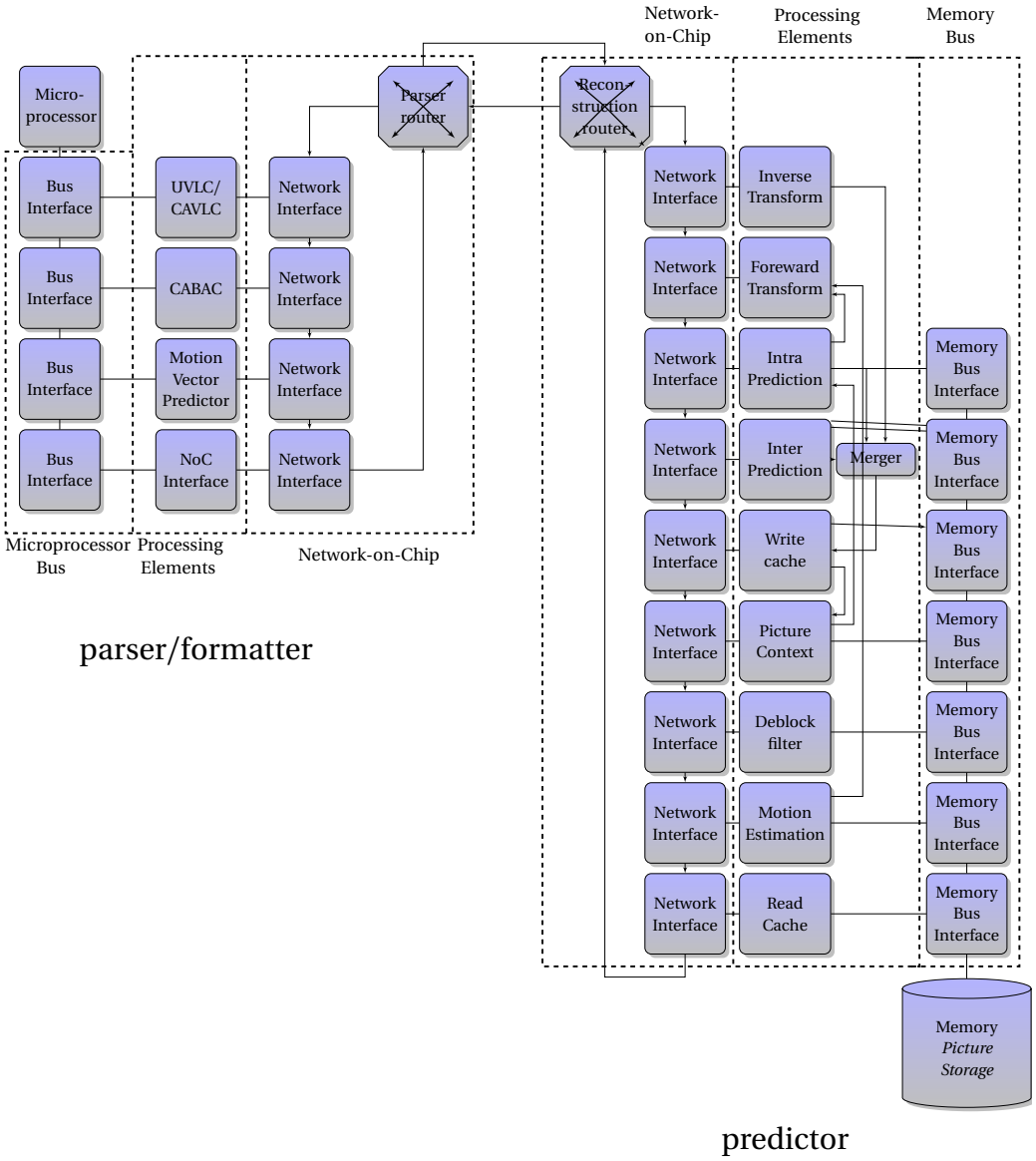


Figure 4.1: Scheme of the codec: each unidirectional link is a two-way (⇆) connection.

Global encoding and decoding data flow

The proposed codec is able to decode an AVC bitstream, or encode video and produce an AVC compliant bitstream. If this codec is used as decoder, the bitstream is loaded to the buffer in the parser. A microprocessor analyzes the bitstream, and sends it for decoding to specialized accelerators. The results of decoding, together with the decoding instructions, are sent to the predictor to restore the image. At the end, samples of the restored image are sent to the memory.

In the case of encoding, the image to be encoded is stored in the memory on the prediction site. A microprocessor controls the encoding process sending commands, how to encode a macroblock, so that the output bitstream is compliant to the AVC compression standard. Prediction blocks receive macroblocks from the memory, and according to the microprocessor commands, send the results to the microprocessor and to the predictor. Then, in the formatter, the prediction results are encoded into a bitstream, with the use of accelerators, and stored in the formatter buffer. A copy of macroblock prediction results is sent to the predictor to obtain a reconstructed image needed for the further encoding as a context (in the case of intraframe prediction) or reference frame (in the case of interframe prediction).

4.2 Bitstream parser/formatter

An AVC bitstream consists of many symbols which control the further encoding or decoding process. In these circumstances, using dedicated modules for bitstream parsing would be inefficient. In the proposed codec, bitstream is analyzed by a microprocessor, which also controls the encoding and decoding process. In order to speed up computations, the microprocessor delegates some encoding or decoding tasks to its accelerators (see figure 4.1). The accelerators are listed below:

- a Universal Variable Length Coding (UVLC)/Context-adaptive Variable Length Coding (CAVLC) codec,
- a Context-based Adaptive Binary Arithmetic Coding (CABAC) codec,
- a motion vector codec.

The accelerators are connected to the microprocessor with the use of a segmented shared bus. Every device has its own bus segment, which minimizes bandwidth competition during the access to the medium. At the end of the bus, a dedicated NoC interface is located. This interface is used only by the microprocessor to communicate with the prediction devices. The accelera-

tors of the microprocessor are connected directly to the NoC, each using its own Network Interface (NI).

Parser/formatter Processing Elements

The entire bitstream decoding processes could be implemented in the microprocessor, however, such an approach would lead to the extensive usage of program memory, and would slow down the encoding or decoding process due to the lack of parallelization. Furthermore, bitstream processing involves many bit operations that are time-consuming in the microprocessors which operate at register level. Bit operations, are easy to perform in dedicated modules in the hardware, which comprises the microprocessor accelerators. The following description covers modules that were designed and verified with the author's active participation.

Microprocessor

The main unit of the parser/formatter is a microprocessor which controls the encoding or decoding process[**?**]. The main problem was that general-purpose processors do not offer an instruction set that is adjusted to the AVC bit stream parsing. This process often requires a decision to be made about the further decoding/encoding process upon a currently coded value. In a general-purpose processor such an operation needs to be split into a set of instructions. The microprocessor instruction set proposed complies with the specifics of the video compression standards and enables the processing of the operation mentioned in one instruction. This advantage speeds up the encoding/decoding process and also reduces program length.

The microprocessor is a 32-bit processor that uses external memory for the data and program. It communicates with its accelerators over a dedicated segmented bus. Over the bus the microprocessor sends commands and data needed to decode or encode a bitstream. If a response is directed to the microprocessor, it travels over the same bus.

The author contributed a code of the microprocessor that controls the process of decoding of the transform coefficients and motion vector prediction. The main problem of this process, was the arrangement of the order of instructions, so that flag values would be available on the instruction execution without a delay. Another issue solved, was the right order data sending to the accelerators without causing a delay in the decoding process, because of context initialization in the accelerators and the reloading of the buffers.

Motion vector decoder

The microprocessor sends macroblock parameters, symbols obtained from the stream decoder, and the macroblock context to the motion vector decoder. Macroblock parameters describe the macroblock position in the image and the partitioning scheme, the current and reference frame type and their index. These data help to interpret the symbols, that are prediction errors of the motion vectors. The result of the motion vectors prediction is sent into two outputs: to the interframe prediction block over the NoC and the to the microprocessor to store them as a future context.

Motion vector prediction consists of a multiple and irregular prediction algorithm and of references to the memory. The algorithm was defined in three modules, each responsible for a single type of prediction. Synchronization of those blocks was resolved by passing readiness signal through the stages of prediction. Such an approach allows for performing calculations in a pipelined manner, even though each prediction type requires a different number of clock cycles to compute the result. Irregularity in references to the context of a macroblock was also resolved with use of a read-only memory block, which defined the relative address of the context for each prediction type. The proposed solution resulted in a small structure that requires 2100 LUT's and 1300 FF's after synthesis.

NoC interface

The microprocessor's NI provides a two-way connection with an NoC and is connected with the microprocessor over a dedicated bus. It means that when the microprocessor sends data, the interface receives them, and translates their format into the NoC format to send them over the network. On the other hand, if the microprocessor is about to receive data from the network, the interface receives packets from the network and translates the data format into the bus format and sends them over the bus, to the microprocessor.

4.3 Picture Processing Element

Image prediction requires more computations than bit stream analysis. In order to accelerate computations, parallelization was introduced at the functional level. It means that intraframe prediction, interframe prediction and transform computation can be performed in parallel, and their results compared and stored almost simultaneously.

Another problem was high data transfer compared to the traffic sent over the NoC, which was at least twice as high. In communication between the parser/formatter and the predictor, compressed data were sent, and between the predictor modules, picture samples were sent. Moreover, the predictor requires downloading picture samples and uploading the reconstructed image form or to the memory. Such a communication scheme resulted in three types of communication infrastructures: NoC, direct links and segmented shared bus which is described in section 4.4.

In table 4.1 data transfers in the image predictor are presented that are independent of the implementation of the codec. These results present transfer sizes excluding memory organization issues. Also the deblocking filter was excluded from the analysis, because its use is optional. Nevertheless, the deblocking filtration process requires a large amount of data of different types [**?**], which is, according to the estimation of the author, 1.5 of the image size, which equals approximately 576 bytes per macroblock.

The constraints presented impose a separation of to/from parser/formatter communication from intraframe predictor data transfers. Also downloading and uploading data from/to the memory should be provided with the use of a dedicated communication infrastructure, to provide enough bandwidth. Therefore, communication between the modules of image predictor blocks is provided with the use of direct connections and communication with the memory operates on a dedicated memory bus, with the protocol adjusted to the memory transfers. The mentioned proposal is presented in section 4.4.

Another problem is related to memory organization. In order to limit data transfers to/from the memory, cache blocks need to be introduced. Furthermore, the context for intraframe prediction can be stored locally due to its size (about 1128 bytes in the case of Standard Definition Television (SDTV) resolution), however, storing it requires reformatting of the data to store only the right and bottom border on a macroblock. A proposal on this issues is discussed in paragraph "Blocks for storing the context".

4.3.1 Predictor blocks

Division into sets of PEs is described in section 3.2.1. The proposed parser/formatter complies with this description. Each particular block in the predictor accepts and performs a different type of computation, e.g., forward and inverse transform block, or interframe and intraframe prediction. In the case of intraframe prediction block, it performs both the encoding and decod-

Table 4.1: Data transfers between blocks of image predictor per macroblock. All headers, to/from parser/formatter communication and writing reconstructed samples are excluded

Operation type	Data type	Amount of data [bytes]
Decoding process	intraframe prediction or interframe prediction error	432
	intraframe prediction or interframe prediction result	432
	reconstructed image samples	384
	context for interframe prediction (only downloaded picture samples)	150 - 900
	context for intraframe prediction (only downloaded picture samples)	32
Encoding process	data transfers of the image predictor (to obtain a reconstructed macroblock as a reference)	1 280 - 2 180
	intraframe prediction and/or interframe prediction result to perform scaling and forward transform	432 - 862
	the context for motion estimation	at least 384, depends on the number of reference frames and search area
	the encoded image samples	384

ing process due to algorithm regularity.

Predictor blocks perform the actual decoding and encoding process. The predictor is divided into PEs as follows: encoding or decoding macroblocks I, interframe prediction, motion compensation, transform (forward and inverse), deblocking filter and storage of the context.

The microprocessor controls the prediction process and sends a command to Intellectual Property cores (IP cores), that contains the scheme of prediction to choose. In the case of decoding, the results are sent to the memory. On the other hand, in the case of image encoding, the results are sent to the microprocessor for entropy encoding and bitstream forming.

Predictor block for macroblocks I

The prediction of macroblocks I is performed in the same PE, both in the encoding and decoding process. In the case of decoding, the microprocessor sends the decoding instructions over NoC, that contain the prediction scheme. Then, with the use of the prediction scheme received and macroblock context, macroblock samples are restored and sent to the merger block (over a direct connection) for adding a prediction error.

In the encoding process, macroblock samples are received from a buffer and upon microprocessor commands the prediction is performed. The result of the prediction is compared to the original macroblock samples to obtain the prediction error, which is sent to calculate transform and quantization. The prediction scheme is sent to the microprocessor for coding into the bitstream.

Interframe prediction block

The interframe prediction block receives the motion vector and reference frame index from the parser. Based on this data set, the context (which in fact consists of samples of the macroblock referred) for prediction is downloaded from the memory over the memory bus. Then, upon these samples, macroblock is restored. The result is sent to the merger block (over a direct connection) for adding a prediction error.

Merger

Merger is an IP core that adds a prediction result to the prediction error. The prediction result comes from intraframe or interframe prediction blocks over a direct link. A direct link is also used to transport the prediction error (inverse transform result). The sum of the two is restored, and macroblock samples are sent to the local memory, to buffer them as the context for the next macroblock, before sending them to the memory.

Motion compensation block

Motion compensation IP core estimates a position (in the reference frame) of the most similar macroblock to the one currently encoded. The most similar macroblock may have a different position in the reference frame, and this difference is the motion vector. Searching for a similar macroblock is performed with integer sample accuracy in the area of a macroblock position with

an added margin of 8 samples in the vertical and horizontal dimension. Only one macroblock partitioning scheme is evaluated i.e., 16x16 samples. At the end the motion vector is predicted upon the values of the neighboring motion vectors and the result is used for the further encoding of the motion vector prediction error. The motion compensation error is sent to the transform IP core over a direct link. The motion vector prediction error is sent to the microprocessor and interframe prediction IP core together with the reference frame index with the use of a multicast transmission mode over an NoC. In the microprocessor it is coded into a bitstream, and in the interframe prediction IP core it is restored to obtain a reference image for further encoding.

Forward and inverse transform blocks

The forward transform block computes transform coefficients on the prediction error. The prediction error is received from intraframe or interframe prediction blocks over a direct link. Then the transform is computed and scaled according to the parameters sent by the microprocessor. The results are sent over an NoC to the microprocessor for entropy coding.

The inverse transform block does the opposite operation. It receives transform coefficients over an NoC, scales them and computes the prediction error. This result is sent to the merger to be added to the prediction result in the decoding process in order to restore macroblock samples.

The author contributed to the design and implementation of the scaling (dequantization), and inverse transform block implementation [**?**]. These two blocks are joined since the first one produces an input for the latter as described in section 3.2.1. The main challenge was to design these modules to enable computations in a pipelined manner, and therefore to speed up the overall computation time for a sequence of macroblocks. The proposal is discussed in [**?**] in detail. Computations are performed in each dimension of the transform independently, one after another. There was also an issue of reorganization of the order of samples to compute second transform dimension on a copy of modules that perform calculations of the first dimension. Such an approach facilitates the debugging process and shortens the design time. Moreover, the proposed architecture is able to perform calculations for all types of transform available in the AVC standard within the same modules.

Deblocking filter

A deblocking filter is used in both, the encoding and decoding process to reduce edges between transform blocks that appear as a result of strong compression (using high Quantization Param-

eter (QP) value during the quantization process). Filtration is done over the edges of the 4x4 blocks. At first all the vertical edges are filtered from left to right and then the horizontal edges from top to bottom. The result is stored in the memory for context.

Blocks for storing the context

The memory is organized in three levels: local context, cache and memory to store frames. The local context is a memory of up to 2kB in size, in which data are stored for prediction e.g., context in intraframe prediction IP core that stores a context for the current prediction. The second level of context are the modules that store and communicate with the memory. These are called caches and are described below. These modules also prepare data for further use in the modules, i.e., to reorganize data order or filter only the useful data as an intraframe picture context. The third level of storage is the external memory block, that stores decoded pictures, context data for deblocking filtration and picture samples in case the of encoding. Communication with this block is carried out on a shared bus, with a simplified protocol, adjusted to the memory reads and writes, described in section 4.4.1.

Reading memory block - reads a specific macroblock from the memory for encoding, as it is specified in the command received from the microprocessor. Read data are sent over a direct connection to the motion compensation and to the intraframe prediction for encoding.

Writing memory block - receives restored macroblock samples and sends them to the intraframe prediction context and to the memory for storage. This module is active in the encoding and decoding process, since a restored macroblock appears in both processes. In decoding it is a result, in encoding it is a reference for further prediction.

Intra prediction context - stores image samples that are a context for intraframe prediction mode. These samples are only from the right and bottom border of the macroblock. Samples from the right border are used to encode or decode the next macroblock in a line, whereas samples form the bottom border are exploited in the intraframe prediction of the macroblock below the currently saved one. This module is used in both the encoding and decoding process.

4.4 Communication infrastructure

Modules send different traffic characteristics and have different demand on the connection speed. This results in differences in communication schemes and types of interconnect:

- a bus, that connects the microprocessor with hardware accelerators
- a memory bus, a simplified bus adjusted to the communication with the memory,
- direct connections between the predictor,
- NoC, that provides parser/formatter-to-predictor communication

The microprocessor and memory communication was designed as a segmented shared bus. The proposed bus protocol results in little hardware consumption (see synthesis results in table 4.3) and the segmentation reduces bandwidth competition. The bus protocol in the case of the memory and microprocessor are similar, however, the memory bus has a simplified protocol adjusted to the reading/writing of data blocks. Direct connections were introduced to connect PEs on the predictor site, to send large amounts of data which are: macroblock samples, prediction error and the context. This transfer is cumulated in time, at the end of a macroblock processing interval, which would block NoC if this traffic should be directed over a network. NoC was introduced to ensure communication between the two major parts of the codec, that can be placed on different FPGAs or even on different boards.

4.4.1 Shared bus

Shared buses used in the codec provide quick data transfer on a short distance. They are 32 bit in width, with segments associated to each device on the bus to reduce bandwidth competition. The difference between the bus used by the microprocessor and the memory bus is the reduced protocol for memory transfers. On the other hand, the microprocessor bus protocol is more complicated and allows for sending more parameters and commands from the microprocessor to the destined accelerator.

4.4.2 NoC

Network-on-Chip (NoC) is used for communication between the parser/formatter and predictor. Its links are 11-bit wide (8 bits for data, 3 bits for signaling), and the topology used is ring [**?**]. A packet may travel from a router, along the NI chain and it is looped back, if it is not received by any PE. In the ring (see figure 4.2), there is a router that enables communication with other

rings and a chain of NIs, that send or receive data. Each NI contains a multiplexer that enables switching capabilities, and allows to use a chain of NI instead of routers. Such an architecture saves much hardware, which is beneficial especially in the case of FPGAs. Each router contains two input and two output ports and routes the packets according to the routing table. Routing tables are static (i.e., they are built-in a router, and are not exchanged by routers), however they can be reconfigured with a control packet. The proposed NoC is based on a deterministic (nonadaptive) routing algorithm because of its low hardware consumption.

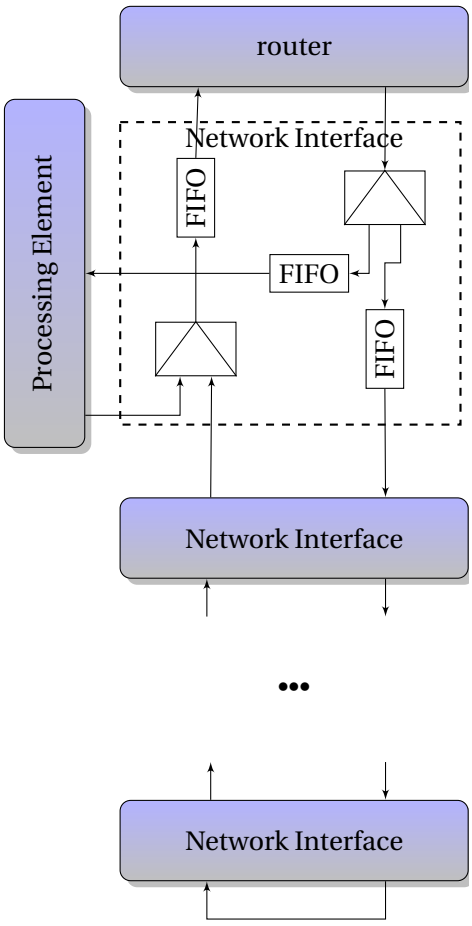


Figure 4.2: Proposed NI structure

4.4.3 NoC — benefits and limitations

To discover the benefits and limitations of NoCs, there is a comparison to other connection techniques performed. This comparison is accomplished from two perspectives: implementation and design. A comparison of hardware implementation properties is given in table 4.2. The table also contrasts direct connections, shared bus and NoC, based on the experience the au-

thor gained in the course of implementation of the AVC codec. Table 4.4 compares the design properties of direct connections, a shared bus and NoC.

The first row in table 4.2 compares the size of the interface required by all the three connection techniques. Specific numbers are given in table 4.3, and show synthesis results for modules used in the connection techniques mentioned. All the presented results show the resource usage of Xilinx Virtex-4 SX35 devices. The proposed NoC uses an 8-bit data bus and shared buses transfer data over 32-bit data links. A shared bus operating at 8-bit data bus after the synthesis utilizes a similar amount of hardware (120LUT's and 11FF's) as a unidirectional NoC interface. LUT's usage in the case of NoC interfaces and routers is caused by the implementation of 32 long input and output queues as shift registers. Such an approach simplifies the placement and routing operation in the final chip, since it does not require any memory block, that has a fixed placement in FPGA. The differentiation between Bidirectional (BD) and Unidirectional (UD) NoC interface, is induced by hardware saving. Namely, not all modules transfer data bidirectionally, some only send, and some only receive data, therefore there is no need of attaching a two-way interface. It is clear that an NoC introduces a cost, that needs to be justified by its capabilities regarding the design or other properties of the implementation.

The first advantage of the NoC cost is the ease of placement and routing of the modules in a physical chip because the place and route tool treats each module separately and hence the design is more flexible than in the case of direct connections or a shared bus. In order to obtain similar effects in a shared bus, a dense segmentation is needed, which was implemented in the presented codec by providing access to the medium to each module with an interface, which is at the same time a transceiver that segments the bus. Similarly, in the case of operating frequency, NoC is highly scalable, as adding new PE to the existing network does not influence the resulting operating frequency. A shared bus show similar properties, if it is densely segmented, as in the proposed codec. Since direct connections are difficult to route in the final chip, the synthesis tool produces long links that are characterize with low operating frequency, as they are a critical path in the design.

Network-on-Chip benefits are also vivid when comparing design capabilities between three connection techniques, as shown in table 4.4. Although with NoC the cost of NoC is a longer design time than in the case of two other communication techniques, it results in simplification and accelerates the debugging process. Firstly, NoC enables the separate debugging of modules based only on the data received. Each module's output is dependent only on the data that are

Table 4.2: A comparison of on-chip connection techniques - hardware implementation

Property	Direct connection	Bus	Network-on-Chip
size of the interface	**Small** – adjusted to the particular communication scheme and data format, it is incorporated into a PE.	**Bigger** – standard interface is a module separated from the PE, that encapsulates and decapsulates data within a bus message.	**Biggest** – standard interface is a module separated from the PE, that encapsulates and decapsulates data within a network packet. Its size depends on the functionality that is implemented.
placing and routing in a physical chip	**Hindered** due to the large number of links between modules some links may be long.	**Better**, however, inside a single segment the routing is hindered due to the links that a few devices have access to.	**Good**, due to segmentation. The placing and routing tool treats each module separately, and is able to flexibly allocate resources on the chip
operating frequency	**Reduced** due to the placing and routing, which can insert long links between modules, that reduce operating frequency significantly	**Depends on the segment size and the outcome of placing and routing.** Large bus segments reduce the operating frequency.	**Does not change** with network size significantly.

Table 4.3: Synthesis results for NoC elements. BD means bidirectional and UD means unidirectional interface

Connection technique	NoC module	number of LUT's	number of FF's
NoC	router	430	330
	NI BD	310	190
	NI UD	110	100
micro-processor bus	transceiver	217	11
memory bus	transceiver	163	85

received over a network. Hence, each module can be checked based on its input.

Secondly, NoC allows for implementing the tools that enable capturing data from a working chip to analyze the correctness of the data sent over a network. The proposed NoC offers a multicast transmission mode that is used to send a single packet to multiple nodes, not only in the case of sending, for instance the same signaling data to different modules, but also allows for capturing of a copy of a packet that is sent over the network. It means that it is possible to analyze on-chip traffic at an off-chip device as in [**?**]. In the proposed NoC it is possible to turn on multicast transmission for any module in the network. Consequently, every packet which is sent over the network from a particular module is copied in the router, and sent to the capturing node, that forwards data to an off-chip device for analysis.

