DISSERTATION

submitted to the
Combined Faculties for the Natural Sciences and for Mathematics

of the
Ruprecht-Karls-Universität
Heidelberg, Germany

for the degree of
Doctor of Natural Sciences

presented by
**Dipl.-Phys. Stefan Philipp**
born in Hannover, Germany

Date of oral examination: July 2, 2008

# Design and Implementation of a Multi-Class Network Architecture for Hardware Neural Networks

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. Ulrich Rückert

## Entwurf und Implementierung einer mehrklassigen Netzwerkarchitektur für Neuronale Netze in Hardware

Die vorliegende Arbeit beschreibt den Entwurf und die Implementierung einer Netzwerkarchitektur, welche Techniken von leitungsvermittelnden und paketvermittelnden Netzwerken verbindet, um zwei verschiedene Dienstgüten anzubieten: isochrone Verbindungen und paketbasierte Verbindungen mit bestmöglicher Zustellung. Isochrone Verbindungen verwenden reservierte Netzwerkresourcen, um eine verlustfreie Übertragung sowie eine niedrige Ende-zu-Ende Verzögerung mit begrenzter Varianz zu garantieren. Die Synchronisierung aller Netzwerkknoten sowie die Berechnung einer kompakten Reservierungsbelegung werden durch effiziente Algorithmen gelöst. Paketbasierte Übertragungen verwenden die verbleibende Bandbreite. Das Multiplexen beider Verkehrsklassen wird von einem neuartigen Bypass-Switch geleistet, der skalierbar ist in der Anzahl der Schnittstellen sowie in der externen Bandbreite und ohne eine interne Beschleunigung auskommt. Die Netzwerkarchitektur kommt in der Forschung innerhalb des FACETS Projektes mit großskaligen künstlichen neuronalen Netzen in Hardware zum Einsatz, für die Vernetzung eines verteilten Systems aus VLSI neuronalen Netzen. Axonale Verbindungen zwischen Neuronen werden mit Hilfe von isochronen Verbindungen modelliert, wohingegen paketbasierte Übertragung die Grundlage für eine systemweite gemeinsame Speicherarchitektur bildet. Der zur Laufzeit ausgeführte Teil des Netzwerkes ist in programmierbarer Logik implementiert und arbeitet mit einer externen Übertragungsrate von 3.125 Gbit/s. Die Arbeit diskutiert die anwendungsbezogenen Anforderungen an das Netzwerk, sowie dessen Entwurf und Referenzimplementierung in programmierbarer Logik und Software. Theoretische Überlegungen über die Leistungsfähigkeit werden durch Messungen und Simulationen verifiziert. Obwohl die Netzwerkarchitektur für die spezielle Anwendung mit neuronalen Netzen entworfen wurde, stellt sie eine generelle Lösung für alle Netzwerkumgebungen dar, welche isochrone Verbindungen und Paketvermittlung mit niedriger Komplexität benötigen. Die Architektur ist insbesondere für den Einsatz in der nächsten Stufe der Hardwareentwicklung des FACETS Projektes zur Vernetzung künstlicher neuronaler Netze auf Wafer-Ebene geeignet.

## Design and Implementation of a Multi-Class Network Architecture for Hardware Neural Networks

This thesis describes the design and implementation of a network architecture that combines techniques from circuit switching and packet switching to provide two different service classes: isochronous connections and best-effort packet transfers. Isochronous connections use reserved resources to provide loss-less transmissions as well as a low end-to-end delay with bounded jitter. Synchronization of all network nodes as well as computation of a compact reservation scheme is achieved by means of efficient algorithms. Best-effort packet transfers use the remaining bandwidth. Both traffic classes are multiplexed by a novel Bypass-Switch architecture, which is scalable in terms of port numbers and line speed and does not require internal speedup. The network architecture is employed within the experimental framework of the FACETS project for research into large-scale hardware neural networks, for which it interconnects a distributed set of VLSI neural networks. Isochronous connections model axonal inter-neuron connections whereas best-effort packet transfers are the basis for a framework-wide shared memory subsystem. The online part of the network architecture is implemented within programmable logic and operates at external line rates of 3.125 Gbit/s. The thesis discusses the service requirements of this kind of application, the design and the reference implementation of the network architecture in programmable logic and software. Theoretical results about the provided services are verified by means of measurements and simulations. Although implemented for a specific application, the developed network architecture is a general solution for all network environments that require isochronous connections and packet processing with low online complexity. It is particularly suitable for use within the next stage of the hardware development within the FACETS project for waver-scale interconnection of hardware neural networks.

# Contents

**Introduction** **1**

**Chapter 1 - Computer Networks** **5**
1.1 Network Topologies . . . 5
1.2 Reference Models . . . 6
1.2.1 The OSI Reference Model . . . 7
1.2.2 Alternative Models . . . 10
1.2.3 Encapsulation of Data Formats . . . 10
1.3 Circuit Switching and Packet Switching . . . 11
1.4 Quality of Service . . . 12
1.4.1 Services Guaranteed by the Network . . . 13
1.4.2 Techniques to Provide QoS . . . 14
1.5 Packet Switching Architectures . . . 16
1.5.1 General Architecture of a Packet Switch . . . 16
1.5.2 Output-Queued Crossbar Switches . . . 17
1.5.3 Input-Queued Crossbar Switches . . . 18
1.5.4 Combined Input-Output Queued Switches . . . 19
1.5.5 Further Switch Architectures . . . 20
1.6 Queuing Schedulers . . . 20
1.6.1 Schedulers to Access a Single Resource . . . 21
1.6.2 Crossbar Schedulers . . . 23
1.7 Summary . . . 27

**Chapter 2 - Framework Description** **29**
2.1 Artificial Neural Network ASICs . . . 29
2.1.1 The HAGEN Chip . . . 30
2.1.2 The Spikey Chip . . . 34
2.2 The FACETS Stage 1 Framework . . . 39
2.2.1 Nathan Network Module . . . 39
2.2.2 Backplane and Control PC . . . 42
2.2.3 Connectivity . . . 42
2.2.4 SlowControl and PowerPC Operation . . . 44
2.3 Neural Network Experiments . . . 46
2.3.1 Experimental Setups . . . 46
2.3.2 Interconnecting Multiple ANN Chips . . . 48
2.3.3 Neuron Mapping . . . 49

2.4 The Transport Network . . . . . 54
2.4.1 Design Considerations . . . . . 54
2.4.2 Transport of Neural Network Data . . . . . 55
2.4.3 Transport of Non-Neural Network Data . . . . . 59
2.4.4 Summary of the Service Requirements . . . . . 60
2.4.5 Existing Solutions . . . . . 61
2.4.6 Concept of the Transport Network . . . . . 62
2.5 Summary . . . . . 63

**Chapter 3 - The Multi-Class Gigabit Network Architecture 65**
3.1 Overview . . . . . 68
3.1.1 Merging of Traffic Classes . . . . . 68
3.1.2 Network Protocol Stack . . . . . 69
3.2 Framing Strategy . . . . . 70
3.2.1 Network Topology . . . . . 70
3.2.2 Formal Description . . . . . 71
3.2.3 Framing of Bandwidth . . . . . 72
3.3 Network Initialization Phase . . . . . 73
3.3.1 Parameter Selection . . . . . 74
3.4 Service for Isochronous Connections . . . . . 75
3.4.1 Model of the Isochronous Switch . . . . . 76
3.4.2 Contention Resolution . . . . . 76
3.4.3 