Thirdly, NoC allowed for a separation in the design and debugging of the parser/formatter and predictor, requiring a definition of data format sent between the two. Also, these two parts could be synthesized independently on two chips, which in turn reduced the time needed for synthesis. Single parser/formatter or predictor synthesis took about 45 minutes, whereas synthesizing the whole design took about 1.5 hour on a personal computer. The synthesis is performed repeatedly during debugging, in order to check the correctness of changes introduced. Moreover, debugging could be performed for each part simultaneously. NoC introduction resulted in shorter time of design and synthesis than competitive connection architectures.

Scalability of the system and a small number of connections are other benefits of a NoC-based Systems-on-Chip (SoCs). The first step in building the codec was launching a decoder, then using it as a base application, the goal was to expand it into a codec. The transition from the first to the second step required adding new devices (e.g., motion estimation and forward transform block). Adding those devices did not introduce many changes in the design, since the network protocol was already defined. Also, there was no significant reduction in the network throughput, since there is no bandwidth competition, and new devices send an amount of traffic that does not exceeds the network capacity. Another benefit of the fact that NoC is smaller than in the case of direct connections, is a smaller number of links in the communication infrastructure. The result is easiness in debugging and routing of the final design on a chip.

The only significant cost of the NoC design is the time spent on the design of the protocol, data format and network devices. It is significantly greater than the design time of direct connections and slightly greater than in the case of a shared bus, since it offers greater debugging capabilities. Nevertheless, this time is gained in the debugging process, which is shortened by

an additional functionality of the network. It also shortens the time of adding new modules to SoC, since a new module needs to have a well-defined functionality and input and output data format. Also an NoC is designed once, and then reused in another SoC. From this perspective, direct connections are poorly reusable.

The proposed shared bus architecture was designed to offer similar capabilities as NoC, and hence it is densely segmented. The difference between the proposed bus and NoC is the transmission type. In a shared bus each segment is reserved for a transmission, and in the network data are sent in the form of packets.

Table 4.5 shows synthesis results for the decoder and encoder. The presented size of the parser or formatter and predictor does not include the size of an NoC. Adding the proposed network means increasing the whole design by less than 13%.

Figure 4.3 presents a working system of a codec. The encoding and decoding process is performed on ML-402 [**?**] boards. In order to handle video input and output signals, two Video IO Daughter Cards [**?**] were used. The connection between the boards was realized with the use of the programmer cable.

4.5 Original contribution of the author - summary

The author actively participated in the design and debugging of the networking elements including interfaces and routers. The author designed, implemented and debugged also the following PEs:

- transform coefficients decoder,
- motion vector decoder,
- decoder of coefficients for the deblocking filter,
- inverse transform module with the dequantization submodule,

Also, the author wrote the microprocessor code for decoding motion vectors, as well as transform coefficients. In the design process the author debugged and helped to improve the following modules:

- inverse transform module,
- intraframe prediction module,
- interframe prediction module,
- merger (which adds the prediction result to the inverse transform result) module.

Table 4.4: A comparison of on-chip connection techniques - design

Property	Direct connection	Bus	Network-on-Chip
number of connections	**High** – each communicating pair requires a separate link	**Reduced** – communication proceeds over a single shared link	**Reduced** – communication proceeds over a shared link
protocol	**Adjusted to data format** sent over a link	**All-purpose** – the data are encapsulated within a message. To send a message, the whole segment is reserved for the transmission	**All-purpose** – the data are encapsulated within a packet. The data are sent as packets.
connection speed	**High** – the link is not shared, and its width is adjusted to the bandwidth requirements	**Reduced** – bandwidth competition between devices in a single segment, the data need to be encapsulated into a message sent (it means additional cost of headers and footers), data encapsulation and decapsulation reduces the speed of connection	**Reduced** – reduced competition due to segmentation of the data need to be encapsulated into a packet sent (it means additional cost of headers and footers), data encapsulation and decapsulation reduces the speed of connection.
time spent on the design of a connection	**Fast** – however, each new connection requires the design of a communication protocol and data format	**Reduced** – requires protocol and data format design and a specific interface, however, it is designed once, only data encapsulation and decapsulation is designed when a new module is added	**Reduced** – requires protocol and data format design and a specific interface, however, it is designed once, only data encapsulation and decapsulation is designed when a new module is added
scalability of design	**Small** – Adding, removing or replacing a module requires redesigning of the connection	**Middle** – Adding or removing a module requires only adding or removing the interface, however, it influences bandwidth significantly (in a single segment due to bandwidth competition)	**High** – Changes in the design (adding, removing or replacing a module) do not influence bandwidth significantly due to the minimum bandwidth competition.
debugging capabilities	**Small**	**Middle** – connection oriented debugging, reduced capability to capture messages after implementation	**High** – connection oriented debug, capturing packets after implementation

Table 4.5: Synthesis results of the codec on Virtex4 SX35. The total size of the implementation equals to the sum of Processing Elements size and NoC size.

Part of codec		Processing Elements size		NoC size		NoC share	
Decoder	Parser	10000 LUTs	6000 FFs	870 LUTs	730 FFs	9% of LUTs	12% of FFs
	Predictor	10000 LUTs	9000 FFs	1090 LUTs	930 FFs	11% of LUTs	10% of FFs
Encoder	Formatter	10000 LUTs	6000 FFs	870 LUTs	730 FFs	9% of LUTs	12% of FFs
	Predictor	16000 LUTs	14000 FFs	1510 LUTs	1220 FFs	9% of LUTs	9% of FFs

The modules listed were designed with the use of Verilog Hardware Description Language (HDL) and debugged with the use of a simulation tool, as well as implemented in the final design. The author's active participation in the project resulted in a few conclusions. They are gathered in sections 4.6 and sec:cdc-con, as well in tables 4.2 and 4.4.

4.6 Video codec design problems

The proposed codec architecture contains a few innovatory solutions. The first is the flexible NoC architecture that enabled a multichip design and simplified the debugging process. The placement of the codec on two chips allowed for a separate parser/formatter and predictor design and debugging. The proposed NoC architecture is characterized with an original communication protocol that allows for the construction of small networking devices.

Another innovatory proposal is the fast and simple communication bus to the microprocessor and the memory, which uses the NoC solutions, such as the benefits of a dense segmentation. Moreover, the benefits and limitations of the NoC implementation were gathered in tables 4.4 and 4.2 that summarize the NoC properties. The summary points to the conclusion that although NoC is a cost in terms of hardware consumption, it is beneficial in the design and implementation.

The presented codec is a rather small design, however in the course of the NoC design, no efficient tools were to be found to design an efficient communication infrastructure on a chip. It refers to a simple simulation tool, that would help to examine different network topologies and protocols. Also a pre-simulation topology and/or network protocol assessment tool is needed, to eliminate unnecessary simulation runs. Furthermore, such a tool needs to give similar results

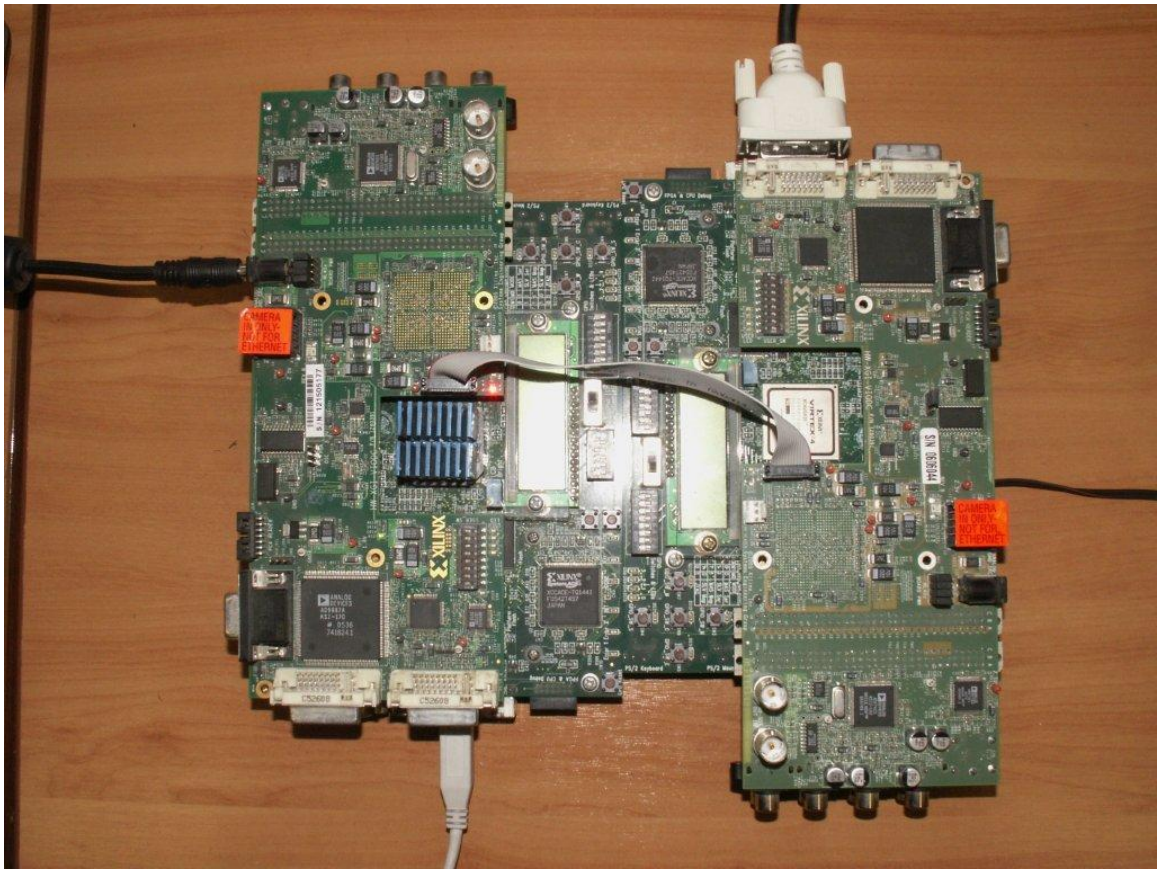


Figure 4.3: Running AVC codec implementation on two ML-402 boards. Each part of the codec runs on the ML-402 motherboard. On top there is Video IO Daughter Card [**?**] for handling video input and output.

to the simulation within the average processing time of a macroblock.

The need for assessment tools grows with the size of a network to implement, because the complicacy of the design rises with the number of nodes. The problem of communication infrastructure for multiview coding techniques on single or multiple chips was signaled in [**?**]. A multiview codec may consist of multiple single-view video codecs and a device that would control the encoding/decoding process. Such a system is much more complicated than the AVC codec presented and is oriented on communication issues between codecs. The presented codec contains 2 routers and 13 PEs. In the case of system presented in [**?**], these numbers need to be multiplied by the number of codecs (i.e., 9) and inter-codec communication infrastructure needs to be added. The proper assessment of communication infrastructure i.e., the protocol and topology for such a system is an open issue.

Another example of the growing complicacy of SoC are SigmaDesigns chips [**?**], that transcode

different types of audio and video formats in a single SoC. Moreover, the communication in Chip MultiProcessors (CMPs)[**?**] systems, exceeds chip borders, which is followed by communication issues involving a low delay on the board links[**?**].

Appropriate tools are needed to estimate the influence of the following network protocol options on the speed of the application implemented:

- delay on different data paths,
- debugging overhead,
- multicast transmission,
- best protocol and topology,
- memory access scheme.

Such an estimation can be done using a simulation which is costly in terms of time spent for calculations. Moreover, the influence of the interconnect parameters mentioned is not straightforward to obtain from the simulation results. However, the knowledge of the share of different communication paths in the whole application performance is very helpful during the design of communication infrastructure.

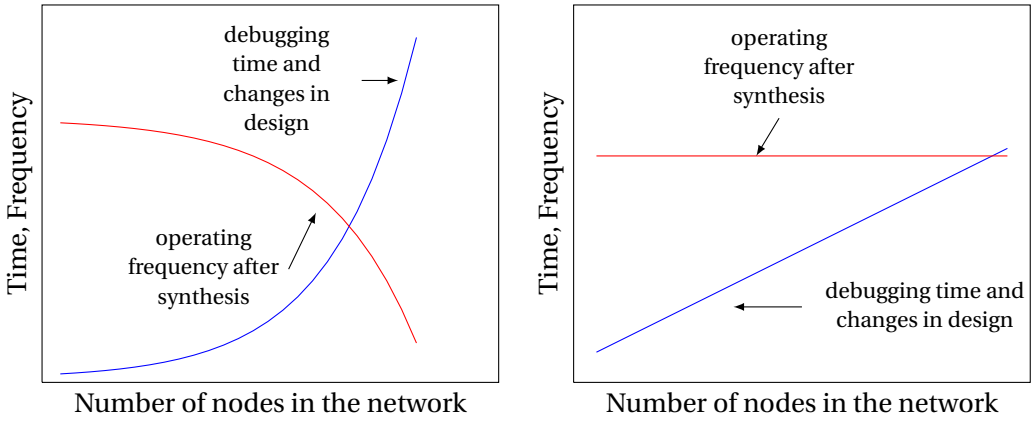
4.7 Conclusions

The codec implementation presented in this chapter shows that the NoC approach is competitive to other connection strategies due to the reduction of design and debugging time. Figure 4.4 presents the unscaled graph of different factors that concern the design and implementation of hardware applications with use of the NoC and non-NoC connection approach. These factors are:

- debugging time,
- time of implementation of changes (e.g., adding or removing modules),
- operating frequency after synthesis.

Figure 4.4 has been drawn based on the author's experience gained during the implementation of the codec. In the case of the non-NoC approaches, such as the shared bus and direct connections, the time spent on debugging and introducing changes is shorter than in the case of the NoC, but only for a small number of nodes in the application. It is related to the high initial cost of the NoC design. However, the greater the number of PEs in the SoC, the smaller the difference gets, and finally, the NoC approach results in a shorter design and debugging time.

The results of the parts of the codec synthesis proved that adding new modules to the structure reduces the operating frequency in the case of non-NoC approaches, especially when implementing peer-to-peer connections. If the design contains a small number of modules, the reduction is not significant. Nevertheless, the increasing number of modules added lowers significantly the operating frequency value. In the case of the NoC the operating frequency is constant, independently of the number of modules. It is caused by the fact that the NoC elements cut off long critical paths in the modules, and these paths are limiters of the codec operating frequency. The synthesis tool is able to create each module separately, hence they do not affect each other.



(a) non-NoC architectures (b) NoC architecture

Figure 4.4: Debugging and design comparison between the NoC and non-NoC connection architectures

The presented results of the codec synthesis (see tabletab:synth-res) show, that the proposed NoC structure requires adding from 9% to 12% of the Look-up tables (LUTs) and Flip-Flops (FFs). Given the design and debugging advantages of the NoC, the hardware cost mentioned is acceptable. The presented NoC proposal can be adopted in any type of application, especially in those containing a small number of nodes.

CHAPTER V

Simulation of Network-on-Chip

5.1 Simulator goals

In order to find the optimal communication infrastructure for hardware implementation of advanced video codecs, an architectural exploration tool is needed, which can be a Hardware Description Language (HDL) or a model of application simulation. The first approach requires the implementation of application modules and communication architecture in hardware. Although such a simulation provides exact results, it is time consuming and each change in communication structure often requires redesigning the hardware. A simulation requires building of a model of application modules and network elements, and identifying the most important characteristics from the communication infrastructure perspective. It means that the application modules are modeled as elements that receive or send data, and calculations are modeled by their approximate time intervals. Network elements, such as network interfaces and routers are described from the high level view, as modules, that forward data of defined amount. Moreover, ports are defined on the socket level, not on the signal level [**?**]. Such a model allows for a general research of communication architecture, which is not restricted to a given proposal. A model may concern different types of communication architectures such as direct connections, a shared bus or Network-on-Chip (NoC). The following work is concentrated on the modeling of advanced video codecs with the use of a network. Such an architecture benefits with scalability, design and debug properties, as described in section 4.4.2.

The purpose of software simulation is to estimate system parameters, i.e. to observe values such as throughput, delay, jitter and communication bottlenecks. The simulator gives results that may be very close to the real performance values, but not the exact values. The purpose of a simulation is to choose the optimal NoC for a specific design. The results obtained this way

are needed for comparison between different NoC architectures and therefore the estimation of parameters is sufficient.

A simulator is needed to estimate system behavior and performance of the hardware video application. The simulation results should include approximate values of processing times in different encoding scenarios of the video codec. In order to simulate a NoC architecture, traffic in a video decoder needs to be modeled, and a simulator that supports that model needs to be found. Since the network architecture is very simple, and needs to be prepared for further research on topology and protocol, a simulator should operate at a low level of traffic description (i.e., sending packets at the network level, not exchanging messages at the application level). It means that the simulator needs to operate on a cycle-by-cycle basis, not on the transaction level. The simulator should also allow for the introduction of changes in the topology and network protocol.

5.2 Review of simulation

In order to choose a suitable simulator for the NoC, a designer needs to consider a few aspects such as the speed of simulator, and the level of results related to its accuracy, flexibility in terms of possible changes in the network architecture, and the ease of writing simulation test cases. The accuracy level describes how detailed results can be obtained from a simulator. There are three levels of simulators' precision that can be obtained: transaction, clock-cycle and combined (partly transaction and clock-cycle accuracy). The choice of precision is influenced by the required speed of the simulator. Transaction-level simulators are the fastest but give coarse results, whereas clock-cycle accuracy level simulators give the most detailed results, but the computation time may be much longer than in the first case, because they run on a cycle-by-cycle basis. The combined accuracy simulator includes parts that are run with the clock-accuracy for detailed results, and parts that operate on the transaction level. Also the speed of the simulator depends on its hardware acceleration e.g., [**?**]. The mentioned features need to be balanced in order to choose the most suitable simulator, that would give results of sufficient detail in a reasonable time.

5.2.1 Accuracy level of simulation results

A simulator may give results with different levels of accuracy: transaction, clock-cycle and a combination of the two. In the first case, the application behavior is described with a high level view, with transactions taking place between the Processing Elements (PEs). Such simulators where proposed in e.g., [**? ? ?**]. The results obtained in this way give basic information about the system performance. However, these are not as accurate as the ones obtained at the clock-cycle level. Nevertheless, such an analysis is faster, and is sufficient for some, especially large systems (e.g. [**?**] and [**?**]).

In order to obtain results that are more accurate, a simulation at the clock-cycle level precision should be performed. The simulator evaluates system behavior on the cycle-by-cycle basis, and gives estimated values of application performance. Such simulators where proposed in e.g., [**? ? ?**].

In the case of large systems, there is a pressure to speed up the computation time. However, in the case where detailed analysis is needed, clock-cycle level accuracy is required. Such circumstances lead to the choice of a simulator of combined accuracy. In [**?**], the authors combine cycle-accurate simulation for NoC observations and transaction-based for the multiprocessor system simulator.

5.2.2 Hardware simulators

Simulation of NoC in hardware is introduced mainly due to its speed. It also allows for emulating the NoC behavior in hardware. The exemplary hardware NoC emulator is presented in [**?**]. Field Programmable Gate Array (FPGA) is used as a test platform for the speeding up of a simulation of a specific design. The proposed network architecture is torus and it is characterized with a few drawbacks: changes in NoC require a redesign of the emulator which is followed by a complete compilation/synthesis of FPGA design, and capacity of the FPGA restricts the size of NoC. In order to tackle those problems, in [**?**], the authors proposed a NoC simulation architecture, called DART, which gives more flexibility due to its parameterization. The routers available use the wormhole routing protocol with virtual channels control. Flow control in this network is credit-based and the topology is configurable. DART consists of a set of tiles (partitions), each contains a few nodes connected to a shared bus. Tiles are connected to each other using a crossbar. The nodes consist of a router and traffic generators which emulate the behavior

of an Intellectual Property core (IP core). Although this approach is more flexible than in [**?**], it still lacks the flexibility in terms of available network protocols and router architecture, which are already defined. Also, the size of the network is limited by the FPGA capacity.

Although hardware simulators are fast and give detailed results, they are destined for testing and debugging of certain implementations, not for general research. A hardware simulator requires some predefined hardware to be put on e.g., FPGA device to do the emulation. This restricts the range of available architectures to simulate, which is a disadvantage in the case of searching for brand new solutions.

5.2.3 Available libraries and simulators

One of the first software simulators for computer networks and parallel and distributed systems is OMNET++ proposed in [**?**]. OMNET++ is C++ based, a discrete-event simulator designed for educational use. It offers built-in parallelism (for reduction of simulation time) and is licensed free, for academic use. It is used in [**?**], where the authors propose a simulator-based NoC on OMNET++ framework. Also in [**?**] the authors propose another NoC simulator, based on OMNET++.

In [**?**] an ×pipes architecture for Chip MultiProcessors (CMPs) was proposed, and in [**?**] where an ×pipes compiler was proposed. The ×pipes compiler uses the ×pipes library, and using a configuration file it generates SystemC description of a network. However, it is destined for CMP and lacks flexibility of architecture design.

Polaris [**?**] is a toolchain for NoC which reflects three design passes: projection of workloads, architecture exploration and circuit and technology projection. It offers an analysis of all architectural issues of a NoC (topology, protocol, flow control as well as physical implementation model), but still contains inflexible and limited traffic modeling for IP cores.

In [**?**], the authors propose DARSIM, which is a flexible, open source, C++ based NoC simulator. Packet sources and sinks can be trace-based or MIPS core simulators. It has parameterized high level network options such as topology, bandwidth and crossbar dimensions. However, the simulator has a predefined router architecture, which is a limitation of its flexibility. Also, the proposed packet injection mechanism is flexible, but it still does not reflect the needs of video applications, since it does not recognize the type of data sent in the network.

In [**?**] a NoC simulator, based on SystemC was proposed. It emulates only mesh topology which significantly restricts its flexibility.

Nigram [**?**] is a NoC simulator written in SystemC for Linux platform. The network is analyzed at the low level that prepares for the NoC proposal for final implementation.

There are also proposals of parameterized NoC models such as Æthereal[**?**] and Proteo [**?**] which provide a parameterized NoC proposal and programming model. However, verification of a NoC implementation is restricted to a single proposal.

A need for new simulators was signaled in [**?**] which announces the "MEMOCODE 2011 Hardware/Software CoDesign Contest: NoC Simulator". This proves that there is still space to introduce a new simulation engine that would answer the problems with the NoC simulation.

The proposals do not provide a flexible simulation model that would allow for defining the traffic type of data transmitted within the package. This is a major factor that restricts the usage of the previously presented tools for video decoder simulation. The problem is discussed in detail in section 5.3. The author proposed a simulator that includes a complicated traffic model of advanced video codecs. Such a model recognizes the different types of macroblocks that differ among each other with the amount and type of data sent, and possibly, a receiving device.

5.3 Proposed NoC simulator

The modeling of hardware application can be discussed on two layers of abstraction: application, and platform. The first one describes tasks that model an application functionality, the second defines a model of platform on which tasks are performed. Such an approach allows for a simulation mechanism separate from the model, and makes it possible to simulate any model of hardware application with the use of the same simulation software. In section 5.3.2, the author proposes an extension of model description, described in [**?**], to improve the accuracy of simulator results, concerning video applications.

[**?**] is the document that specifies the modeling of applications and hardware description, and contain guidelines to build comparable benchmarks for NoC. It is proposed by Open Core Protocol International Partnership (OCP-IP), which is an independent, non-profit semiconductor industry consortium. [**?**] is based on a review of the literature on the NoC, and the modeling of applications.

5.3.1 Model of application

A simulator complies with main instructions of application modeling and XML-file structure according to [**?**]. In [**?**], there is a specified separation of the application, platform and NoC description. Such an approach allows for a simulation of a System-on-Chip (SoC) of any functionality and communication infrastructure architecture. The major difference between the proposed simulator and those described in section 5.2.3 and in [**?**], is the flexible traffic description that allows for the modeling of the amount of data sent over a network with respect to the macroblock type and allowing ordering the destination PEs in the SoC. These modifications are described in section 5.3.2.

Application description contain tasks that communicate with each other. Model of computation described in [**?**] is similar to Kahn Process Network [**?**] where application graph can be built. In such graph vertices represent computation tasks and edges represent communication channels. Tasks are mapped on resources to determine where they are executed. Resources and network model are part of platform which is highly abstracted using characteristic hardware parameters.

Such a model includes a description of the connection structure and processing elements performance distribution. The connection structure contains elements that model on-chip network devices such as routers and network interfaces. The network and processing devices are connected using wires, similarly as in a hardware network. The network elements deliver packets to modeled Processing Elements (mPEs). The routers contain routing tables, which describe which output port to forward a packet to. The model of Network Interface (NI) receives packets destined to the attached mPE and unwraps the data contained from the network headers and passes them to the proper task. Computations inside an mPE are modeled by tasks. The tasks receive data and depending on their amount and type decide what action to undertake.

Each simulated processing element is described only by its interaction with the network i.e., receiving and sending data. It means that none of the calculations inside an mPE (i.e., task) are performed, these are modeled just as a sleep-mode during which the task does perform any interaction with the network. It is treated as a black-box described by following statistical distributions that represent interaction with the network:

- amount of data needed to start calculations,
- duration of data processing,

- amount of data sent.

These distributions are used by each mPE which work according to the scheme presented in figure 5.1.

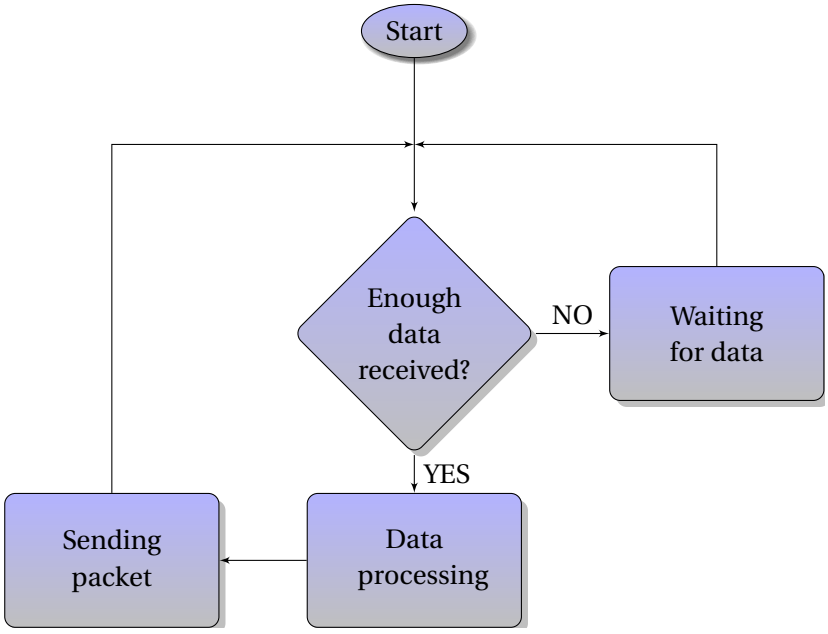


Figure 5.1: The states of modeled Processing Element (mPE) during simulation

At first the mPE checks if there is enough data to start calculations. If not, a mPE waits for more data. If there is enough data, the mPE models the processing of data. During this time the mPE does not receive any data from the network. After completing the calculations, the mPE sends a packet and waits for data to process. The mPE receives data from the network using modelled Network Interface (mNI) which buffers the received and/or sent data and the control logic controls interaction with the network.

The chosen implementation language is C++ due to the convenient and common usage. The proposed simulator was written following the guidelines in the [**?**], and it requires a model of a system, described in an XML input file. Data transfer is modeled with clock-cycle accuracy. However, in the modeled application ports do not simulate a real physical port, only sockets [**?**] which are responsible for receiving or sending packets. Such an approach allows for the analysis of the architectural design of a NoC without the detailed specification of a low-level network protocol.

5.3.2 Proposed modification of application modeling

Simulator proposals presented in section 5.2.3 offer insufficient traffic model which in the case of a video decoder is complex. According to [**?**], each module is modeled, based on:

- the amount of data received,
- the amount of data sent,
- the processing time.

Processing time in calculations is the time, in which a module does not consume or send any data. Each of the variables listed can be described as a statistical distribution [**?**]. The behavior of a such model is described in detail farther in this section, and is illustrated with figure 5.1. Some simulator proposals, such as [**?**], offer a replacement of the distribution with a trace of real values sent. These approaches are insufficient in the case of modeling of video codecs. The first approach proposes too simple a model that does not provide a sufficient level of accuracy, and the second one would require providing data for the whole video sequence that can be as well simulated with the use of a hardware simulation. Moreover, such an approach also does not allow for the provision of a general communication model, but restricted to a specific video content.

In the course of preparing of the model of a video decoder, the author observed that modeling of a video codec depends not on the amount of data (as proposed in [**?**]), but on the *type* of data sent between modules. Another observation was that some modules depend on the conditional probability, in particular the computation time depends on the previously chosen value in the microprocessor. These two observations are further discussed in the following paragraphs.

Type of data sent between modules

The decoder processes data depending on the macroblock type, which defines a range of values for computation time and amount of input and output data, and also determines the modules that participate in the decoding. Namely, the module for intraframe prediction does not participate in the decoding of macroblocks P and B. Also the interframe prediction module is excluded from the decoding of macroblocks I. Such behavior is difficult to model with just a definition of the amount of data sent between modules, especially if the simulation needs to model a system with different macroblock types in a single frame type. Moreover, there are different partitioning schemes for each macroblock type that define the amount of data and processing time of

macroblock in each module. Also, in the case of macroblocks P and B, the segmentation type defines amount of context data that need to be downloaded from the memory. These factors need to be considered and implemented in the model of the codec, indepedenly of its hardware implementation.

Conditional probability of computation time

The processing of a sequence of the same type of data may result in a different processing time for consecutive portions of data. This may occur especially in microprocessors, when the start of such a sequence of data is preceded with the reloading of the buffers for the program and data, whereas the next blocks of data require only the reloading of data memory. Moreover, the reloading of data buffers overlaps data processing, causing further shortening of the computation time. In such a situation a single statistical distribution does not model computation time with sufficient accuracy, and the information of the previously processed data need to be exploited. Given the presented circumstances, the author decided to introduce the possibility of modeling an amount of data and a computation time with the use of conditional probability. Their distribution depends on the number of previously processed blocks of data.

The proposed NoC simulator allows to define statistical distributions, which reflect video decoder characteristics. It allows to describe a module reaction to different data types it can receive. Also, modules computation time and amount of output data can be modeled with conditional probability. Moreover, each module can imitate the same sequence of procedures as in real hardware implementation i.e., a trace. It means that if a module, in the case of macroblock type X, sends data at first to module A, then B and then C, the same sequence is reconstructed in a simulator. Also, if this sequence is changed with the type of macroblock, the simulator is as well able to reproduce such a sequence. Each of the mentioned cases, is characterized with its own statistical distribution. The choice of the currently decoded macroblock type is based on drawing lots from the distribution describing the frequency of each macroblock.

5.4 Methodology of simulator accuracy assessment

In order to check if the proposed NoC simulator gives reliable results for simulated structure, comparative test has been performed. An available hardware implementation of an Advanced Video Coding (AVC) decoder has been modeled and simulated using the proposed NoC simula-

tor.

An assessment was performed for four Motion Pictures Experts Group (MPEG) test sequences (*basket*, *city*, *football*, and *sunflower*, see figure 5.2) of resolution 704×576. These sequence sets give sufficient results on the simulator and decoder model accuracy, since the statistical distributions for other sequences would be gathered according to the same procedure. Moreover, the contents of each sequence differs significantly, and provides a wide spectrum of parameters that are exploited in the decoder. The *basket* sequence contains a lot of moving objects (basketball players and supporters, see figure 5.2(a)), also there is a lot of reflections on the floor which are difficult to compress. Also the *football* sequence contains moving objects (players, see figure 5.2(b)), however there is camera movement and two black strips at the top and at the bottom. *city* is a bird's view of the city centre with camera movement (see figure 5.2(c)) and the *sunflower* presents a bee walking on a big sunflower (see figure 5.2(d)). The compression scheme for each sequence is different, for example *basket* contains many more macroblocks with large density of motion vectors, whereas *football* contains many intraframe predicted macroblocks in interframe predicted frames. Such differences influence largely the timing performance of the hardware decoder and are difficult to simulate just upon rough statistical distributions.

At first, these sequences were encoded with use of all possible encoding macroblock types available in a single frame type as specified in table 3.1. The bitstream was encoded using the UVLC/CAVLC version of entropy codec, and $QP = (15,\ 20,\ \ldots\ 45)$. Then these sequences were decoded with use of AVC reference decoder to obtain statistical distributions regarding number of coefficients and motion vectors in certain macroblock type and partitioning.

Simulated system architecture is, as mentioned, the AVC decoder, described in chapter IV, which has been shown in figure 5.3. Blocks, that are used only in the encoding process are omitted, as well as the block for arithmetic decoding, since, as previously stated, the four sequences were encoded using the UVLC/CAVLC coding. The modeled decoder includes 10 mPEs, that exchange data with each other. The mPEs are connected to the network using mNIs. mNI not only provides connectivity for the mPE, but also models a multiplexer that provides switching capability, as described in section 4.4.2. Similarly, buses were modeled in the parser and in the predictor.

As mentioned, the simulator takes a configuration file as an input, in which statistical distributions and the system description are given. Statistical distributions were gathered for each PE independently. In the case of simulating other applications or sequences, the operation of

(a) basket (b) football

(c) city (d) sunflower

Figure 5.2: MPEG test sequences

gathering information needs to be repeated. Models of PEs are defined based on general statistics that are present in every PE such as the processing time, the amount of data received and/or sent. Also, network elements are defined in a way that is not tied to a specific implementation. The parameters, e.g. buffer size and processing time are commonly used in the description of routers and NIs. Therefore, the accuracy of another system depends not on its architecture, but on the quality of its description.

The goal of the comparison is to obtain accuracy of simulation results, to assess model precision and simulator operation correctness. The simulation case consists of two elements: the simulator and the modeled system description, which are independent. The former supplies the mechanism for simulation of any on-chip system. The latter, describes SoC interconnect architecture and PE interaction with it. The accuracy of simulator results depends on the model, not on the mechanisms of the simulator itself.

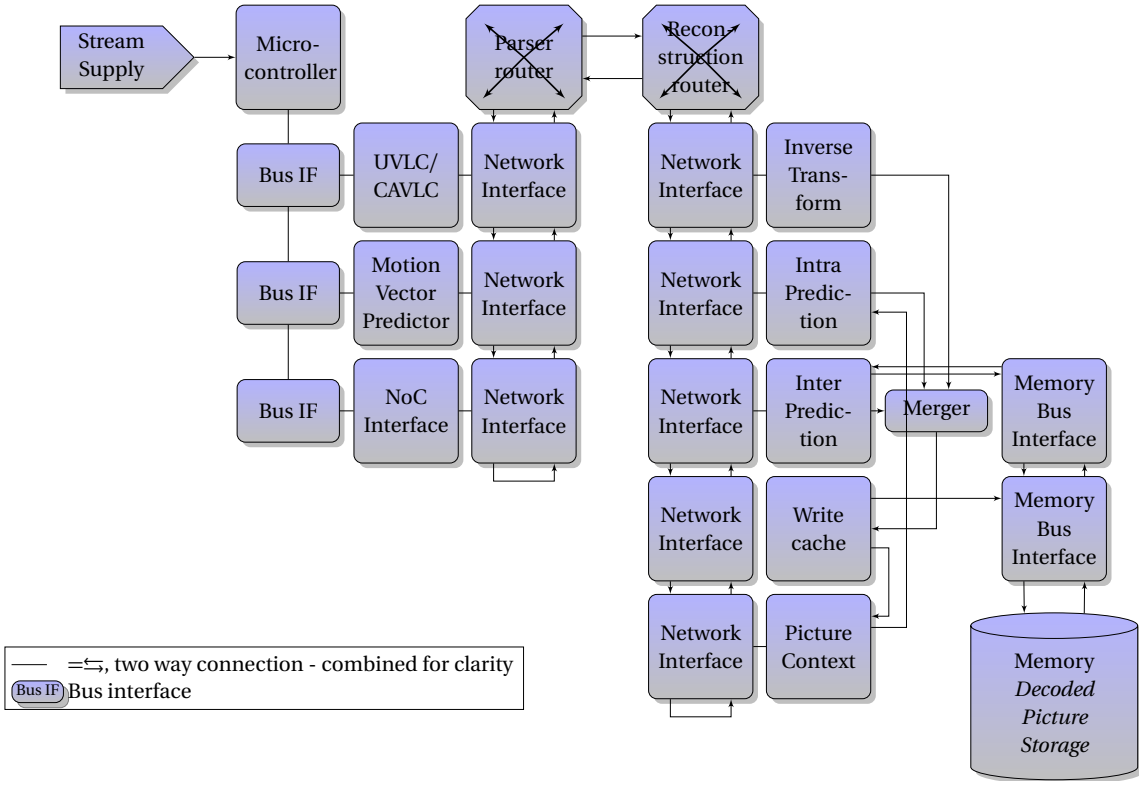


Figure 5.3: Scheme of decoder (extracted from figure 4.1)

The simulation was performed separately on I, P and B frames, and there was no division into slices (one frame contained a single slice), because the simulation needed to estimate system performance in a set-state within a slice. Each frame has different distribution of macroblock types. Also no frame break was simulated i.e. the simulation allowed for system observation during one frame. It is caused by the length of the interval between frames, which is much longer than the processing time of a single macroblock.

The simulator was designed to estimate the average processing time of one macroblock. This value describes how effective the system is. It is expressed in the number of clock cycles, which makes it independent from hardware operating frequency. In fact, the average processing time allows to choose the operating frequency, in order to adjust to system requirements, which is the frequency of video frame at the output.

For each sequence, the average processing time was obtained and compared with a value given by the HDL simulation. The simulation has been repeated 5 times in each case. The author assumed that 10% disparity level is acceptable, considering generalization during modeling, i.e. defining statistical distributions. The detailed results are shown in appendix A, and a description

is provided in section 5.5.

5.5 Results of accuracy of average processing time assessment

Figures 5.4, 5.5 and 5.6 show relative differences, for frames I, P, B, respectively, between average processing time obtained from HDL simulation and its estimation obtained from NoC simulator. Each particular figure presents results gathered for 4 video sequences for comparison.

Figure 5.4 presents the percentage errors for frames I. Curves for *basket*, *city* and *football* follow a similar shape, which is a smaller value at a low Quantization Parameter (QP) value, then rises and peeks for QP = 25 or QP = 30 and falls rapidly for QP = 35 or QP = 40. To the contrary, the curve for *sunflower* has the highest value at the lowest QP, then the discrepancy falls until QP = 40, and for QP = 45 rises slightly. The number of values, for which the discrepancy is higher than 10%, is 8 per 21 measurement points and the highest discrepancy for frames I is below 14%.

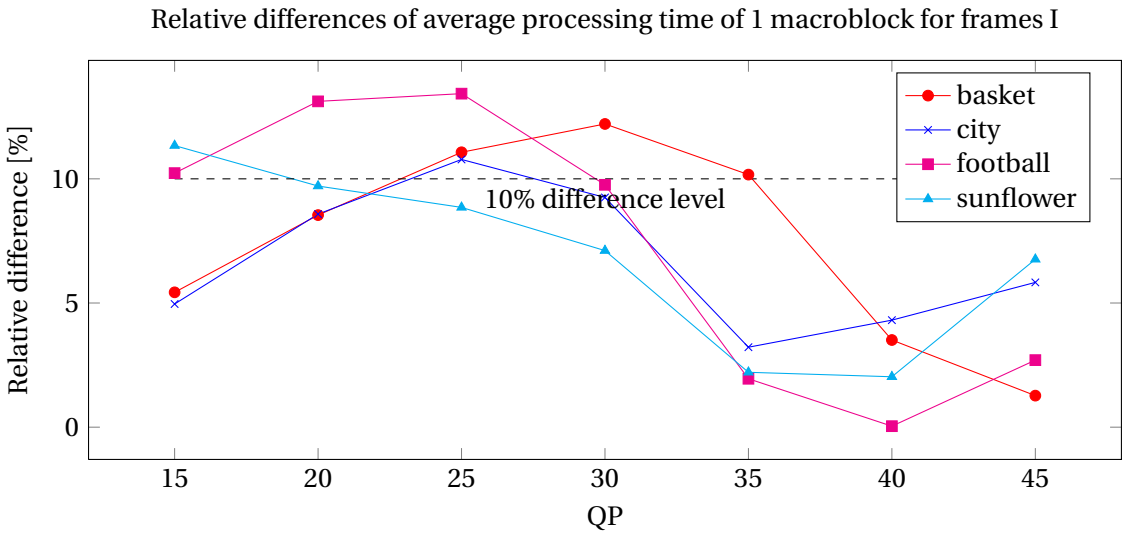


Figure 5.4: Average processing time of one macroblock - simulation accuracy for frames I

In the case of frames P (see figure 5.5), there is a better correspondence between hardware and simulator results. The highest discrepancy is below 12%, and there are only 2 results that exceed the accepted 10% disparity level. The results for all 4 video sequences follow the same pattern, which is the lowest discrepancy for low QP values (i.e., QP $\leq$ 30) and rise for higher QP values. Such a mismatch of the simulator's results suggests problems in the modeling of decoder behavior in the case of lowering of the number of coefficients in the bitstream. A share of coefficient in the encoded bitstream falls due to the quantization process, and other elements

of bitstream gain at expense of the transform coefficients [**?**]. The transition interval between the high and low share of transform coefficients is for QP values in the range of QP $= \langle 25,35 \rangle$. The presented results in [**?**] may differ slightly depending on the Group of Pictures (GOP) used, however, proportions remain the same.

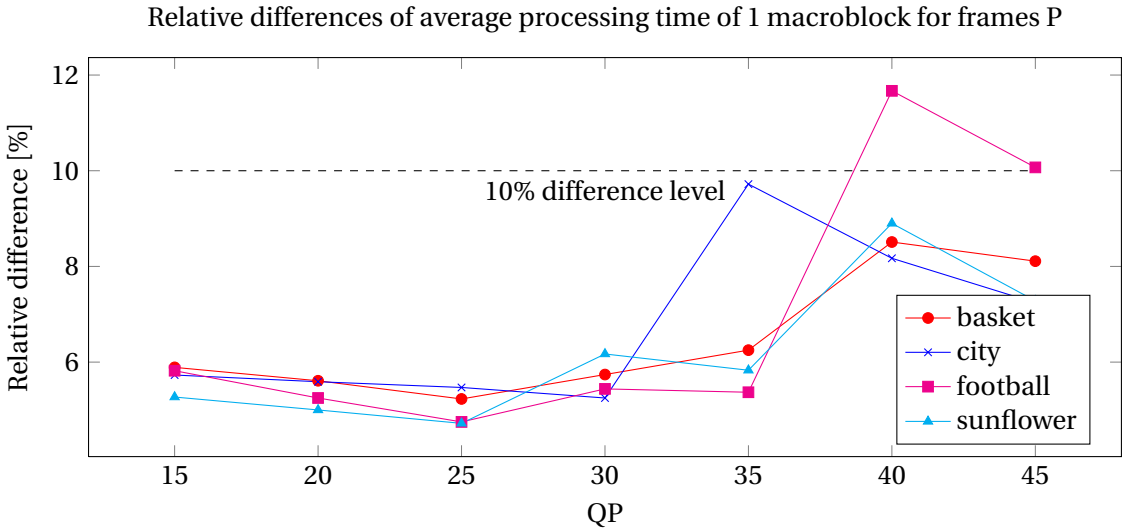


Figure 5.5: Average processing time of one macroblock - simulation accuracy for frames P

Modeling of a system is much more difficult in the case of frames B, which illustrates figure 5.6. The highest discrepancies occur in the case of changing share of transform coefficient and other parts of bitstream [**?**]. It is similar to frames P, nevertheless, the slope of transition is more abrupt, i.e. in the range QP $= \langle 25,35 \rangle$ transform coefficient share falls from 65% to 42% [**?**]. Also the results' discrepancy rises above the accepted 10% level for this range, especially for the *sunflower* sequence. The discrepancy for this sequence equals to 24% at QP $= 30$, nevertheless discrepancies for other sequences do not exceed 14%. Such a difference between video sequences is not only caused by the share of transform coefficients in the bitstream, but also by the content of the sequence.

Since most of the results share same curve shape, the presented discrepancy pattern is independent of the video content. In fact, only in the case of the *sunflower* sequence, there is a significant simulator estimation error for frames B. The *Sunflower* sequence has a very static content which is a bee walking on sunflower. There is no other movement in the sequence. The bee is encoded with the use of macroblocks I, other picture elements are encoded using interframe prediction. Moreover, macroblocks I are concentrated at one frame region. Such encoding characteristics entail the switching of the decoder between two decoding schemes, which is difficult

to model in a simulator, due to the different paths for intra- and inter- predicted macroblocks, and thus for independent decoding.

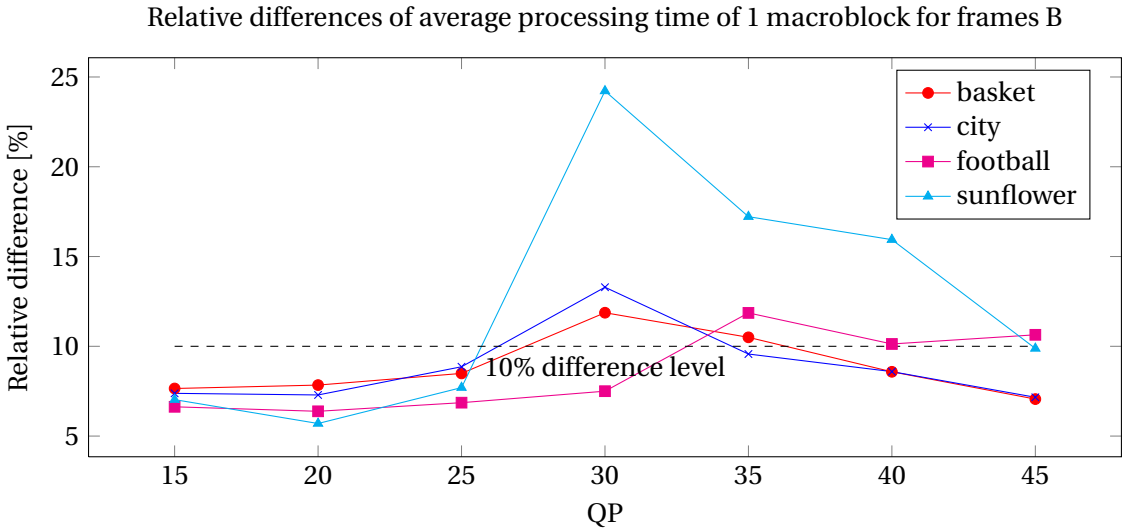


Figure 5.6: Average processing time of one macroblock - simulation accuracy for frames B

The curves for among one frame type often share a similar shape with the maximum discrepancy around one QP value. This points to modeling problems of bitstream with specific properties such as the balance between parameters and the number of transform coefficients. The transition range which is approximately $QP = \langle 25, 35 \rangle$, is difficult to model especially in the case of frames B. In case of frames P difficulties concern higher QP values i.e., $QP \geq 40$, although they are not as big as in the case of frames B.

In order to have more detailed view on acquired results, distribution compliance has been analyzed and discussed in section 5.6.

5.6 Statistical sample correspondence

Estimation of network performance is needed to assess if the proposed network meets the requirements on throughput and other traffic parameters, and hence to answer the question, if the decoder is able to process a video sequence on time. One of the parameters describing system capabilities is the average processing time of one macroblock. The accuracy of estimation of this parameter is described in section 5.5. Nevertheless, a designer needs to obtain information about the minimum and maximum processing values, and more importantly, how frequent they are, to determine if a video frame will be decoded on time. Such information can be obtained

from statistical distribution of simulation results. In order to verify the simulation and hardware distribution compliance, results have been compared with the use of Kruskal-Wallis test. The outcomes are presented in table 5.1.

Kruskal-Wallis test

Kruskal-Wallis test requires the ascending ordering of combined samples, which allows for their rank determination [**?**], i.e. assigning a value to the observation which denotes its order. Then H statistic is computed with use of the following formula if there are no ties (i.e., if no two observations are equal):

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{C} \frac{R_i^2}{n_i} - 3(N+1) \tag{5.1}$$

where C is the number of samples, n_i is the number of observations in the i-th sample, N is the number of observations in all samples combined (i.e., $\sum_{i=1}^{C} n_i$), and R_i is the number of the ranks in the i-th sample [**?**]. If there are ties, each observation is divided by:

$$1 - \frac{\sum T}{N^3 - N} \tag{5.2}$$

where the sum is performed over all groups of ties and $T = t(t-1)(t+1) = t^3 - t$ for each group of ties, t is the number of tied observations in the group [**?**]. The result of computations is H statistics, which is used to compute corresponding p-value, which is approximated by $Pr\left\{\chi^2_{C-1} \geq H\right\}$. χ^2_{C-1} is chi-squared distribution with $C-1$ degrees of freedom. Large H values lead to rejection of null hypothesis.

Kruskal-Wallis test results

The purpose of the test was to compare statistical distributions of the simulation and hardware macroblock processing time. In each case the output (i.e., the processing time of consecutive macroblocks) was used to calculate the probability of occurrence of each macroblock time. A Kruskal-Wallis test on distribution compliance [**?**] has been performed assuming that the significance level is $\alpha = 0.05$, which is treated as a "border-line acceptable" error level [**?**]. The results have been gathered in table 5.1. Since the p-value needs to be greater than the significance level ($\mathrm{p-value} > \alpha$) to accept the hypothesis H_0 that both random sequences have been drawn from the same distribution. Only in the case of *football*, for frames P (QP = 15 and 45)

and B, hypothesis H_0 cannot be accepted. *Football* sequence has a significantly different content from other test sequences which is two black stripes at top and at the bottom of the image. Macroblocks placed in that area, are encoded very effectively, especially in the case of interframe prediction and are decoded very quickly, especially if placed one next to the other. The simulator is not able to reconstruct the original order of macroblocks, and thus the estimated decoding time differs significantly from the original one. Macroblocks decoded according to the real order, are decoded faster than in the case when they are mixed with other macroblock types. This difference has a significant influence on the minimum and maximum processing time. Also it affects the probability density function, as shown in figure 5.7.

In order to estimate how significant maximum processing time is in course of assessing network performance (i.e. is it worth to scale network to maximum processing time - worst case scenario), comparison of two distributions has been drawn: *football* sequence, frame P, QP = 15 and QP = 30. In the first case, there is worst distribution compliance (p-value = 0.012). In contrast, in the case of QP = 30, distributions are compliant.

Figure 5.7(a) shows histogram of macroblock processing time as it is in hardware decoder (*football* sequence frame P and QP = 15). Figure 5.7(b) illustrates exemplary simulation realization.

Although the null hypothesis H_0 (that two random sequences were drawn from the same statistical distribution) needs to be rejected in the case of the *football* sequence, frame P, QP = 15 (see table 5.1), figure 5.7 and 5.7(b) show similar properties, i.e. cumulative density function saturates most for the low macroblock processing time values. It means that macroblocks that are processed long occur very rarely, and the scaling of a network can be based mostly on the average processing time.

Figure 5.8(a) shows a histogram of macroblock processing time as it is in the hardware decoder (*football* sequence frame P and QP = 30). Figure 5.8(b) illustrates an exemplary simulation realization.

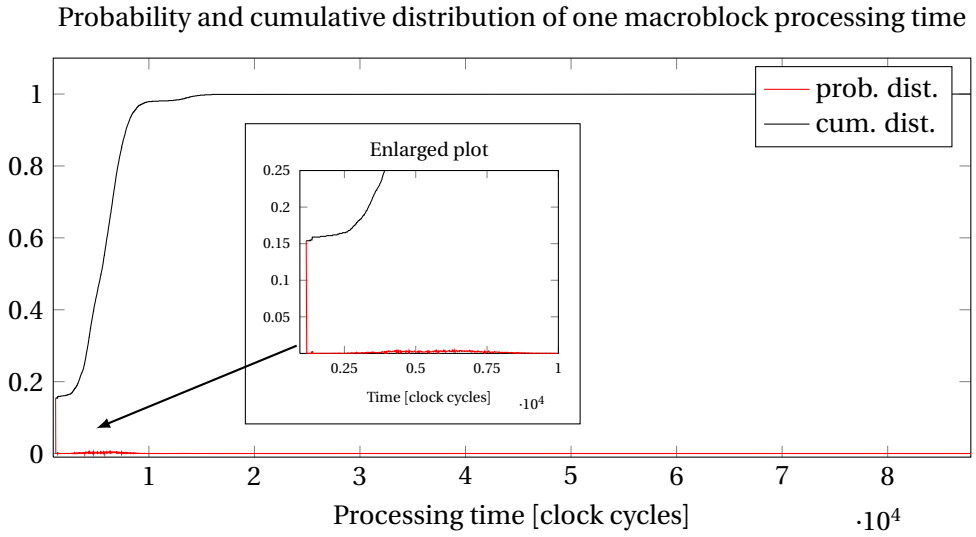
Figure 5.8 present two random sequences that can be regarded as drawn from the same distribution (see table 5.1). In both cases cumulative density function saturates mostly for the lower processing time values, which means that long processed macroblocks occur very rarely. The simulation gives similar results as the hardware decoder.

Table 5.1: Kruskal-Wallis test results (p-values) for distribution compliance

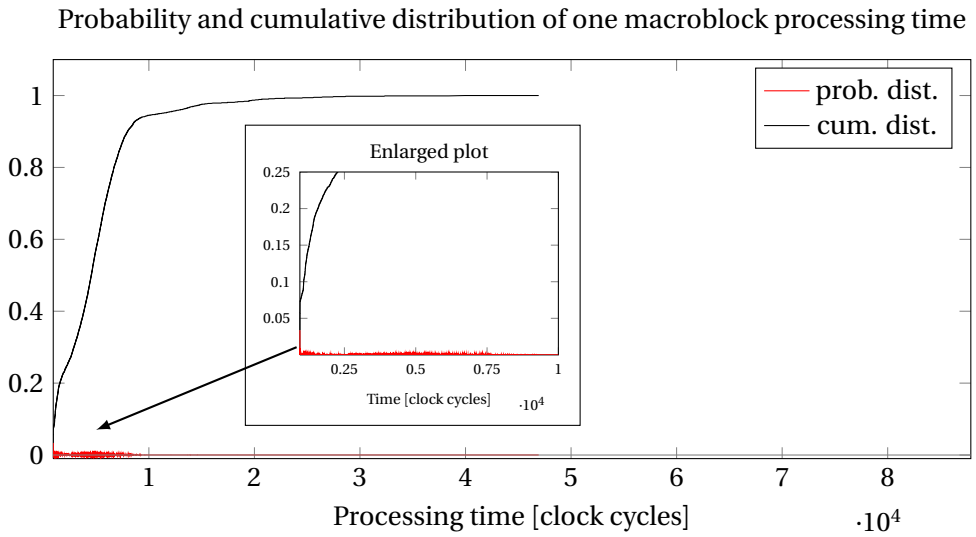
Frame	QP	*basket*	*football*	*city*	*sunflower*
I	15	0.394	0.732	0.709	0.938
I	20	0.303	0.961	0.571	0.749
I	25	0.641	0.915	0.899	0.778
I	30	0.823	0.815	0.704	0.711
I	35	0.93	0.647	0.876	0.245
I	40	0.732	0.308	0.746	0.185
I	45	0.464	0.224	0.93	0.4
P	15	0.516	0.012	0.748	0.946
P	20	0.977	0.067	0.907	0.877
P	25	0.961	0.148	1.0	0.641
P	30	0.719	0.222	0.315	0.586
P	35	0.627	0.274	0.679	0.292
P	40	0.738	0.343	0.512	0.784
P	45	0.769	0.031	0.443	0.453
B	15	0.807	0.001	0.055	0.108
B	20	0.748	0.001	0.631	0.543
B	25	0.961	0.016	0.567	0.549
B	30	0.808	0.034	0.496	0.838
B	35	0.555	0.061	0.34	0.79
B	40	0.5	0.081	0.608	0.438
B	45	0.369	0.016	0.095	0.702

5.6.1 Discussion

The purpose of the simulation is to estimate overall system performance. A metric that describes system capabilities is the average macroblock processing time. The proposed simulator gives results with a 10% accuracy level, with a few exceptions, which are described in section 5.5. Nevertheless, a designer should obtain information about the minimums and maximums



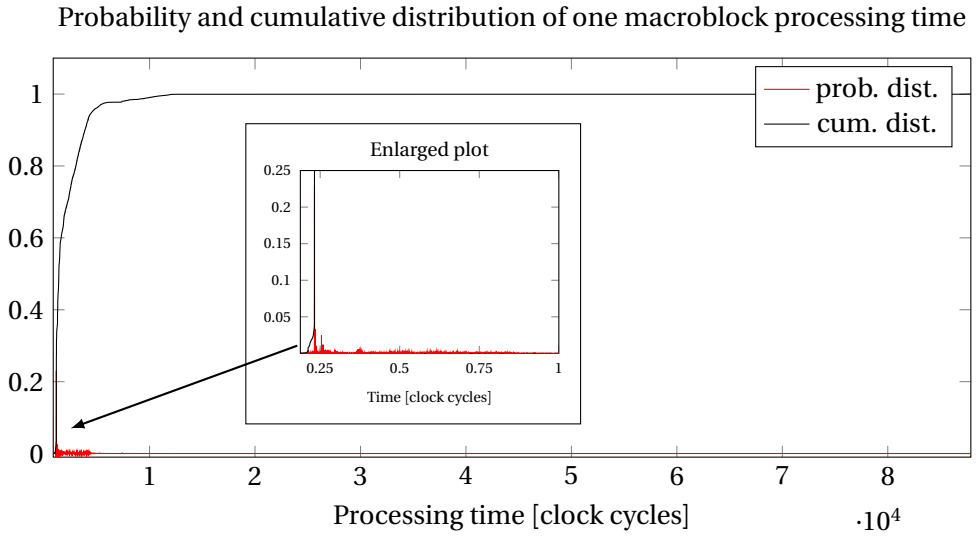
(a) hardware decoder



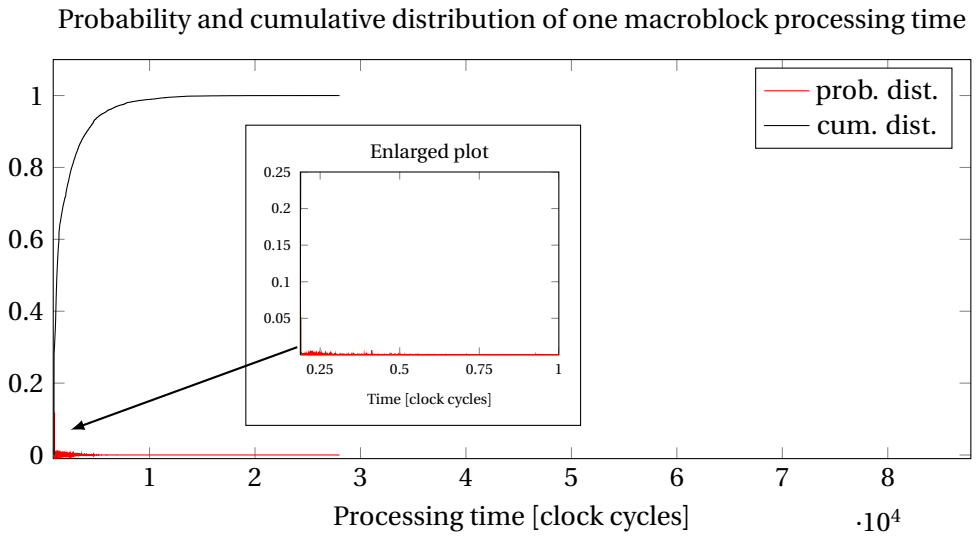
(b) simulation

Figure 5.7: Probability distribution (prob. dist.) and cumulative distribution (cum. dist.) functions of macroblock processing time (*football* sequence, frame P, QP=15)

with the emphasis on their frequency. Such information can be obtained from the analysis of macroblock processing time distribution. Section 5.6 presents accuracy of distribution obtained from the simulation. Although in a few cases, the hypothesis H_0 about distribution compliance has to be rejected, the simulation distributions show the same regularity as the hardware results. The most frequent are the shortly processed macroblocks, and long processed macroblocks are very rare. In order to calculate precisely how much above the mean the cumulative density function saturates up to the probability of 0.9, the quantile function [**?**] was calculated. Assuming



(a) hardware decoder



(b) simulation

Figure 5.8: Probability distribution (prob. dist.) and cumulative distribution (cum. dist.) functions of macroblock processing time (*football* sequence, frame P, QP=30)

that there is a random variable X with cumulative density function, $F(x) = Pr\{X \leq x\} = p$, the quantile function is defined as follows:

$$Q(p) = x \tag{5.3}$$

Quantile function is the inverse of cumulative density function: $Q(p) = F^{-1}(p)$. In order to find the relation between $x_{0.9}$, such that $Q(0.9) = x_{0.9}$, and distribution mean, the following values

have been calculated:

$$m_{\%} = \frac{\text{abs}(mean - x_{0.9}) \cdot 100\%}{mean}. \tag{5.4}$$

Table 5.2 gathers $m_{\%}$ values for four sequences, each calculated for frames I, P and B and QP = 15,20...45. In most of the cases cumulative density function saturates at 90% level below the value: $mean \cdot 1.5$. In a few cases the saturation occurs close to the $mean \cdot 2$ especially in the case of QP = 20,25 and 30. In the case of QP = 40 and 45 for frame P and B, $m_{\%}$ drops to a few percent, however it depends mainly on the video sequence content and its compression rate. The presented analysis shows a fast saturation of cumulative density function referenced to the mean value. The maximum processing time measured equals at least 10 times the mean value. Therefore the estimation of the maximum processing time is not very useful in the estimation of SoC performance.

The simulator was designed to estimate the average processing time of a macroblock, however information can be retrieved from its results about the minimum processing time required to decode one macroblock. The difference between hardware and simulation results equals up to 250 clock-cycles, which corresponds to the processing time of the inverse transform module. In most of the cases the simulator gives constant value that is higher than the result obtained from the hardware decoder. It means that the simulator tends to give more prudent results about the best decoder performance.

The analysis of the macroblock-processing-time distribution gives information about system behavior, and helps to select hardware operating frequency in the targeting platform, further discussed in section 5.7. The distribution gives not only a minimum, maximum and mean of macroblock processing time, but also helps to estimate how rare the extreme values are. The proposed simulator gives macroblock distribution time compliant to the hardware results in most of the cases. Moreover, even though the sequence content is encoded in such a way that the ordering of macroblocks influences the decoder performance (such as *football* sequence), the distribution obtained from the simulator results still reflects the real hardware behavior as shown in figure 5.7.

Table 5.2: $m_{\%}$ values which indicate how much the mean need to be increased to obtain $Pr\{X \leq x_{0.9}\} = 0.9$. Based on hardware results.

Frame	QP	*basket* $m_{\%}$[%]	*football* $m_{\%}$[%]	*city* $m_{\%}$[%]	*sunflower* $m_{\%}$[%]
I	15	26	18	39	25
I	20	33	25	45	26
I	25	35	25	48	29
I	30	37	23	56	42
I	35	40	24	64	67
I	40	42	35	77	90
I	45	54	54	74	87
P	15	55	25	50	69
P	20	75	34	72	102
P	25	85	58	85	91
P	30	89	48	84	46
P	35	59	7.1	67	2.3
P	40	16	1.9	15	1.8
P	45	1.7	0.1	3.3	2.1
B	15	51	26	42	65
B	20	74	34	51	98
B	25	93	48	63	93
B	30	93	60	70	47
B	35	63	11	70	11
B	40	17	1.8	59	0.5
B	45	0.7	0.8	1.6	2.1

5.7 Selection of hardware operating frequency

The simulation results presented in section 5.5 can be used not only for comparison between different interconnect architecture capabilities, but also for the calculation of hardware

operating frequency. This frequency needs to provide decoder with the ability to decode a video sequence in real time, which is 25 frames per second in the case of television systems. A detailed analysis with respect to frame types and QP values shows system performance in different decoding schemes and helps identify the paths that introduce the largest delay. The operating frequency is related to power consumption, as it describes the switching activity of an application [**? ?**].

The operating frequency is calculated depending on the system type which is a low and high delay. In both systems, output frame frequency is the same, however, the high delay system allows for a delay in the decoding of a frame which requires long calculations. In this case other frames need to be processed faster than required, in order to save time for the next long-processed frame. The allowed delay time depends on two parameters: the decoded frame buffer size and the GOP structure. A large buffer allows for storing more frames, and thus for a higher delay. Also the GOP structure shows how many long and short frames are there in the stream and what are the proportions between them which influence the length of the allowed delay. In small delay systems all the frame types need to be processed at least in real time (in the case of a faster decoding rate, the frames are stored), which means that the operating frequency is adjusted to the longest processing time frame type.

A hardware simulation was performed for 9000 macroblocks, and based on them, the operating frequency has been calculated. The estimated operating frequency (based on simulation results) has been calculated using the 5.5 formula. It refers an average number of clock cycles spent on decoding one frame to the frame rate that needs to be obtained at the output of the decoder:

$$f = t_{MB} \cdot N_{MB\_in\_frame} \cdot f_{frame\_rate}, \tag{5.5}$$

where

t_{MB} – average processing time of a single macroblock - expressed as a number of clock cycles,

$N_{MB\_in\_frame}$ – number of macroblocks in a single frame,

f_{frame_rate} – number of frames per second.

At first, in order to get results for a small-delay system, the operating frequency was calculated from real hardware results (section 5.7.1), and then compared to the simulation results in section 5.7.2. Finally, the operating frequency for a high delay system was calculated, with re-

spect to the assumed GOP and frame buffer size 5.7.3.

5.7.1 Frequency of design based on hardware performance

Hardware simulation has been performed for 9000 macroblocks, and based on these results, the operating frequency needed has been calculated with its confidence interval (see detailed results in table B.1). Figures 5.9, 5.10 and 5.11 present the graphical form of those results, for frames I, P and B, respectively. Each particular figure contains the calculated frequencies and their confidence intervals for four sequences: *basket*, *city*, *football* and *sunflower*.

Figure 5.9 presents the required minimum frequency to decode frames I in real time. This value decreases with the growth of the QP, as the frame processing time diminishes. This decline is steady throughout the QP range. Moreover, confidence intervals do not exceed 10 MHz, and are greater in the case of low QP values (i.e., $QP = 15, 20, 25$) than in the case of high QP ($QP \leq 35$). This fact suggests, that differences in the average macroblock processing time are caused by the decoding path of macroblock coefficients, whose share in the AVC stream declines with the increase of the QP value [**?**].

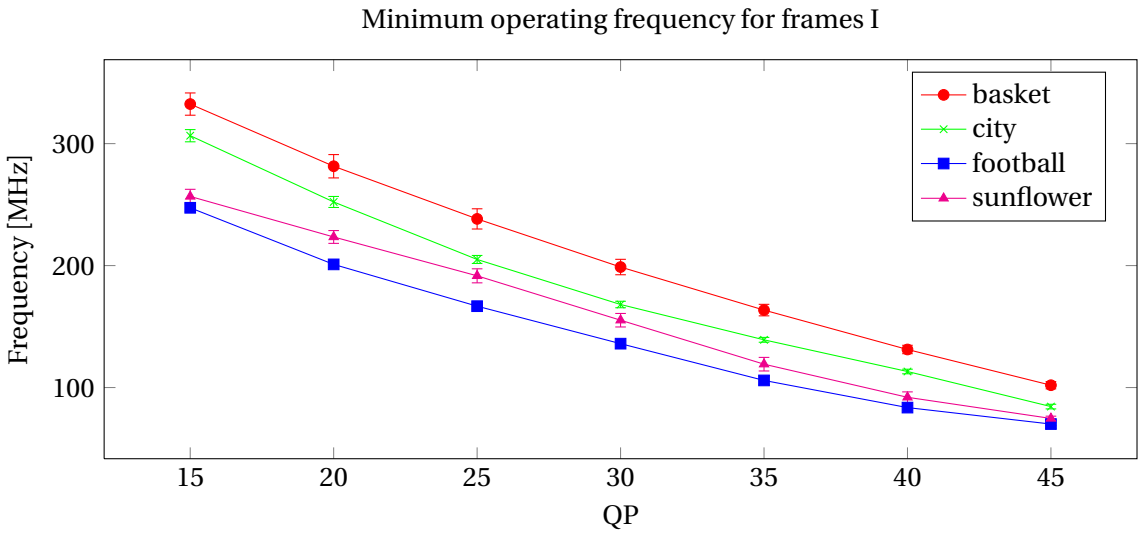


Figure 5.9: Operating frequency for frames I

In the case of decoding of P frames, the highest operating frequency is by about 76 MHz lower than the highest frequency for I frames (as presented in figure 5.10, for details, see table B.1). Confidence intervals are greater, when compared to results gathered for I frames. For P frames, the confidence intervals are below 18 MHz. These values are decreasing with the growth of QP. It could be caused by decreasing share bits related to transform coefficients in the AVC

stream.

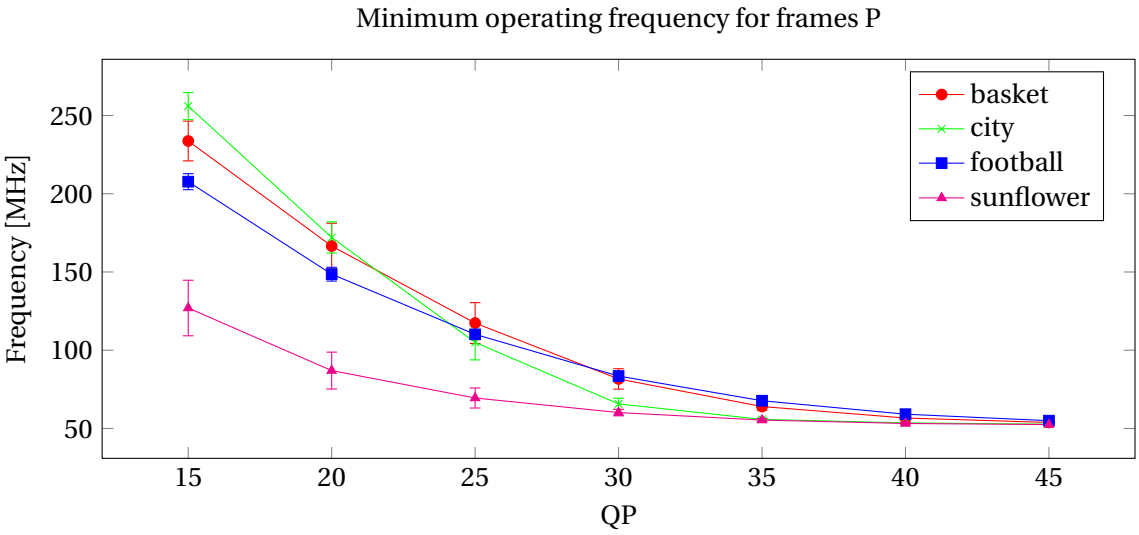


Figure 5.10: Operating frequency for frames P

The operating frequency for frames B is higher (about 14 MHz in the case of two highest values) than in the case of frames P. This could be caused by the increased complexity of bidirectional interframe prediction and a higher number of memory reads, in order to get the context for a macroblock. The results are presented in figure 5.11, and detailed results are gathered in table B.1.

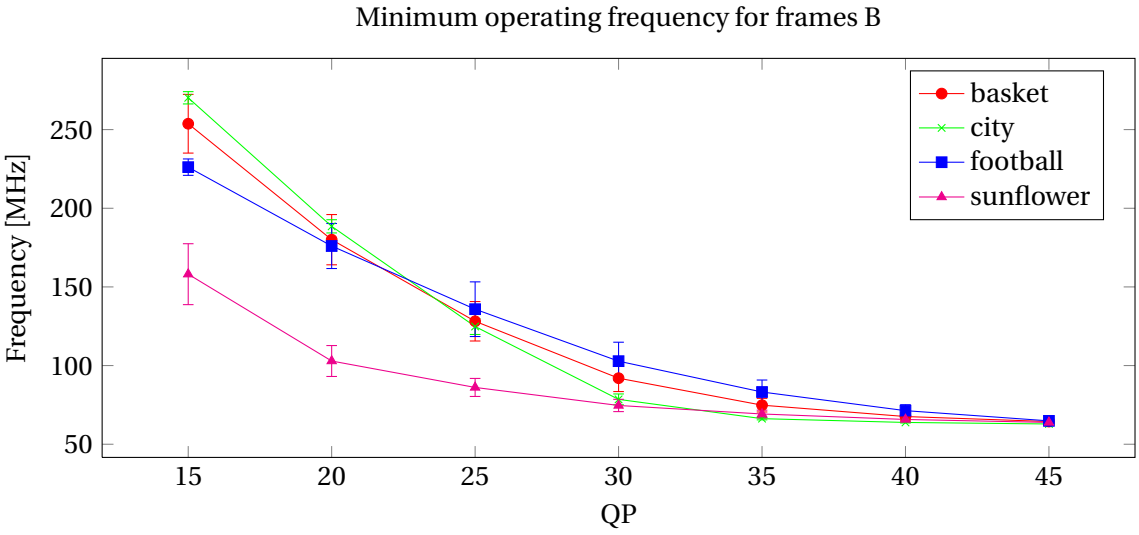


Figure 5.11: Operating frequency for frames B

The presented results show that the operating frequency should be chosen based on the performance of frames I, which are decoded in the longest time. Also, the frequency for the lowest

QP value should be taken as the base for operating frequency, since it is the highest of all. It means that the presented system should have the operating frequency of at least 332 MHz. However, the scaling of operating frequency to the processing time of frames I is not efficient, since frames I are placed only at the beginning of a GOP, and then frames P or B are chosen, because they offer a higher compression efficiency. The results that include the size GOP structure and buffers are presented in section 5.7.3.

The presented analysis is helpful, however, it is not sufficient. One of its weaknesses is little information about which module introduces the greatest delay, only a path of macroblocks I is indicated as the source of potential long delays. Nevertheless, this path contains several modules, and each introduces some delay. With the analysis presented, there is no precise indication of the module which causes the greatest delay. This suggests a necessity for a tool of more detailed analysis of the data path, even before the simulation.

5.7.2 Frequency of design based on simulator results

In some cases there is a need to roughly estimate the operating frequency, upon simulation results to assess system performance, especially in the energy-consumption-sensitive (e.g., handheld) devices. Such an analysis should also provide information on the identification of modules that process data longer than others in the overall data flow. The detailed results of the system operating frequency are gathered in table B.2. In figures 5.12, 5.13 and 5.14, the absolute errors of that estimation are presented. Zero value means that there is no difference between the simulation and the hardware results. The values above zero mean that the simulation overestimates the operating frequency, and below indicate that the simulation underestimated the operating frequency.

In the case of frames I the simulator overestimates the results for low QP values ($QP = 15, 20, 25$) the most and for $QP = 15$ is about 60 MHz. With the growth of QP this difference falls to 10 MHz at $QP = 45$. A higher discrepancy in the case of lower QP values is related to the higher variability of the processing time which is illustrated in figure 5.9 with broader confidence intervals. It also suggests modeling problems in the case of a high rate of transform coefficient in the AVC stream.

Similarly, in the case of frames P, the operating frequency is mostly overestimated, however, with about three times lower error. There is also one case in which the simulation slightly underestimates the operating frequency (*football*, $QP = 40$).

The absolute estimation error in the case of frames B is lower than with frames P, even in the

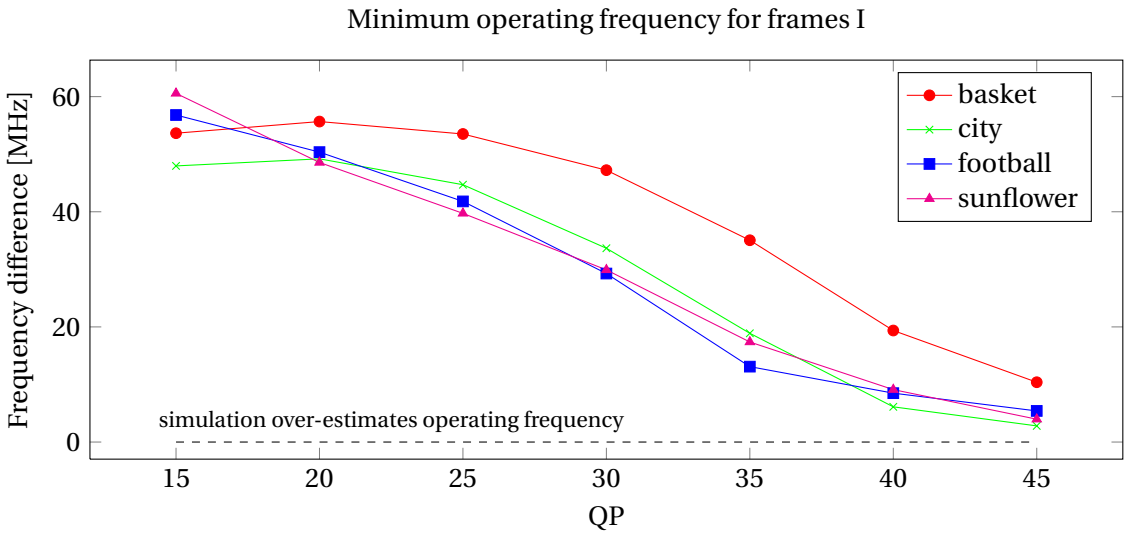


Figure 5.12: The operating frequency computed from the hardware and compared to the simulation results for frames I

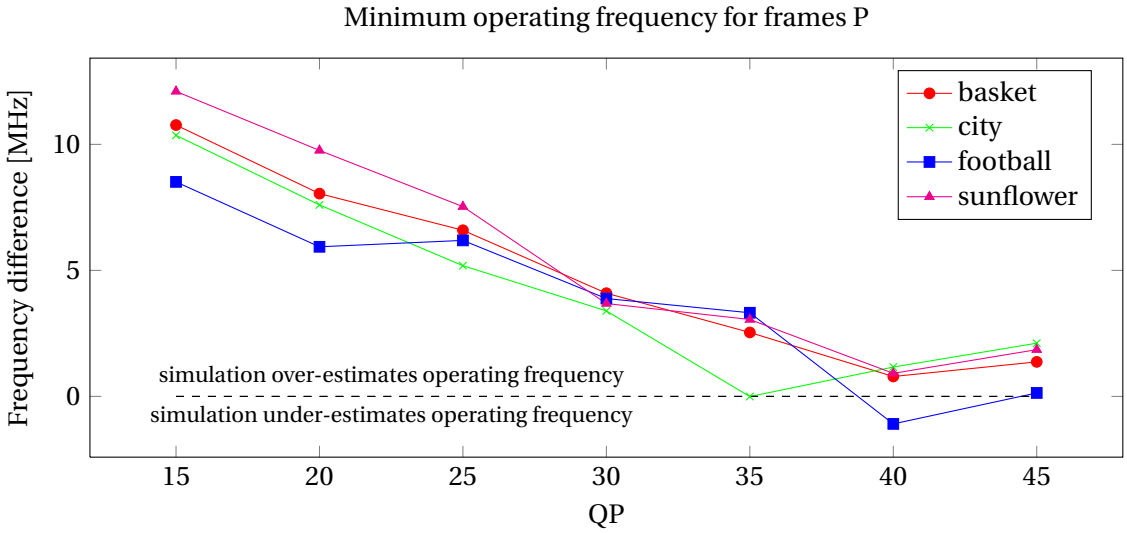


Figure 5.13: The operating frequency computed from hardware and compared to the simulation results for frames P

case of *sunflower* sequence, at QP = 30, 35, 40. This underestimation is related to the high relative difference of the average processing time of one macroblock.

The resented results suggest difficulties in the modeling of system behavior in the case of high rate of transform coefficients in the AVC stream, namely the absolute error value falls with the growth of the QP value, as the rate of transform coefficients [**?**]. Also the better the compression efficiency when comparing frames I, P, B, the lower the estimation error.

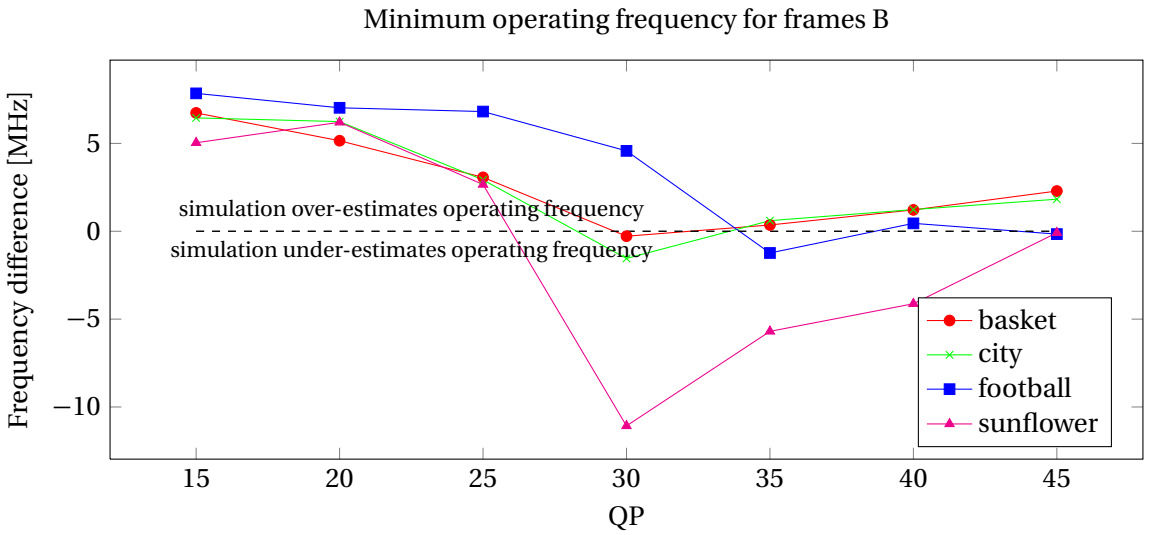


Figure 5.14: The operating frequency computed from hardware and compared to the simulation results for frames B

5.7.3 Operating frequency with respect to theGroup of Pictures and frame buffer size

Systems that allow for a delay are able to decode frames at a lower frequency, than the low delay systems, due to the buffering of decoded frames. The resulting operating frequency allows for the decoding of frames I above the real-time decoding, however, other frames i.e., P and B, are decoded faster than their required time. All the frames are buffered, so the display rate is constant at 25 Hz, but the operating frequency is weighted between the frequencies required by the real time decoding for frames I, P and B. The AVC standard defines the size of the decoded picture buffer for SDTV video resolution to 5 frames. Also, television systems require that GOP should not exceed 0.5 second [**?**], which means that with a 25Hz frame rate, I frame should appear at least once per 12 pictures. Other frames can be P or B frames for their better compression efficiency. Given that, the GOP contains a single I frame and 11 P or B frames. In order to obtain the best compression efficiency, frames B are more frequent than frames P. Since the buffer includes 5 full frames, and frames I and B decode longer (see section 5.7.1) than frames P, and furthermore, frames B are more frequent than other frames. The worst case scenario is to decode and display in real time a sequence of the 5 following frames: I, P, B, B, B.

In order to obtain the minimum operating frequency in such a system, the results presented in section 5.7.1 were used to calculate the result based on hardware performance. For comparison between a real system and its model, the simulation results presented in section 5.7.1 were used to perform similar calculations. In the end, both results were subtracted, to obtain the

absolute difference value.

Figure 5.15 presents results for the frequency based on hardware results and figure 5.16 shows the difference between the frequency obtained on the basis of the simulation and the hardware results i.e., the approximation error. The detailed results are gathered in tables B.3 and B.4 for hardware and simulation results respectively.

The resulting operating frequency is lower than the frequency for frames I and higher than in the case of frames P and B (see table B.3). The required operating speed falls with the growth of the QP values, as shown in the figure 5.15. Nevertheless, in order to provide the sufficient operating frequency for the most frequently usable QP values, the operating frequency for this system should be at level of 275MHz. Comparing this value to the low delay system results (see section 5.7.1), it is about 58 MHz lower.

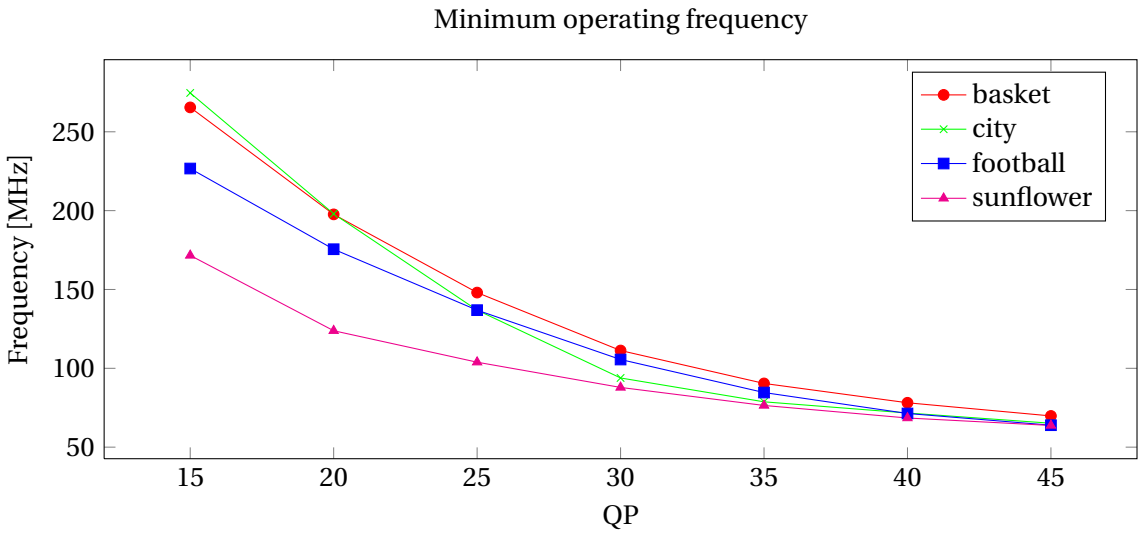


Figure 5.15: Operating frequency in the case of storing in output decoding buffer frames: I, P, B, B, B - based on hardware results

The presented results can be estimated with use of simulation results, and the absolute estimation error has been presented in figure 5.16. The highest estimation error is below 16.5 MHz and falls with the growth of QP.

5.8 Conclusions

The simulator proposed gives results for an average macroblock processing time with a 10% accuracy. Single cases of higher relative differences are caused by a sequence of macroblock types which is not modeled in a simulator. Another cause of discrepancies is the difficulty in

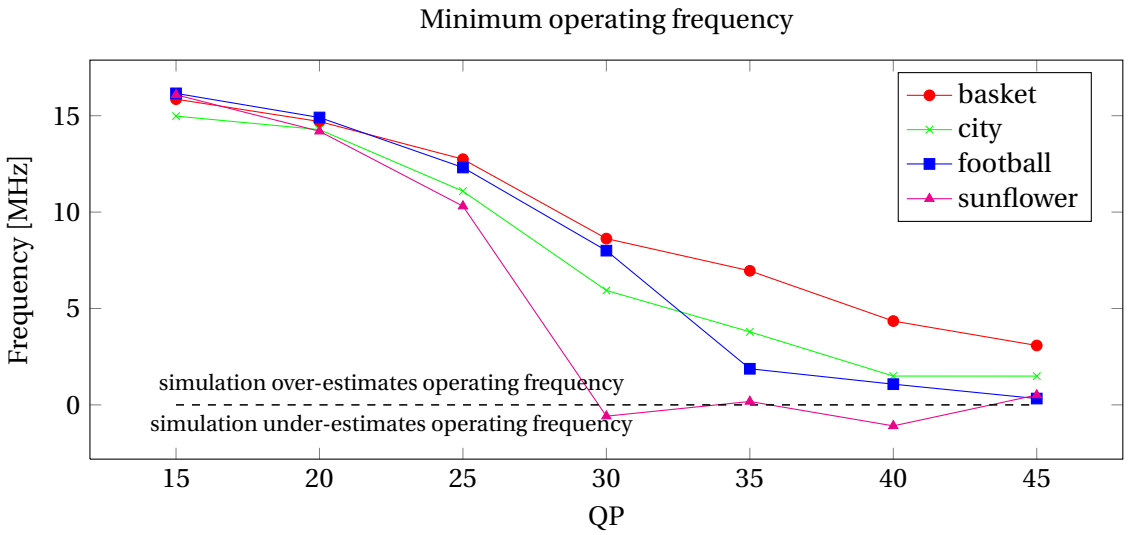


Figure 5.16: Estimation error of operating frequency in the case of storing in output decoding buffer frames: I, P, B, B, B

modeling of the decoding of a stream with high transform coefficient number per macroblock. In this case the performance of hardware decoder varies more than in the cases of low coefficient rate. Nevertheless, the obtained results allow for the estimation of the minimum operating frequency of hardware application.

CHAPTER VI

Queue model for advanced video codecs

6.1 Queue modeling

The main purpose of queue system modeling is to give a simple description to the stochastic processes that occur in the real queuing systems [**?**]. In [**?**] the authors show the applicability of Erlang delay formula to the Internet. The Internet is a packet network, and Erlang delay formula describes the processes observed in a telephone network, however it is still applicable to packet networks. The relevant parameters of this model are: flow rate, link capacity and overall demand [**?**].

A queue model (see figure 6.1) contains a queue that receives an input stream, i.e. calls one or more servers that process a call taken from the queue, and put it to the output channel. If the channel is busy, the calls are waiting in the queue to be served [**?**]. Basic queue models assume a First-In First-Out (FIFO) queue type [**?**]. Depending on the model type, the mentioned parameters may vary. The input stream may be described e.g., by Poisson distribution. Queue length may be finite or infinite, and there can be one or more servers.

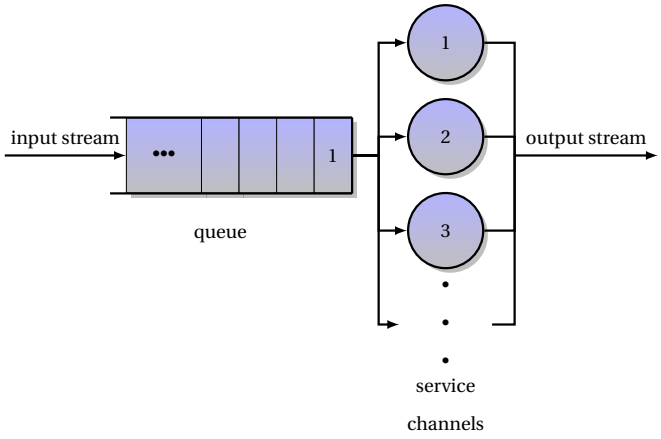


Figure 6.1: Queue system [**?**]

In order to describe the queue model parameters briefly, Kendall notation is used : A/B/N/K/S [**?**]. At each position in Kendall's notation a letter or a number is placed, that describes the type of statistical distribution, or the number of servers, or the size of a queue. The meaning is presented in table 6.1 according to [**?**].

Table 6.1: Kendall notation

Letter	Meaning
A	statistical distribution of time between the arrivals of consecutive call
B	service time distribution
N	the number of servers
K	queue capacity
S	number of traffic sources

The simplest queuing model assumes a Poisson distribution of call arrival times and exponential service time distribution with a single server and infinite queue capacity and an infinite number of sources [**?**]. Since the Poisson distribution is denoted with a letter M and positions with infinite numbers are omitted, such a model is denoted as M/M/1. This queue model is presented in figure 6.2.

A mathematical description of the behavior of such a system is given in [**?**]. There are two random variables that describe input and output: $\tilde{t}$ — interarrival time of calls and $\tilde{x}$ — service time. Each is associated with a statistical distribution. The moments of those distributions are

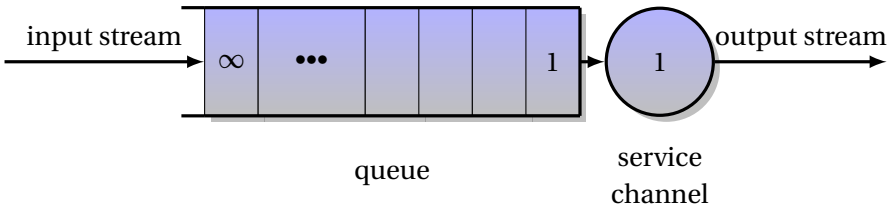


Figure 6.2: M/M/1 queue system [**?**]

defined as follows:

$$E\left[\tilde{t}\right] = \bar{t} = \frac{1}{\lambda} \tag{6.1}$$

and

$$E\left[\tilde{x}\right] = \bar{x} = \frac{1}{\mu} \tag{6.2}$$

Where λ and μ are input and service streams, respectively. Symbol μ is often reserved only for exponential distribution of service times [**?**], and since the M/M/1 system uses this distribution, it is used hereafter.

The M/M/1 queue system contains a single infinite queue and one server. The network consists of many queue systems connected with each other. In order to calculate the utilization factor A_i of the i-th queue system, the following formula is used:

$$A_i = \frac{\lambda_i}{\mu_i} \tag{6.3}$$

The delay on i-th queue is defined as follows [**?**]:

$$T_i = \frac{{}^{1}/_{\mu_i}}{1 - A_i} \tag{6.4}$$

After including the equation 6.3 into the 6.4 the delay is defined as follows [**?**]:

$$T_i = \frac{1}{\mu_i - \lambda_i} \tag{6.5}$$

In the case of serial connections of systems in the network, the end-to-end delay is a sum of delays on the path:

$$T = \sum_{i=1}^{F} \frac{1}{\mu_i - \lambda_i} \tag{6.6}$$

Where F is the number of modules on the path.

The accuracy of this model depends on the input stream and service times compliance with the exponential distribution. The main sender in the decoder modeled in chapter V is the microprocessor that triggers calculations in the decoding modules. Observation of its output gives information about the input stream distribution of all the decoding modules. The microprocessor output has been checked for compliance with the exponential distribution with the use of Kruskall-Wallis statistics[**?**]. The results are given in table 6.2. Each column gathers the resulting *p-values* of that test. In order to accept hypothesis H_0 that the two distributions are compliant, *p-value* needs to be greater than the assumed significance level $\alpha = 0.05$. All the values presented fulfill this requirement, therefore the microprocessor output stream is considered to have the exponential distribution.

6.2 Proposed workflow

A typical workflow is presented in figure 1.1 and consists of a collection of Intellectual Property cores (IP cores) and their statistics, choosing the interconnect architecture and checking it with a software and/or hardware simulation. If the chosen architecture does not fulfill the performance requirements, another architecture is chosen and then checked. This process continues until a satisfactory solution is obtained.

The workflow presented is time-consuming, even though software simulation shortens the time of evaluation of the interconnect performance. Also the results of the simulation need to be further processed and analyzed in order to estimate interconnection capabilities. The estimation of simulation results can be obtained with the use of queue modeling, with the accuracy allowing for a comparison between the interconnection architectures. A new design flow with a proposed new stage is presented in figure 6.3

The first two steps in the modified workflow stay the same, and these are a collection of statistical descriptions of modules and the choice of architecture. These two steps indicate a need to built two graphs: application description and topology, as described in chapter I. Given such knowledge of the network, a queue model can be obtained. As described in section 6.1, the simplest queue model is M/M/1, which is used for further calculations. The main advantage of the proposed workflow (see figure 6.3) is that the inner loop is quick and can be performed repeatedly. Time-consuming simulations are performed rarely.

Table 6.2: Poisson distribution compliance

Frame	QP	*basket* p-value	*city* p-value	*football* p-value	*sunflower* p-value
I	15	0.1788	0.3639	0.3758	0.937
I	20	0.9725	0.8301	0.9176	0.8451
I	25	0.955	0.8994	0.8451	0.973
I	30	0.8068	0.8182	0.7739	0.919
I	35	0.937	0.919	0.7045	0.6144
I	40	0.7303	0.8655	0.5502	0.5739
I	45	0.9911	0.5502	0.5502	0.5502
P	15	0.527	0.2917	0.6929	0.9511
P	20	0.4366	0.8975	0.9253	0.6477
P	25	0.4653	0.734	0.6393	0.972
P	30	0.9511	0.6904	0.631	0.9902
P	35	0.6646	0.8307	0.7476	0.668
P	40	0.598	0.8201	0.4801	1.0
P	45	0.8016	0.8111	0.4581	0.82
B	15	0.9011	0.5048	0.916	0.973
B	20	0.701	0.991	0.973	0.8016
B	25	0.9626	0.8272	0.7915	0.6818
B	30	0.9176	0.8004	0.6875	0.6732
B	35	0.944	0.8201	0.5819	0.6904
B	40	0.8631	0.6477	0.5659	0.8575
B	45	0.8093	0.8602	0.4727	0.5416

6.2.1 Queue system modeling and calculation

According to [**?**] in order to simulate a hardware application, its description is needed in the form of: an application graph (i.e., data flow between modules and computation time for each module) and a topology graph (which describes where modules are placed and what the routers' characteristics are). The same data are needed for queue modeling.

The purpose of queue modeling is to estimate a delay that is needed to decode a macroblock.

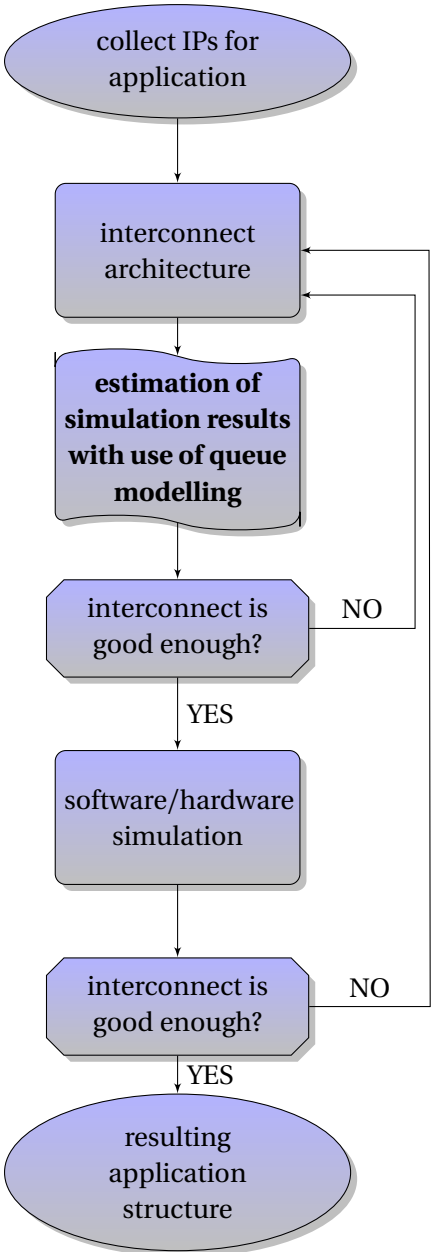
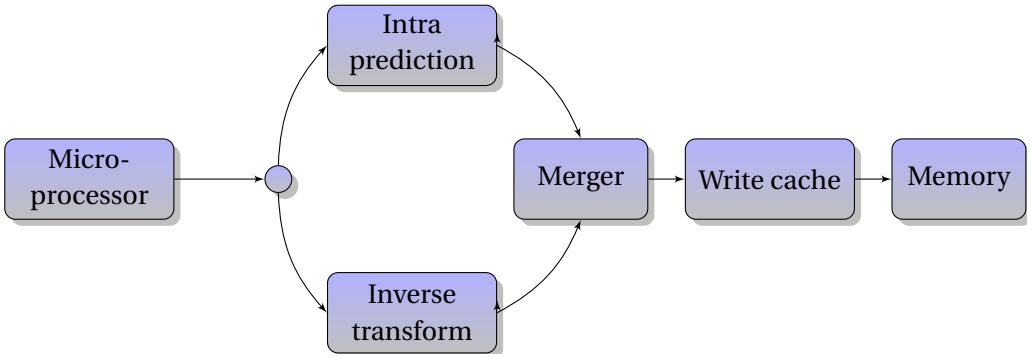


Figure 6.3: Design flow for Network-on-Chip (NoC) with the proposed new step

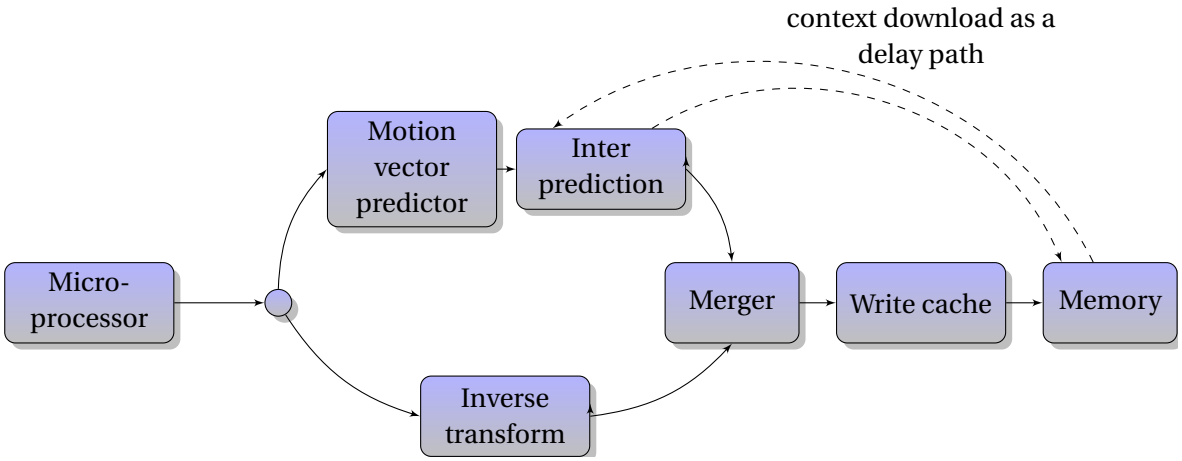
The delay is calculated along the decoding paths, separately for each macroblock type in a frame. The results estimate an average macroblock processing time, and the average values are then used for the computations. The results are expressed in clock cycles.

Decoding paths

The decoding process is performed along two data paths: interframe or intraframe prediction, and inverse transform data paths [**?**] (see figure 6.4). The delay is calculated along those paths. Communication with the memory (downloading a context for prediction) is treated as another delaying path. Such an assumption allows for a simplification of the queue model which does not include loops, only a serial alignment of queue system connections. Another advantage of such an approach is a straightforward calculation of the delay of the context download, and evaluation of its influence on the overall system performance.



(a) Data path for frames I



(b) Data path for frames P and B

Figure 6.4: Data path in the intraframe and interframe predicted macroblocks

Frames, macroblocks and time modeling

In the simulation of a frame of a given type, all the allowed macroblock types (see table 3.1) appear randomly. In queue modeling every macroblock type is considered separately, namely for

each macroblock type, calculations of the delay are performed independently. After performing all the computations, the resulting delay is weighted upon each macroblock-type probability.

In queue modeling, all data transfers are referred to a certain time period, which stays the same for all calculations. In the proposed queue model, the time is expressed in clock cycles, i.e., all the flows are expressed as a number of calls per one clock cycle. Such an assumption allows for expressing the data in the same manner as in the case of the simulation results.

Modules transformation into queue system

All the modules (i.e., Processing Elements (PEs)) are modeled as servers, while ports (sockets [**?**]) are modeled as infinite queues (since the M/M/1 model is used). An exemplary transformation is presented in figure 6.5. The simplest case is a module that contains a single input and a single output port. A model of such a system contains a queue that receives the input stream, a server, and an output queue that receives the output stream.

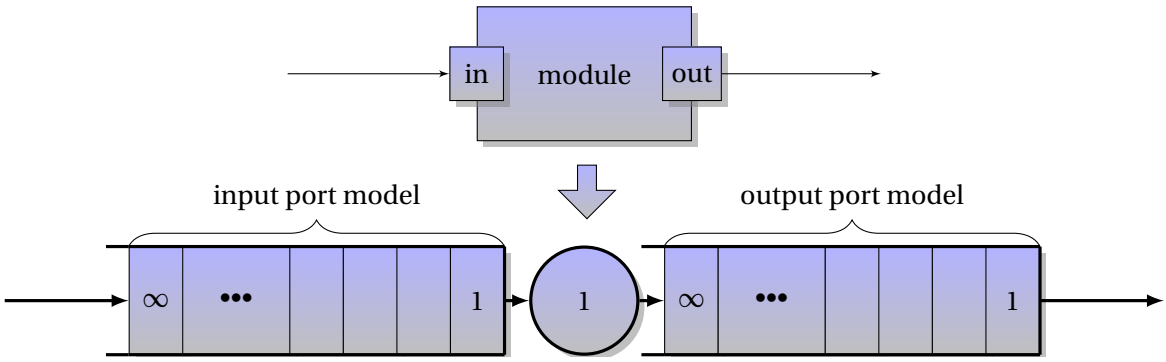


Figure 6.5: Module and its model

If a module contains more than one port, these are modeled as one queue with an adder before it (see figure 6.6). Such a model can be used in the case of one server that requires two calls from two buffers at one time to proceed them, and to send an output stream. However, in the case of the server requiring only one call from either of buffers, scheduling needs to be considered. In the presented decoder such a situation is not, however, present.

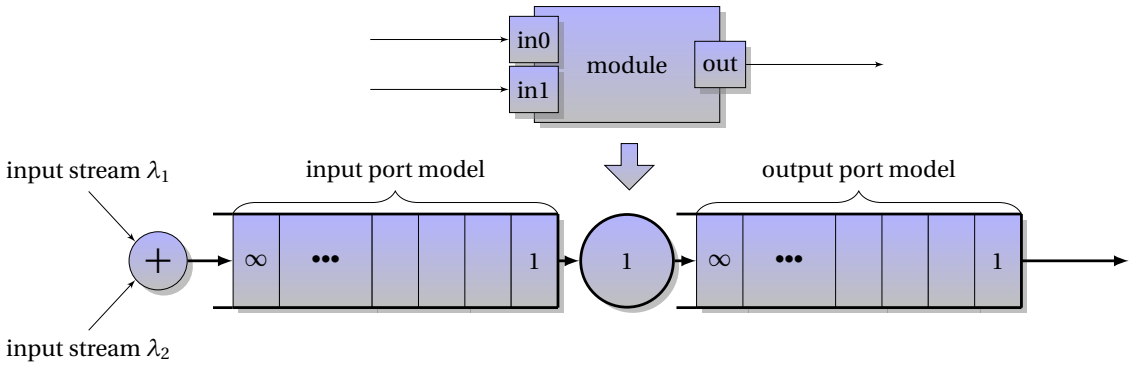


Figure 6.6: Module with 2 input ports and its model

The modules are connected with each other using ports. Directly connected input and output ports (without a server between them) are modeled as a single infinite queue (see figure 6.7).

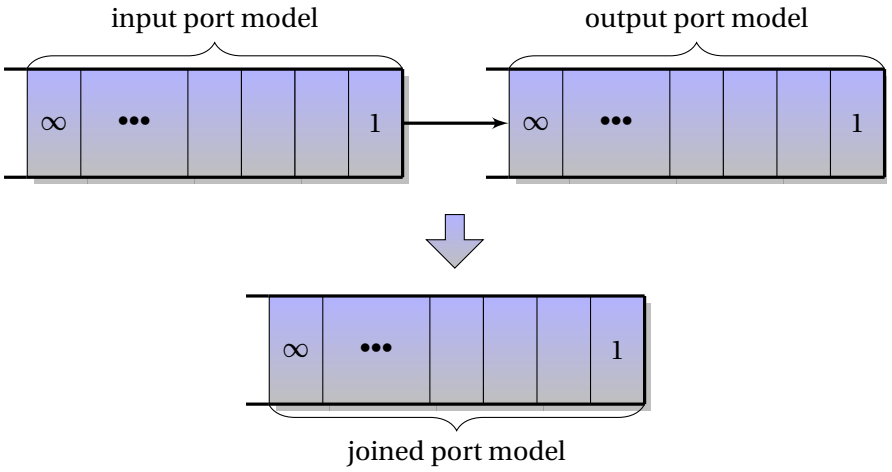


Figure 6.7: Queues

Modeling of input stream

The data gathered in the application graph give the number and size of packets sent from one module to another after some computation time. In order to model the application as a queue system, packets need to be transformed into a stream of calls. Such a transformation requires conversion of all packets into a data stream that consists of data packages of uniform size. The number of such calls corresponds to the size of the. It means, that amount of data stays the same, it is only distributed into a number of calls.

An input stream describes a number of calls that appear in an amount of time. An average time in which calls appear at the input of the system is the same as the average time between

the calls in the output. These streams are equal, according to the law of conservation of a stream [**?**]. The period of time, in which observation takes place, consists of an average computation time and an average buffer filling time. Given that, the input stream of the i-th queue system has been calculated using the following equation:

$$\lambda_i = \frac{n\_call_i}{tc_i + tb_i} \tag{6.7}$$

where n_call_i is the number of calls, tc_i is the computation time of the modeled PE and tb_i is the time needed to fill the output buffer with data. All these values refer to the i-th queue system per macroblock.

Modeling of service stream

A service stream describes the number of calls that are served in an amount of time, needed to produce them. This time is equal to the computation time, hence the equation for the calculation of the i-th system's service stream is given as follows:

$$\mu_i = \frac{n\_call_i}{tc_i} \tag{6.8}$$

where n_call_i is the number of calls and the tc_i is the computation time of a modeled PE.

Modeling of a "black-box" module

Descriptions of some modules do not contain detailed information about the computation time and buffer size (e.g., the producer does not supply such data). In such cases the module's output needs to be observed to collect a statistical description of its performance. Therefore, output packets need to be gathered (for each macroblock type independently) and an average time is assumed as a delay introduced by such a module. The presented approach was applied to the microprocessor.

6.3 The accuracy of queue system calculations

The calculations of the queue model have been conducted with use of the expected values of distributions describing the behavior of a PE. The same distributions were given as an input to the simulation. The results of both processes have been compared in order to estimate queue model accuracy. The queue model, as a pre-simulation stage, needs to estimate the simulation

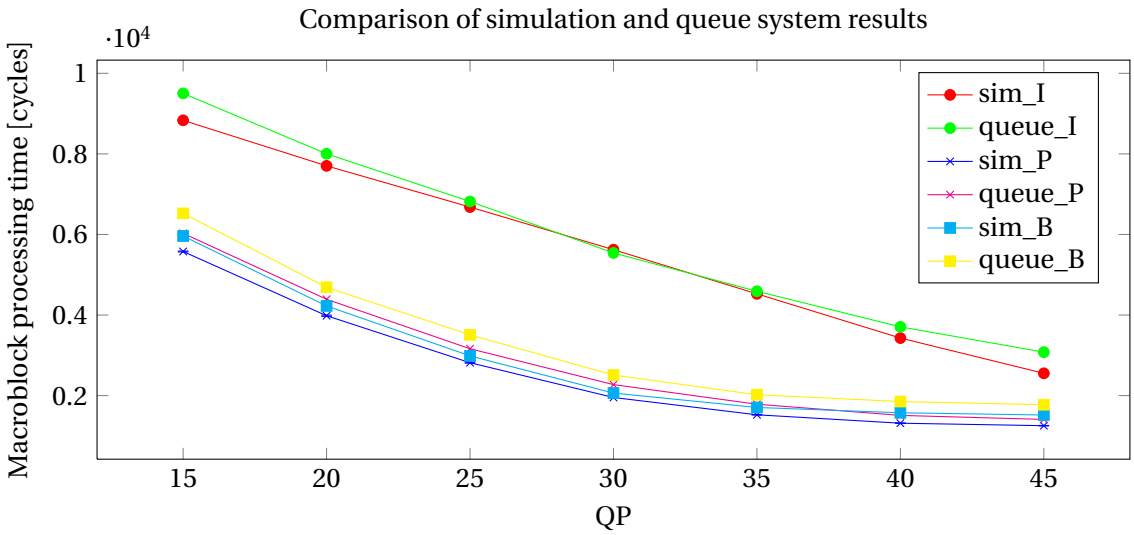


Figure 6.8: Queue system calculations accuracy for *basket* sequence

results, allowing for a comparison between architectures. At first, the queue model results have been compared to the simulation results of the application described in V and presented in section 6.3.1.

6.3.1 Accuracy for four sequences

The presented comparison results are gathered for a system presented in section 5.4. Four video sequences (*basket, city, football* and *sunflower*) have been simulated. The same model has been used as an input of the M/M/1 queue model.

Figures 6.8, 6.10, 6.9, and 6.11 presents the simulation results and their approximation with the use of a queue model. Each figure presents the results for one of the four test video sequences. Detailed results are gathered in tables C.1, C.2, C.3, and C.4.

The results for the *basket* sequence are presented in figure 6.8 (for details, see table C.1). Discrepancies grow with the compression rate, i.e., in the case of frames I they are the lowest (most below 10%), for frames P they are higher (between 8% and 17%), and in the case of frames B, the discrepancies are the highest (between 9% and 21%). In most of cases, the discrepancies are higher than 10%, however the trends are conserved, especially for frames P and B. In these two cases the results of calculations always exceed the values obtained in a simulation, and may be treated as a limit for simulator results.

Simulation and queue modeling results, for the *city* sequence show similar trends (see figure 6.9). A relative difference between the results is the lowest for frames I (all but 2 results are below

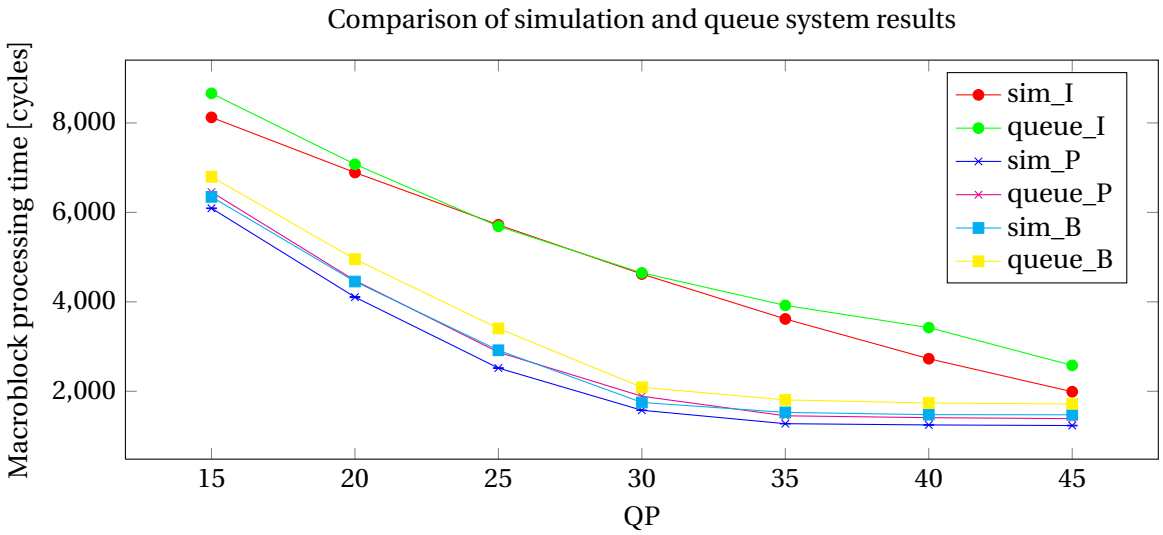


Figure 6.9: The accuracy of queue system calculations for the *city* sequence

10%, the other two are 26% and 30%). Frames P have higher discrepancies, however, all but one below 15% and in the case of frames B the discrepancies reach 19%.

Figure 6.10 shows that in the case of the *football* sequence, the simulation and queue model results follow similar trends. The discrepancies for frames B reach 23% with the minimum at 17%. The curve of the queue model results is significantly above the simulation results. In the case of frames P, the relative difference reaches 21%, and in the case of frames I the highest discrepancy is 20%, although others are below 14%.

Similarly to the *basket*, *city*, and *football* video sequences, the relative difference is the highest in the case of frames B, where it reaches 31% with the minimum of 17% (see table C.4). The discrepancies in the case of frames P are below 19%. For frames I all but 2 discrepancies are below 10% level. In those two cases, the relative difference reaches 16% and 25%. As shown in figure 6.11 the curves representing the simulation and queue model results follow similar trends with greater values for the queue model, especially in the case of frames P and B.

The presented results show that the queue model results can be treated as an upper limit of the modeled system performance. Trends in the case of the simulation and queue model results are consistent, especially in the case of frames P and B. In order to compare the accuracy between sequences, a residual sum of squares has been calculated for each sequence (see figure 6.12). The residual has been calculated as a difference between the values of simulation and queue model results for each Quantization Parameter (QP). Then the residuals were squared

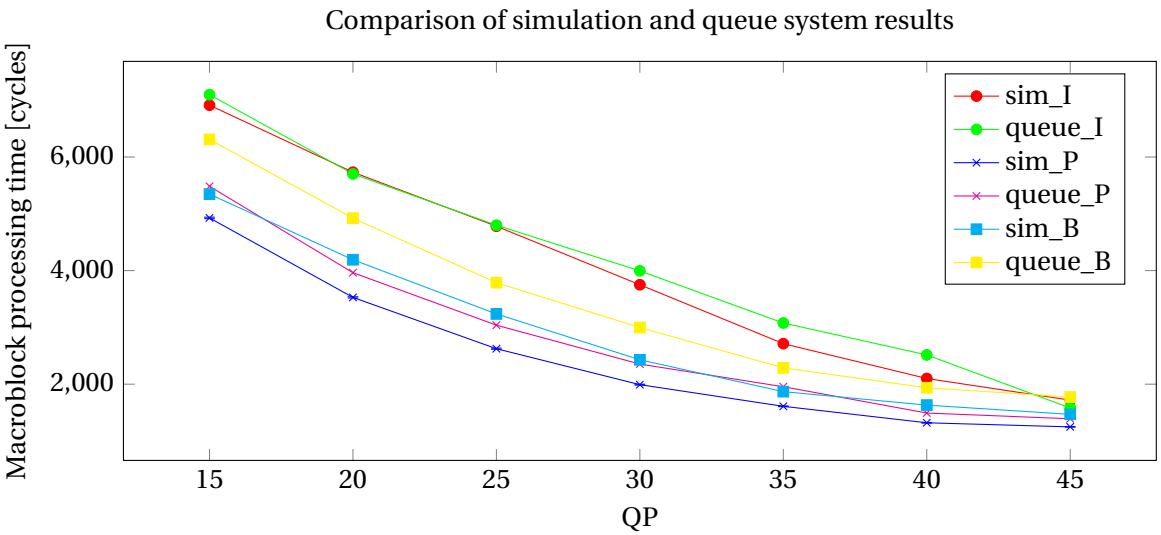


Figure 6.10: Queue system calculations accuracy for *football* sequence

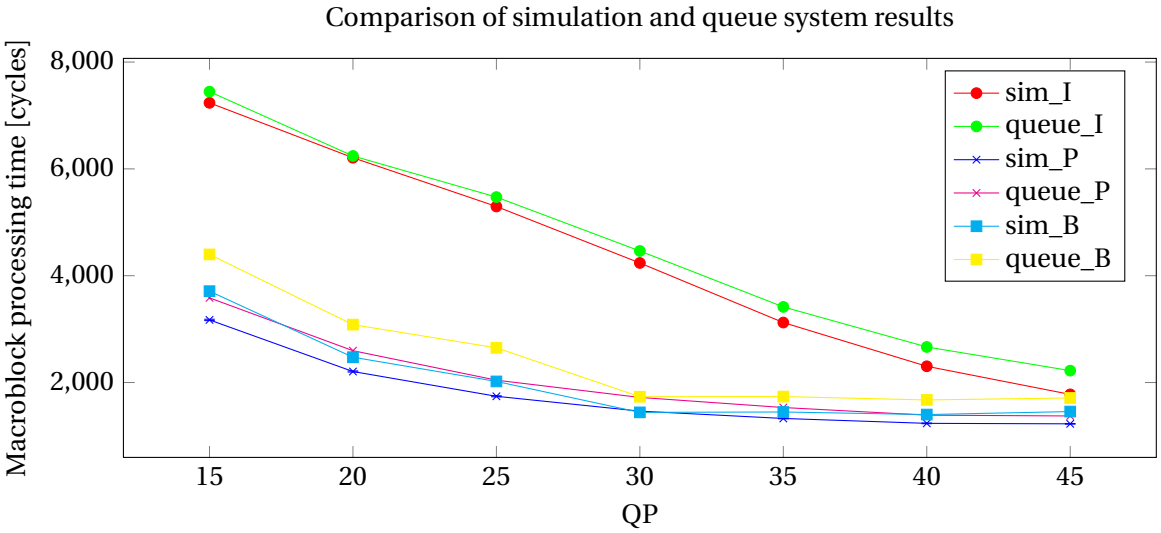


Figure 6.11: The accuracy of the queue system calculations for the *sunflower*

and added together. Figure 6.12 presents those results with respect to the frame type. The size of each bar corresponds to the share of a given frame in the overall approximation error.

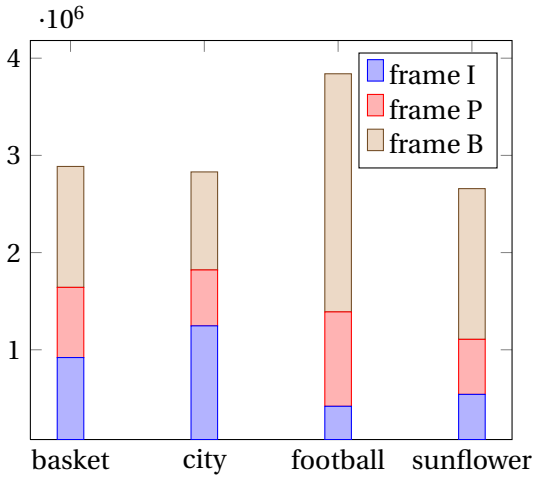


Figure 6.12: Sum of squared differences for the four test sequences

The queue model accuracy depends on the sequence content (refer to figure 6.12). In the case of *basket*, *city*, and *sunflower*, the overall residual sum of squares is similar, with a different share in the error for frames I, P and B in each case. Only in the case of the *football* sequence, a much higher approximation error is observed. Such an error is caused by the coding schemes different from the other sequences, resulting from the black stripes at top and bottom of the frame. Those stripes are characterized with a very good compression rate, especially in the bidirectional interframe prediction mode. In a such case the queue model gives the limit of system performance, because it considers only the expected value of computation time distribution.

6.3.2 Accuracy for multi-scene video sequence

The four video sequences presented, (*basket*, *city*, *football*, and *sunflower*) contain a single scene each, which means that no cut occurs. In the real-life video, such a situation is no longer the case. In order to estimate the accuracy of the queue model in a general case, data for 21 test video sequences were gathered, to obtain general statistical distributions. The procedure of gathering of those data is the same, as if those video sequences were put in a single, long video, one after another. Those statistical distributions are input to the simulator, and to the queue model. The results of both were compared and presented in figure 6.13 and the detailed results are presented in table D.1.

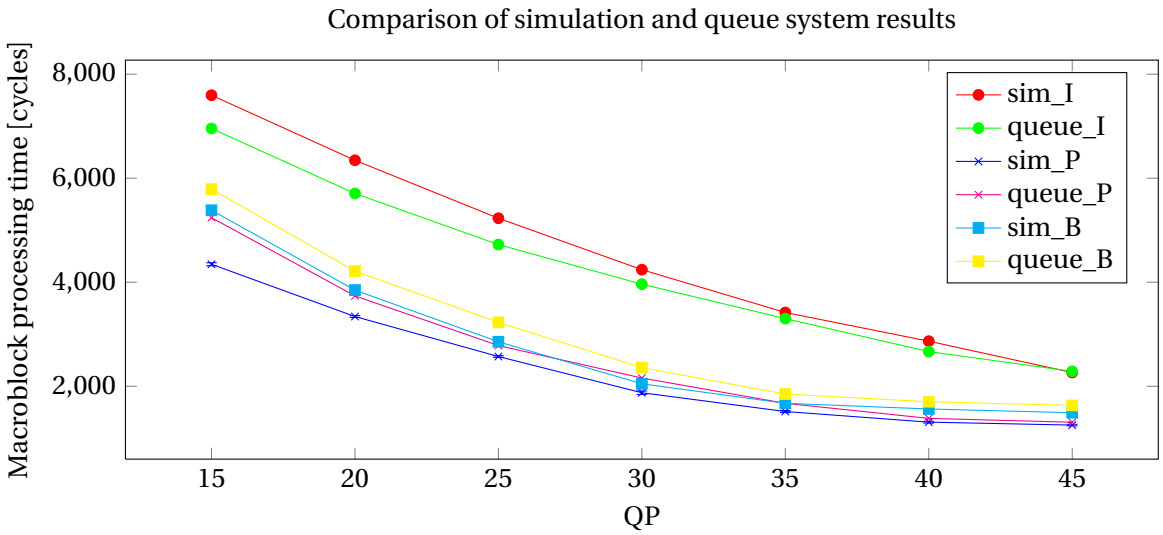


Figure 6.13: The accuracy of queue system calculations for a multiscene video sequence

The discrepancies between the simulation and queue model results are similar to those obtained for the four single-scene sequences. The absolute differences are greater for small QP values (i.e., QP = 15,20,25), however in those cases the simulation results have the greatest values, which in turn produces small relative differences (about 10%). The highest relative difference is in the case of frames P at QP = 15 and equals 21%, which is also reflected in figure 6.13. All the other discrepancies stay below 16%.

6.3.3 Sources of inaccuracies between the simulation and queue modeling

As mentioned in 6.1, the M/M/1 model assumes exponential service time distribution in servers and Poisson distribution compliant input stream. In the application presented, it is not the case for all modules. Table 6.2 presents Poisson distribution compliance only for the microprocessor, which is the major source of data in the application, since it triggers all calculations in the modules. Other PEs cannot be characterized in this way. The absolute differences are in general greater for small QP values (i.e., QP = 15,20,25) than in the case of QP = 40,45. which suggests worse compliance to the Poisson distribution in the microprocessor output stream. It is also noticeable by the *p-values* level in table 6.2, where in case of QP = 15 for frames I and P, the *p-values* are significantly smaller than in other cases.

Another source of inaccuracies is the transformation of packets, that are sent in the simulation, to the stream of calls, that are used in the queue model. The calls are equal-sized data

packages sent over a network. As mentioned before, a packet is divided into a number of calls, depending on the packet size. Such a transformation is an approximation that may introduce an error.

6.4 Analysis of modules performance

The queue model allows for the analysis of a delay introduced by each module on the data path. As shown in figures 6.4, in the Advanced Video Coding (AVC) decoder there are two data paths for interframe and intraframe predicted macroblocks. In sections 6.4.1 and 6.4.1, the analysis of the delay on each data path is provided. The delay was calculated using the equation 6.5, and the calculated value reflects the difference between the service and call streams. The greater the difference (μ_i is always greater than λ_i), the lower is the delay introduced by the module. On the other hand, if the service and call streams differ slightly, the delay introduced by the module grows proportionally. Each result presented is an average value of the seven results, each for one QP value: QP $= 15, 20, \ldots, 45$. The following results do not take into account the delay introduced by communication infrastructure, it is only an analysis of the delay introduced by each IP core.

6.4.1 Intraframe predicted macroblocks

As mentioned in chapter III, there are two types of macroblocks in frame I: of 4×4 and 16×16 intraframe prediction mode. According to table 3.1, frames I may appear in all frame types (I, P and B). Given that, table 6.3 presents the delay introduced by each module on the data path of macroblocks I for each frame type. The greatest delay is introduced by the microprocessor (with its accelerators). For both macroblock types, this delay exceeds the combined delay of all other modules. The results presented show that parsing of a bitstream is the most delaying process for these macroblocks. Moreover, the decoding process in the case of 4×4 macroblocks is two times longer than in the case of 16×16 macroblocks I. In the first case, there is more (i.e., 16) prediction types to be decoded from the bitstream (in the case of 16×16 macroblocks it is one). Furthermore, a 4×4 intraframe prediction is chosen for macroblocks with more complicated contents, that is more difficult to predict. As a result, a larger number of transform coefficients is produced.

The delay introduced by the intraframe prediction module differs significantly for 16×16 and 4×4 macroblocks. In the first case the delay equals to about 1 clock cycle, whereas in the latter

case it is 321 or 217. These numbers show the difference between the call stream and service stream of the module. In the case of the 16×16 intraframe prediction, some data (single prediction mode) are sent, which are processed in a short time. On the other hand, the results for the 4×4 prediction suggest that due to the little difference in the service and call streams the data accumulate.

The analysis of the delay values introduced by the inverse transform module show that for each frame type the inverse transform result is processed longer for the 4×4 macroblocks I. As mentioned before, in this case there are more transform coefficients to send. Such a mismatch of the input and service stream results in a higher delay, due to data accumulation in the input queue. In the case of the 16×16 macroblocks there are fewer coefficients, which are processed longer than in the case of the 4×4 macroblocks I (due to the Hadamard transform, as described in section 3.1.1).

Other modules introduce the same delay, since the amount of data and service time in each case is the same. The delay path in the last column refers to the context update in the intraframe prediction module. A decoded macroblock is received in the write cache, which sends data to the intraframe picture context and to the memory. The intraframe picture context module prepares the data format, and sends them to the intraframe prediction module (see section 4.3.1). Since the context must be updated before the next macroblock is decoded in the intraframe prediction module, it is a delaying path of the macroblock decoding process.

Table 6.3: Summary of modules performance for intraframe predicted macroblocks

Frame	MB type	Delay					
		Micro-processor [cycles]	Intra prediction [cycles]	Inverse transform [cycles]	Merger [cycles]	Wcache [cycles]	Delay path [cycles]
I	4×4	4681	321	16	21	0.75	21
P	4×4	4408	218	17	21	0.75	21
B	4×4	4455	218	17	21	0.75	21
I	16×16	2285	1.09	14	21	0.75	21
P	16×16	2050	1.09	16	21	0.75	21
B	16×16	2970	1.09	13	21	0.75	21

6.4.2 Interframe predicted macroblocks

There are more interframe predicted macroblock types than in the case of intraframe prediction. Each type refers to a different partitioning scheme, and reference direction. In order to mark macroblock types, a following description scheme is introduced: *XX_YY_SIZE*. The *XX* and the *YY* codes refer to the prediction direction of one of the macroblock's big partitions. The values available are *L0*, *L1* and *Bi* which stand for the backward, forward and bidirectional prediction of a partition. The *SIZE* code refers to the partitioning scheme, as shown in figure 3.3(a). For example code *Bi_L0_16×8* means that a macroblock is partitioned into two parts, each with 16×8 samples: the first partition is bidirectionally predicted, and the second one references only to the previous frame. If there is only one partition, the code is as follows: *XX_SIZE*, where *XX* refers to the prediction direction of the single partition. In the case of partitioning of 8×8 samples, the reference direction is not specified. In frames P, there is another macroblock type, which means that all partitions refer to the same picture, and are marked as *8×8ref0*. Another macroblock type is *Skip*, which refers to macroblocks encoded with no transform coefficients. The codes described are used in tables 6.4 and 6.5 to denote the macroblock type.

Table 6.4: Summary of modules' performance for interframe predicted macroblocks

Frame	MB type	Delay						
		Micro-processor [cycles]	Motion vector predic-tor [cycles]	Inter predic-tion [cycles]	Inverse trans-form [cycles]	Merger [cy-cles]	Wcache [cycles]	Delay path [cycles]
P	Skip	1185	33	25	23	21	0.75	0.07
P	L0_16×16	2486	38	25	19	21	0.75	0.07
P	L0_L0_16×8	2865	39	16	14	21	0.75	0.07
P	L0_L0_8×16	2695	39	22	17	21	0.75	0.07
P	8×8	3417	18	21	13	21	0.75	0.02
P	8×8ref0	3286	18	21	13	21	0.75	0.02

The results gathered in tables 6.4 and 6.5) show that, similarly to intraframe prediction, the delay introduced in the parser dominates in the overall decoding process. The fastest decoded macroblock type is *Skip* with nearly 1200 clock cycles. These macroblocks are encoded with a flag

Table 6.5: Summary of modules' performance for interframe predicted macroblocks

Frame	MB type	Delay						
		Micro-processor [cycles]	Motion vector predic-tor [cycles]	Inter predic-tion [cycles]	Inverse trans-form [cycles]	Merger [cy-cles]	Wcache [cycles]	Delay path [cycles]
B	L0_16×16	2414	86	26	19	21	0.75	0.06
B	L1_16×16	2073	83	24	18	21	0.75	0.06
B	Bi_16×16	2687	90	38	26	21	0.75	0.07
B	L0_L0_16×8	3113	66	17	12	21	0.75	0.07
B	L0_L0_8×16	3222	62	23	14	21	0.75	0.07
B	L1_L1_16×8	2674	64	16	13	21	0.75	0.07
B	L1_L1_8×16	2809	61	21	14	21	0.75	0.07
B	L0_L1_16×8	2837	65	15	12	21	0.75	0.07
B	L0_L1_8×16	3142	63	24	13	21	0.75	0.07
B	L1_L0_16×8	2702	65	15	12	21	0.75	0.07
B	L1_L0_8×16	3261	63	22	12	21	0.75	0.07
B	L0_Bi_16×8	3275	68	22	13	21	0.75	0.06
B	L0_Bi_8×16	3183	64	33	17	21	0.75	0.06
B	L1_Bi_16×8	2965	65	21	15	21	0.75	0.06
B	L1_Bi_8×16	2920	60	32	19	21	0.75	0.06
B	Bi_L0_16×8	3085	68	22	13	21	0.75	0.06
B	Bi_L0_8×16	3528	63	31	17	21	0.75	0.06
B	Bi_L1_8×16	3174	66	30	13	21	0.75	0.06
B	Bi_Bi_16×8	2819	62	33	24	21	0.75	0.05
B	8×8	4861	38	17	10	21	0.75	0.04

indicating the *Skip* macroblock series, and their number. This means that bitstream decoding in such a case is extremely fast and most of the time is spent on reloading the context and sending the information about the macroblock's position and prediction type for reconstruction. The longest decoding time is observed for 8×8 macroblocks in B frames. Such a result is related to the

most complex prediction scheme, and the amount of data needed for decoding. Decoding time in the case of other macroblock types is more dependent on the prediction efficiency (i.e., the number of transform coefficients) than on the amount of data describing the decoding process (i.e., the reference frame indicators and motion vectors). For example the decoding time of *Bi_Bi_16×8* in B frames is shorter than the decoding time of *L0_L0_16×8* in frames P.

A motion vector predictor introduces a delay of up to 40 clock cycles in frames P and up to 90 clock cycles in frames B. It is caused by the increased context transfers in frames B (motion vectors are written for two prediction directions, regardless of the current prediction type). This ensures that the later downloaded context is always filled correctly for each prediction direction with the appropriate data, even though some are not used. Such an approach makes the downloading data algorithm more regular and the data format is easier to handle.

The interframe prediction introduces a delay from 15 up to 38 clock cycles. For example, performing a bidirectional prediction of 16×16 macroblock delays the data by about 38 clock cycles, however, the prediction of *Bi_Bi_16×8* macroblock introduces a delay of 33 clock cycles. The latter case requires twice as much download of the context data from the memory than in the first case. For a *Bi_Bi_16×8* macroblock the data are coming to the interframe prediction module at a lower pace, which gives the prediction module the time to perform calculations.

Inverse transform module introduces a delay from 10 up to 27 clock cycles. In the case of *Skip* macroblocks, only an indication of no coefficients is sent, and the transform module produces zeros at the output. The pace of acquiring data is higher than in the case of other macroblock types, and the delay introduced is rather high (22 clock cycles). The two other high delay values (24 and 27 clock cycles for *Bi_Bi_16×8* and *Bi_16×16*, respectively) are related to the prediction efficiency and the resulting mismatch of the size of call and service streams.

The delay introduced by other modules is the same as in the case of frames I. The delay path consists only of the delay introduced by the modules (context download handling and memory response). Data transfer delay is not taken into account.

6.5 Benefits and limitations of the proposed queue system modeling

The delay analysis in section 6.4 presents a few benefits of queue modeling of System-on-Chip (SoC). The first is a logical division into data paths, and their delay analysis. Such an analysis allows for the identification of bottlenecks in the system. The calculated results also show

how much the call stream is unbalanced with the service stream. In the case of e.g., prediction modules, the values are satisfactory, however, the microprocessor introduces a delay that conditions the overall macroblock processing time. This also means that the bitstream parser needs further optimization. The scale of the parser delay is largely dependent on the AVC compression standard specification [**?**], which restricts the parallelization of bitstream decoding.

The proposed application of queue modeling to SoC also allows for the separation of the delay introduced by modules on the data path from the time needed to e.g., download data from the memory. This allows for the estimation of how much accessing the memory delays the decoding process.

Although queue modeling allows for a flexible analysis of the IP core performance, it still cannot replace the simulation to analyze the hardware application in detail, i.e. to observe system reaction in various circumstances. However, queue modeling proves its usefulness even with the use of the simplest model.

CHAPTER VII

Network-on-Chip architectures for video codecs

7.1 Introduction

The comparison of a few hardware codec implementation proposals is described in section 3.2, and it shows that such an application is rather small to be implemented with the use of a Network-on-Chip (NoC). Nevertheless, section 4.4 discusses many benefits of a NoC implementation for the video codec proposed in chapter IV. The proposed architecture was implemented at a very low cost, to reduce hardware consumption, and is based on a simple packet exchange. A modification that can be made in such a structure, concerns the choice of topology and modules' placement, since any additional functionality would increase the hardware cost.

Goal

The goal of the following topology exploration is to find the optimal architecture for an advanced video codec. Such architecture should provide system performance at the lowest hardware cost. System performance is defined as the average macroblock processing time, and hardware cost is the number of Look-up tables (LUTs) and Flip-Flops (FFs) required to implement such a proposal in an Field Programmable Gate Array (FPGA) device. Topology exploration concerns both, the structure and placement of modules.

Optimization

The optimization of module placement in the topology was the basic criterion for the preparation of the simulation. An application can be defined by a graph of the application $G_a = (V_a, E_a)$ in which vertices (V_a) represent modules and edges (E_a) represent data flow. Each vertice can be

described with size of data sent between two modules. The application graph can be mapped onto a structure of a communication network, described with a graph of topology $G_t = (V_t, E_t)$, where vertices V_t represent a node in the communication structure, and edges E_t represent connections between nodes. Optimization of module placement answers the question of how to map graph G_a of the application onto a topology graph G_t, to obtain the best overall system performance.

The mapping function assigns modules to the specific nodes in the network, and can be defined as follows:

$$\text{map}(G_a, G_t) = \{V_a \mapsto V_t\} \tag{7.1}$$

with weights (size of data flow) assigned to arcs in G_t.

The author chose a criterion of minimization of distance in a weighted G_t graph, so that data would travel over a minimum number of hops f:

$$\forall_{s,d}, \quad \text{where} \quad s,d \in V_t \quad \text{and} \quad s \neq d \, f(s,d) = \sum_s \sum_d h(s,d) \cdot b(s,d) \tag{7.2}$$

where $h(s,d)$ is hop count in terms of the number of network devices between the source (defined as s) and destination (noted as d) in the topology graph, and $b(s,d)$ is number of bytes sent between the two.

The author considered NoC topologies that serve not only the traffic between the parser and predictor, but also the traffic between the modules of the predictor and traffic to the memory. It means that the traffic between the parser and predictor is no longer separated from the traffic between the predictor modules and from the traffic to the memory. Such an assumption allows for a comparison between the architecture with traffic separation (i.e. a proposed architecture in chapter IV) with the architecture of non-separated traffic.

7.2 Topology exploration methodology

Explored topologies were chosen among the proposals presented in chapter II, and these are the following

- **fat tree**; In order to reduce the number of routers, the author used a configuration that attaches 3 modules at the most to each router of the lowest level. Also, due to the hardware consumption reduction, there is only one additional level of routers to connect routers

from the lower level. Additionally, each lower-level router is connected to all the higher-level routers. The number of ports was not restricted, as in the case of a star topology, since it would increase the number of routers needed to provide connectivity between all the modules.

- **mesh**; Mesh topology was configured as described in section 2.3. The author allowed for routers that are not connected to the modules to preserve regularity of the network.
- **ring**; Ring topology corresponds to the architecture proposed in chapter IV, however it is built with the use of routers instead of NIs with a multiplexer inside.
- **star**; In order to make routers smaller, the author decided that the routers will contain 4 ports at the most. Such a configuration is synthesized well on FPGAs.
- **binary tree**; Each router contains 4 ports at the most: one to connect with the upper level router, two to connect with the lower-level routers and one port to connect the module.

The proposals have been compared to the architecture proposed in chapter IV.

The placement of modules for each topology was chosen based on the criterion described in section **??** which is the minimization of the distance between modules with respect to the amount of data sent between them. The optimization result was obtained using the Monte Carlo method. Calculations were performed based on the statistical data gathered for 21 video test sequences. It is the situation of a long video sequence that contains cuts between the scenes of different content. This optimization was performed for frames P and separately for 7 Quantization Parameter (QP) values: 15, 20, 25, 30, 35, 40, 45. Frames P were chosen, since they contain intraframe and interframe predicted macroblocks, and do not contain bidirectionally predicted macroblocks. These macroblocks occur in frames B, and differ mostly with the size of traffic between the interframe predictor and the memory. In a few cases the optimal (in the sense of optimization criterion described in section **??**) module arrangement differs for the different QP values, since the size of traffic between the modules changes with the compression rate. The results of optimization are gathered in table 7.1.

7.3 Comparison results

The topologies listed in table 7.1 where simulated to estimate their performance. Each of the topologies mentioned was checked for 3 frame types: I, P and B, and for 7 QP values: 15, 20, 25, 30, 35, 40, 45. The result of each test case is the average time needed to process a single macro-

Table 7.1: Optimization results for different architectures. Columns contain median, minimum, and maximum of f (see equation 7.2) results for frame P and QPs = (15, 20, 25, 30, 35, 40, 45).

Topology	median	minimum	maximum
fat tree for QP 15-40	7112	6979	7249
fat tree for QP 45	7150	6982	7349
binary tree for QP 15	9201	8992	9443
binary tree for QP 20-45	8972	8569	9473
mesh for QP 15-35	8319	8103	8581
mesh for QP 40-45	8312	8124	8581
ring for QP 15-45	8826	8564	9141
star for QP 15-40	7112	6979	7254
star for QP 45	7150	6996	7349

block, which is further used as a basis for a comparison. It means that for each topology 21 values were obtained (3 frame types, each with 7 QPs). The author chose 4 metrics for the comparison between the architectures: average, median, minimum and maximum value obtained for each topology.

The average is the mean of all of the 21 average macroblock processing time values obtained for a given topology. The median, minimum and maximum were chosen to get a picture of the distribution of values. The results are gathered in table 7.2.

The differences between the means for different topologies are small and in a few cases differ by less than 10 clock cycles. The best result was obtained for the architecture proposed in chapter IV. The result is caused by traffic separation into two segments: the NoC that transports packets between the parser and predictor is separated from data exchange between the modules of the predictor and the memory.

The difference between the highest and the lowest value of a median equals to 232 clock cycles which is about 9% of the lowest value, and shows small differences between the results. The lowest median values were obtained for the star and mesh topologies. The highest value of the median was obtained for the fat tree topology with module arranegment optimal for high QP values.

The lowest maximum and minimum values were obtained for the proposed architecture,

Table 7.2: Simulation results for different architectures. CI is the confidence interval of the value.

Topology	Average [clock cycles]	Median [clock cycles]	CI [clock cycles]	Maximum [clock cycles]	CI [clock cycles]	Minimum [clock cycles]	CI [clock cycles]
fat tree for QP 15-40	3218	2869	19	7604	36	1276	23
fat tree for QP 45	3232	2884	4	7662	39	1264	14
binary tree for QP 15	3222	2870	26	7622	30	1272	39
binary tree for QP 20-45	3227	2861	26	7649	32	1307	82
mesh for QP 15-35	3238	2869	25	7620	41	1265	39
mesh for QP 40-45	3254	2661	17	7642	58	1286	56
ring for QP 15-45	3264	2676	12	7692	47	1284	51
star for QP 15-40	3237	2652	29	7650	23	1270	19
star for QP 45	3251	2682	18	7672	51	1266	15
Proposed architecture (chapter IV)	3192	2855	19	7595	36	1254	15

however, the size of confidence interval makes these results statistically insignificant when compared to other results. Nevertheless, the difference between the maximum and minimum average processing time is the lowest for proposed architecture.

7.4 The choice of optimal architecture

The results presented in 7.2, do not give a straightforward indication on the optimal communication structure for an Advanced Video Coding (AVC) decoder. The choice needs to be based on other factors, such as hardware consumption. As mentioned before, a video decoder as well as an encoder are relatively small applications to connect their modules with a NoC. The re-

Table 7.3: Simulation results for different architectures. CI is the confidence interval of the value.

Topology	routers to modules ratio	mean rank	median rank	graph planarity	Maximum I:P:B
Metric number	1	2	3	4	5
fat tree for QP 15-40	1:1.5	2	7	no	1:0.58:0.71
fat tree for QP 45	1:1.5	5	9	no	1:0.58:0.71
binary tree for QP 15	1:1	3	8	yes	1:0.58:0.71
binary tree for QP 20-45	1:1	4	6	yes	1:0.57:0.71
mesh for QP 15-35	1:1	7	7	yes	1:0.58:0.71
mesh for QP 40-45	1:1	9	2	yes	1:0.65:0.71
ring for QP 15-45	1:1	10	3	yes	1:0.65:0.71
star for QP 15-40	1:2.25	6	1	yes	1:0.64:0.71
star for QP 45	1:2.25	8	4	yes	1:0.64:0.70
Proposed architecture (chapter IV)	1:4.5	1	5	yes	1:0.57:0.71

sults obtained for the different topologies confirm that conclusion. The obtained results show a similar performance for different topology structures. It means that the topology does not influence system efficiency. Given the results, the choice of optimal architecture needs to be based on other factors, that cover hardware consumption and operating frequency analysis with respect to Group of Pictures (GOP). A few metrics that correspond to the mentioned factors are gathered in table 7.3. The first is the ratio of routers to modules, which shows how many additional network elements need to be added to build a topology. The second and the third are the performance ranks based on the average and the median respectively. The fourth metric is an information on the topology graph planarity. The last metric is the ratio of processing time between maximums for frames I and P, and B. The maximum average value is obtained for QP = 15 for each frame type. The values for frames P and B were related to the maximum average processing time of frames I. It shows the ability of balancing the processing time for different frame types, as described in section 5.7.3.

Table 7.4: Synthesis results for NoC router and NI with and without switching capability for Xilinx Virtex-4 device. BD means bidirectional and UD means unidirectional interface.

NoC module	number of LUTs	number of FFs
router 2IN 2OUT (proposed architecture)	430	330
router 3IN 3OUT	774	495
router 4IN 4OUT	1032	660
router 5IN 5OUT	1290	825
NI BD without MUX	232	160
NI BD with MUX (proposed architecture)	310	190
NI UD without MUX	81	78
NI UD with MUX (proposed architecture)	110	100

Hardware consumption

Hardware consumption, which is related to topology, is based on a few factors: the number and size of network elements, and the planarity of the topology graph. The number of network elements can be characterized by routers to modules ratio which gives an average number of modules per router. Routers are the most expensive network elemnets, in terms of hardware consumption, and minimization of their number reduces the overall hardware cost of a NoC. The best ratio was obtained for the proposed architecture, which required only two routers (each containing 2 inputs and 2 outputs) to connect the modules. This result could be even better, if routers were eliminated, in the case of implementation on a single chip. However, the architecture proposed requires network interfaces that contain switching capability. Nevertheless, they do not contain the routing table, and buffer just a few words of a packet. Therefore, their size is much smaller than the size of a router (see synthesis results for the proposed architecture in table 7.4). Other architectures require routers that contain, more than the proposed architecture, inputs and outputs, however the author assumed the same packet switching policy. Synthesis results for such network elements are gathered in table 7.4. The increased size of network elements is caused mainly by the increased number of buffers due to a greater number of ports.

In order to compare different architectures, a estimation of the network size has been per-

formed. Synthesis results from table 7.4 have been used with respect to the number and type (e.g., number of ports in a router) of elements needed for each topology. The results are gathered in table 7.5.

Modification of proposed topology

The topology proposed in chapter IV does not forward traffic between the modules of the predictor and the memory (see figure 5.3). In order to estimate the size of an architecture that would separate traffic, as the case of the proposal, and would connect predictor modules with the use of a NoC, the estimated size of a modified structure has been calculated. The modification assumes that peer-to-peer connections and memory bus are removed, and in that place NoC is introduced. Since the whole predictor is placed on a single chip, no additional router is needed. The modification assumes an additional one unidirectional (UD) interface (for inverse transform module) and five bidirectional interfaces (BD) for the rest of the modules. A hardware cost for such architecture is given in the last row of the table 7.5.

Although the proposed modified architecture is nearly twice as big as the non-modified proposal, it is still the smallest architecture. On the other hand, the binary tree topology results in the highest total cost of NoC, since it requires 9 routers, each with 4 inputs and 4 outputs. Another topology of high hardware cost is the mesh, which requires a single 5 input and 5 output router, and 4 routers of 4 and 3 ports.

Other architectures than those proposed in chapter IV are characterized with a worse routers to modules ratio, and they do not feature the performance results that would justify the hardware cost. Also, all but two topologies are characterized with a planar graph layout. It means that the placing and routing of the software is theoretically able to make busses on the chip without intersections other than in the routers. In the case of non-planar graph topology, such intersections are inevitable, and require additional switching devices, which in turn causes hardware consumption and frequency loss.

Operating frequency selection analysis with respect to GOP

The selection of operating frequency for the final hardware implementation was described in detail in section 5.7.2, and concerns two types of systems: of low and high delay. The selection of operating frequency for a low delay system requires compliance with the worst case scenario, which is the case of intraframe predictions and low QP values. For the simulated topologies, the

Table 7.5: Estimated size of a NoC for different topologies.

Topology	routers	NIs	total NoC cost
fat tree	6186 LUTs and 3960 FFs	810 LUTs and 780 FFs	6996 LUTs and 4740 FFs
binary tree	8388 LUTs and 5940 FFs	810 LUTs and 780 FFs	9198 LUTs and 6720 FFs
mesh	8112 LUTs and 6225 FFs	810 LUTs and 780 FFs	8922 LUTs and 6225 FFs
ring	6966 LUTs and 4455 FFs	810 LUTs and 780 FFs	7776 LUTs and 5235 FFs
star	3870 LUTs and 2475 FFs	810 LUTs and 780 FFs	4680 LUTs and 3255 FFs
Proposed architecture (chapter IV)	860 LUTs and 660 FFs	1100 LUTs and 1000 FFs	1960 LUTs and 1660 FFs
Proposed architecture (modification)	860 LUTs and 660 FFs	2760 LUTs and 2050 FFs	3620 LUTs and 2710 FFs

lowest maximum value was obtained for the proposed architecture, however all the results are characterized with small differences between each other and are statistically insignificant.

In the case of a high-delay system, which allows for balancing between the processing times for frames I, P and B, the relation of the processing time between intraframe and interframe prediction needs to be considered. The last column of table 7.3 presents the ratio between the maximum average processing time for each frame. The author considered the case of GOP used in digital television systems which contains 12 pictures [**?**]. Since a GOP may contain a different number of P and B pictures, the author considered 3 scenarios of GOP containing:

- 1I, 1P, 10B,
- 1I, 2P, 9B,
- 1I, 3P, 8B.

These scenarios were applied to the best (1:0.57:0.71) and the worst (1:0.65:0.71) ratios obtained. Based on the relations and the maximum average processing time obtained, frequency was calculated in each case. The results are gathered in table 7.6 and show that the difference grows with the number of frames P in the GOP. Nevertheless, the differences between the results are below 10MHz. The best ratio was obtained for two topologies: the proposed architecture and the binary tree with the arrangement of modules for QP $= 20 \ldots 45$.

Table 7.6: Operating frequency

GOP	Frequency [MHz]	
	1:0.57:0.71	1:0.65:0.71
1I, 1P, 10B	218	223
1I, 2P, 9B	214	221
1I, 3P, 8B	211	220

Conclusions on the architecture choice

The simulation results obtained show that performance differences for the explored topologies are relatively small, which leads to the conclusion that a NoC architecture choice can be made based on other factors. The author presented an analysis on hardware consumption and the selection of operating frequency. Based on the results, the architecture of the lowest hardware consumption is the architecture proposed in chapter IV. It also produces the best ratio of processing time between intraframe and interframe predictions, allowing the choice of a lower operating frequency. Nevertheless, the hardware consumption factor gives the strongest premise of the choice, since the performance results are similar in most of the cases.

CHAPTER VIII

Recapitulation and conclusions

8.1 Recapitulation

The dissertation has been focused on the application of Network-on-Chip (NoC) for advanced video codecs with the emphasis on implementation and debug issues. Although video codecs are relatively small applications to exploit NoC as an interconnect technique (see section 3.2), the author proves the applicability of those structures for video codecs (chapter IV). The advantages and disadvantages of adopting NoC as an interconnect technique for an Advanced Video Coding (AVC) codec has been discussed in tables 4.2 and 4.4, and compared with other available connection techniques. The comparison shows the benefits of NoC which are:

- the reduction of the number of connections,
- scalability in terms of operating frequency and introducing architectural changes,
- debugging capability,
- flexibility in placing and routing in final hardware implementation.

These benefits outbalance the cost of NoC in terms of higher design time and the amount of required hardware. Moreover, NoC offer more capabilities in controlling the flow of data in the network. However, every new capability consumes hardware. Therefore, the proposed NoC architecture (section 4.4.2) implements only a basic functionality, which results in low hardware cost (see table 4.3), and requires from 9% to 12% of additional hardware for the codec design proposed, as shown in table 4.5. The author actively participated in the designing and debugging of modules of the proposed codec, which is reported in section 4.5.

The conclusions presented drove the author to a further research on NoC design. The author found the lack of suitable tools that would offer hardware codec application modeling and simulation of NoC, as shown in 5.2. The results of the analysis of previous work and own research

resulted in the proposal of an NoC simulator that implements the key aspects of advanced video coding modeling, shown in 5.3. These take into consideration:

- the type of data sent between modules,
- conditional probability.

The work on the improvement of simulator accuracy proved that application modeling as proposed by [**?**] is, in the case of advanced video codes, insufficient. The above-listed aspects of application modeling turned out to be crucial in the implementation of a simulator traffic model.

For the proposed simulator, the accuracy is about 10% of the obtained value of an average macroblock processing time (see section 5.5). Moreover, the simulator gives macroblock processing time distribution compliant to the values obtained from a hardware decoder allowing for a detailed analysis of the frequency of short and long processed macroblock times. These results are presented in section 5.6.

The results obtained encouraged the author to further generalize the modeling of a video decoder. The goal was to roughly estimate the simulation results with the use of queue modeling. Statistical distributions that are utilized in the simulator are employed to obtain an expected value of distribution. The results of these calculations are then used in queue modeling. The author chose the simplest queue model (i.e., the M/M/1) available, because it does not assume rejecting a packet from a queue. The accuracy of such modeling is presented in section 6.3. Such modeling is useful to estimate the delay introduced by each module, as well as the delay introduced by a path that e.g., communicates with the memory. Such an application of the proposal is presented in section 6.4.

A simulation tool was exploited to compare the performance of different topology proposals to choose the optimal structure for an AVC decoder. The results are gathered in chapter VII. Performance differences between the topologies explored turn out to be small and sometimes statistically insignificant. Therefore, other factors to choose the optimal architecture need to be considered. Section 7.4 discusses two properties of the proposed topologies: hardware consumption and the selection of operating frequency and efficiency of an NoC structure proposed in section 4.4.2.

8.2 Original achievements of the dissertation

The **original achievement** of the dissertation is a proposal (as a development team member) of an NoC architecture, which is a design achievement that offers a high performance rate at a low hardware cost. The proposal is based on the ring topology and employs network interfaces with switching capability to reduce network consumption. Routers are used to separate traffic and to offer buffering between parts of the hardware application. The proposed architecture is beneficial especially for small applications such as video codecs, since it introduces NoC capabilities at a cost comparable to a shared bus connection.

The author also **proved** the applicability of NoC for video codecs on an exemplary realization of an AVC codec despite the small size of such an application. The benefits obtained concern a simplification of the design and verification. The NoC scalability allows for further development of the decoder application to enrich it with encoding capabilities. Moreover, connection-oriented design eased the debugging process and separated the functional design from communication infrastructure, allowing for the verification of modules based on the correctness of the input and output signals.

The **important achievement** of the dissertation is the proposal of modeling a scheme of advanced video codecs for NoC simulators. The simplified model allows for the simulation and analysis of performance results for any topology and a wide class of video processing applications. The author provided exemplary modeling results for an AVC decoder. The simulator was exploited to explore different topology proposals. Although the experiments showed similar results for different structures, the results indicate the benefits of traffic separation even for small designs, such as an AVC decoder.

Another **original achievement** is the proposal of the estimation simulation results with the use of simple calculations. For this purpose queue modeling was employed and showed the results allowing for a rough assessment of system performance. The adoption of queue modeling was proposed as a tool for pre-simulation analysis, however, not as a simulation replacement. It allows for the estimation of a delay introduced by different modules, as well as on delaying paths. The proposal gives an opportunity to adjust capabilities of Intellectual Property cores (IP cores) at their selection stage, to identify possible bottlenecks of the application.

8.3 General conclusions

The dissertation showed the applicability of NoC for video codecs. The benefits and limitations can be generalized for a variety of applications, especially the ones with functionalities that are planned to be expanded (e.g., from decoder to encoder). The analysis is based on hardware implementation of an AVC encoder which has implications for a wide spectrum of applications, that include implementations of new standards of video coding, transcoders, and video analysis tools. Each of the mentioned applications can be modeled with the use of the proposed modeling tools and simulated with the use of the proposed simulation. The traffic modeling proposed, can be employed for any application type and is especially suitable for modules of complex data exchange characteristics.

The simulation results showed similar system performance for different architectures, which proves that NoC is characterized with a capacity that exceeds that application's requirements. Nevertheless, the results obtained indicate that the best effects are achieved for the NoC architecture that offers traffic separation between parts of the application. They also show that communication infrastructure for hardware realizations consisting of about 10 modules requires the separation of traffic.

The author also showed applicability of queue modeling for hardware applications and its utilization for delay estimation. The presented calculations can be applied to any hardware application with the Poisson distribution of call arrival and exponential service times. However, the results obtained indicate that even if an application cannot be characterized with the above-mentioned parameters, some other more suitable queue models can be applied for the analysis. The dissertation showed the usefulness of such modeling.

APPENDICES

APPENDIX A

Simulator accuracy

Table A.1 shows results for *basket* sequence. All results present narrow confidence interval which is mostly below 1% and 2 cases with below 1.2%. The largest relative differences can be obsereved for frame I and $QuantizationParameter$(QP) = 25, 30, 35 with values: 11%, 12%, and 10% respectively. All other cases present less than 10% relative difference and average relative difference is below 7%.

Discrepancies of average processing time for *city* are smaller than for the *basket* (see table A.2). The highest difference is for frame I with QP = 25 which is equal 11%, other differences are lower than 10%.

Average processing time highest discrepancy values (table A.3) for *football* sequence are for low qp values in case of frame I and for high qp values in case of frame P. Although the highest discrepancy is 13% (frame I, QP = 25) most values are below 10%.

Table A.4 shows comparison of average processing time of one macroblock for simulation and real hardware decoder for *sunflower* sequence. The only one value (frame I QP = 15) exceeds 10% level and is equal 114%. Confidence intervals are below 1.6%.

Table A.1: Simulation accuracy for basket sequence - average processing time of one macroblock

Frame	QP	Simulation results [cycles]	Confidence interval [%]	Hardware results [cycles]	Relative difference [%]
I	15	8830	0.37	8375	5.43
I	20	7707	0.42	7100	8.54
I	25	6678	0.3	6013	11.07
I	30	5624	0.42	5013	12.21
I	35	4536	0.49	4118	10.17
I	40	3433	0.67	3317	3.51
I	45	2557	0.83	2590	1.27
P	15	5576	0.64	5926	5.89
P	20	3985	0.57	4223	5.61
P	25	2816	1.04	2972	5.23
P	30	1949	0.97	2068	5.74
P	35	1520	0.49	1622	6.25
P	40	1313	0.5	1435	8.51
P	45	1252	1.16	1363	8.11
B	15	5961	0.3	6455	7.65
B	20	4226	0.59	4586	7.84
B	25	2986	0.88	3264	8.49
B	30	2063	2.03	2341	11.87
B	35	1706	1.16	1907	10.5
B	40	1574	0.38	1722	8.58
B	45	1518	0.84	1634	7.06

Table A.2: Results accuracy for city sequence - average processing time of one macroblock

Frame	QP	Simulation results [cycles]	Confidence interval [%]	Hardware results [cycles]	Relative difference [%]
I	15	8110	0.33	7727	4.96
I	20	6895	0.33	6350	8.58
I	25	5718	0.26	5162	10.78
I	30	4624	0.16	4233	9.25
I	35	3613	0.41	3500	3.22
I	40	2720	0.65	2843	4.31
I	45	1990	0.58	2113	5.83
P	15	6091	0.38	6462	5.73
P	20	4104	0.49	4348	5.59
P	25	2516	0.5	2662	5.47
P	30	1581	0.33	1669	5.25
P	35	1277	0.33	1415	9.72
P	40	1243	1.05	1354	8.17
P	45	1236	1.88	1333	7.24
B	15	6347	0.07	6853	7.38
B	20	4453	0.39	4804	7.29
B	25	2917	0.54	3201	8.86
B	30	1750	0.96	2019	13.29
B	35	1527	0.46	1690	9.57
B	40	1477	1.13	1616	8.59
B	45	1474	0.73	1588	7.16

Table A.3: Results accuracy for football sequence - average processing time of one macroblock

Frame	QP	Simulation results [cycles]	Confidence interval [%]	Hardware results [cycles]	Relative difference [%]
I	15	6909	1.07	6268	10.23
I	20	5747	0.4	5081	13.12
I	25	4776	0.22	4211	13.43
I	30	3761	0.87	3427	9.76
I	35	2707	0.86	2656	1.95
I	40	2100	0.61	2101	0.04
I	45	1719	0.76	1767	2.7
P	15	4926	0.76	5231	5.82
P	20	3537	0.28	3734	5.25
P	25	2636	1.29	2768	4.75
P	30	1987	0.91	2102	5.44
P	35	1616	0.94	1708	5.37
P	40	1317	1.1	1492	11.67
P	45	1248	1.35	1388	10.07
B	15	5346	0.46	5726	6.63
B	20	4191	0.26	4477	6.38
B	25	3237	1.16	3476	6.86
B	30	2429	1.47	2626	7.5
B	35	1868	0.68	2120	11.86
B	40	1631	1.03	1815	10.13
B	45	1468	1.1	1644	10.64

Table A.4: Results accuracy for sunflower sequence - average processing time of one macroblock

Frame	QP	Simulation results [cycles]	Confidence interval [%]	Hardware results [cycles]	Relative difference [%]
I	15	7250	0.46	6512	11.34
I	20	6221	0.39	5670	9.71
I	25	5298	0.23	4868	8.85
I	30	4234	0.37	3953	7.11
I	35	3119	0.47	3052	2.21
I	40	2308	0.57	2356	2.03
I	45	1777	1.58	1906	6.76
P	15	3162	0.97	3338	5.27
P	20	2200	0.94	2316	5.0
P	25	1747	1.18	1834	4.72
P	30	1460	0.41	1556	6.17
P	35	1329	0.89	1412	5.83
P	40	1227	1.22	1347	8.9
P	45	1230	1.43	1327	7.28
B	15	3710	0.94	3991	7.03
B	20	2473	1.24	2623	5.7
B	25	2020	0.86	2189	7.7
B	30	1441	1.25	1902	24.22
B	35	1448	0.69	1759	17.67
B	40	1400	1.02	1666	15.94
B	45	1456	0.53	1616	9.88

APPENDIX B

Operating clock frequency

Table B.1: Results for operating frequency derived from HDL simulation

Frame	QP	*basket*		*city*		*football*		*sunflower*	
		Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]
I	15	333	9	307	5	248	3	257	6
I	20	282	10	253	5	201	3	224	5
I	25	239	8	206	3	167	2	192	6
I	30	199	6	169	3	136	3	156	6
I	35	164	5	140	2	106	3	120	6
I	40	132	3	114	2	84	2	93	4
I	45	102	3	85	2	71	1	75	2
P	15	234	13	257	9	208	5	127	18
P	20	167	15	173	10	149	4	88	12
P	25	118	13	105	11	111	3	70	6
P	30	82	7	66	4	84	2	62	2
P	35	65	2	56	0.9	68	2	56	1
P	40	57	0.8	54	0.3	60	1	54	0.6
P	45	54	0.3	53	0.2	55	0.4	53	0.3
B	15	254	19	271	3	227	5	159	19
B	20	181	16	189	4	177	14	103	10
B	25	129	13	125	5	136	17	87	6
B	30	92	8	79	3	103	12	75	4
B	35	75	4	67	0.8	84	8	70	2
B	40	68	2	64	0.4	72	4	66	1
B	45	65	1	63	0.1	65	1	64	0.4

Table B.2: Results for operating frequency derived from simulation

Frame	QP	*basket*		*city*		*football*		*sunflower*	
		Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]	Freq [MHz]	CI [MHz]
I	15	385	1	354	1	301	3	316	1
I	20	336	1	301	1	251	1	271	1
I	25	291	0.9	250	0.7	209	0.5	231	0.5
I	30	246	1	202	0.3	164	1	185	0.7
I	35	198	0.9	158	0.6	118	1	136	0.6
I	40	150	1	119	0.8	92	0.5	101	0.6
I	45	112	0.9	87	0.5	75	0.6	78	1
P	15	243	2	266	1	215	2	138	1
P	20	174	1	179	0.9	155	0.4	96	0.9
P	25	123	1	110	0.5	115	2	77	0.9
P	30	85	0.8	69	0.2	87	0.8	64	0.3
P	35	67	0.3	56	0.2	71	0.6	58	0.5
P	40	58	0.3	55	0.6	58	0.6	54	0.7
P	45	55	0.6	54	1	55	0.7	54	0.8
B	15	260	0.8	277	0.2	233	1	162	2
B	20	185	1	195	0.7	183	0.5	108	1
B	25	131	1	128	0.7	142	2	89	0.8
B	30	90	2	77	0.7	106	2	63	0.8
B	35	75	0.9	67	0.3	82	0.6	64	0.4
B	40	69	0.3	65	0.7	72	0.7	61	0.6
B	45	67	0.6	65	0.5	64	0.7	64	0.3

Table B.3: Results for operating frequency with respect to decoded frame buffer size in case of storing frames: I, P, B, B, B - hardware results

Frame	QP	Freq I [MHz]	Freq P [MHz]	Freq B [MHz]	Freq [MHz]
basket	15	333	234	254	266
basket	20	282	167	181	198
basket	25	239	118	129	149
basket	30	199	82	92	112
basket	35	164	65	75	91
basket	40	132	57	68	79
basket	45	102	54	65	70
city	15	307	257	271	275
city	20	253	173	189	198
city	25	206	105	125	137
city	30	169	66	79	94
city	35	140	56	67	79
city	40	114	54	64	72
city	45	85	53	63	67
football	15	248	208	227	227
football	20	201	149	177	176
football	25	167	111	136	137
football	30	136	84	103	106
football	35	106	68	84	85
football	40	84	60	72	72
football	45	71	55	62	64
sunflower	15	257	127	159	172
sunflower	20	224	88	103	124
sunflower	25	192	70	87	104
sunflower	30	156	61	75	88
sunflower	35	120	56	70	77
sunflower	40	93	54	66	69
sunflower	45	75	53	64	64

Table B.4: Results for operating frequency with respect to decoded frame buffer size in case of storing frames: I, P, B, B, B - simulation results

Frame	QP	Freq I [MHz]	Freq P [MHz]	Freq B [MHz]	Freq [MHz]
basket	15	385	243	260	282
basket	20	336	174	185	213
basket	25	291	123	131	161
basket	30	246	85	90	120
basket	35	198	67	75	98
basket	40	150	58	69	83
basket	45	112	55	67	73
city	15	354	266	277	290
city	20	301	179	195	213
city	25	250	110	128	148
city	30	202	69	77	100
city	35	158	56	67	83
city	40	119	55	65	74
city	45	87	54	65	67
football	15	301	215	233	243
football	20	251	155	183	191
football	25	209	115	142	150
football	30	164	87	106	114
football	35	118	71	82	87
football	40	92	58	72	73
football	45	75	55	64	65
sunflower	15	316	138	162	188
sunflower	20	271	96	108	139
sunflower	25	231	77	89	115
sunflower	30	185	64	63	88
sunflower	35	136	58	64	77
sunflower	40	101	54	61	68
sunflower	45	78	54	64	65

APPENDIX C

Accuracy of queue system calculations for four test sequences

Table C.1: Queue system calculations accuracy for basket sequence

Frame	QP	Simulation results [cycles]	Confidence interval [cycles]	Queue system results [cycles]	Difference [cycles]	Relative difference [%]
I	15	8833	13	9502	669	8
I	20	7703	16	8002	299	4
I	25	6681	12	6818	137	2
I	30	5622	9	5542	80	1
I	35	4526	8	4590	64	1
I	40	3429	10	3707	278	8
I	45	2554	9	3079	525	21
P	15	5577	10	6028	451	8
P	20	3983	16	4391	408	10
P	25	2817	11	3163	346	12
P	30	1954	8	2273	319	16
P	35	1524	8	1786	262	17
P	40	1315	7	1509	194	15
P	45	1253	6	1409	156	12
B	15	5962	18	6522	560	9
B	20	4226	25	4689	463	11
B	25	2986	26	3509	523	18
B	30	2063	42	2510	447	22
B	35	1706	20	2022	316	19
B	40	1574	6	1853	279	18
B	45	1519	13	1774	255	17

Table C.2: Queue system calculations accuracy for city sequence

Frame	QP	Simulation results [cycles]	Confidence interval [cycles]	Queue system results [cycles]	Difference [cycles]	Relative difference [%]
I	15	8125	11	8661	536	7
I	20	6891	10	7076	185	3
I	25	5722	8	5688	34	1
I	30	4621	6	4648	27	1
I	35	3617	10	3920	303	8
I	40	2728	10	3424	696	26
I	45	1990	5	2579	589	30
P	15	6092	7	6455	363	6
P	20	4106	11	4478	372	9
P	25	2519	8	2873	354	14
P	30	1576	5	1889	313	20
P	35	1275	5	1452	177	14
P	40	1248	6	1410	162	13
P	45	1234	8	1388	154	13
B	15	6347	5	6795	448	7
B	20	4454	17	4954	500	11
B	25	2917	16	3406	489	17
B	30	1750	17	2089	339	19
B	35	1528	7	1807	279	18
B	40	1477	17	1738	261	18
B	45	1475	11	1716	241	16

Table C.3: Queue system calculations accuracy for football sequence

Frame	QP	Simulation results [cycles]	Confidence interval [cycles]	Queue system results [cycles]	Difference [cycles]	Relative difference [%]
I	15	6912	20	7098	186	3
I	20	5732	15	5702	30	1
I	25	4781	6	4799	18	0
I	30	3750	11	3995	245	7
I	35	2714	11	3077	363	13
I	40	2100	6	2515	415	20
I	45	1719	5	1582	137	8
P	15	4927	13	5484	557	11
P	20	3528	10	3963	435	12
P	25	2624	11	3039	415	16
P	30	1992	8	2354	362	18
P	35	1610	6	1955	345	21
P	40	1320	6	1492	172	13
P	45	1248	5	1391	143	11
B	15	5347	25	6310	963	18
B	20	4191	11	4923	732	17
B	25	3237	38	3787	550	17
B	30	2429	36	2998	569	23
B	35	1869	13	2287	418	22
B	40	1631	17	1937	306	19
B	45	1469	16	1773	304	21

Table C.4: Queue system calculations accuracy for sunflower sequence

Frame	QP	Simulation results [cycles]	Confidence interval [cycles]	Queue system results [cycles]	Difference [cycles]	Relative difference [%]
I	15	7237	13	7446	209	3
I	20	6209	12	6242	33	1
I	25	5296	10	5472	176	3
I	30	4238	11	4463	225	5
I	35	3123	10	3416	293	9
I	40	2304	8	2666	362	16
I	45	1778	9	2223	445	25
P	15	3172	13	3585	413	13
P	20	2206	11	2596	390	18
P	25	1742	9	2044	302	17
P	30	1464	4	1721	257	18
P	35	1327	6	1532	205	15
P	40	1236	6	1389	153	12
P	45	1227	6	1374	147	12
B	15	3710	35	4398	688	19
B	20	2474	31	3084	610	25
B	25	2020	17	2650	630	31
B	30	1441	18	1734	293	20
B	35	1448	10	1737	289	20
B	40	1400	14	1674	274	20
B	45	1457	8	1711	254	17

APPENDIX D

Estimation of accuracy of queue system for the multi-scene sequence

Table D.1: Queue system calculations' accuracy for multi-scene video sequence sequence

Frame	QP	Simulation results [cycles]	Confidence interval [cycles]	Queue system results [cycles]	Difference [cycles]	Relative difference [%]
I	15	7595	36	6956	639	8
I	20	6345	22	5706	638	10
I	25	5230	19	4724	506	10
I	30	4241	19	3963	278	7
I	35	3418	28	3301	118	3
I	40	2867	25	2667	200	7
I	45	2264	23	2288	24	1
P	15	4345	29	5242	896	21
P	20	3341	18	3738	397	12
P	25	2572	16	2782	211	8
P	30	1876	33	2158	283	15
P	35	1515	20	1672	157	10
P	40	1310	18	1383	72	6
P	45	1254	15	1308	55	4
B	15	5384	33	5790	406	8
B	20	3849	34	4213	365	9
B	25	2855	19	3229	374	13
B	30	2047	23	2358	311	15
B	35	1673	17	1850	178	11
B	40	1563	13	1703	139	9
B	45	1492	17	1632	140	9

BIBLIOGRAPHY

BIBLIOGRAPHY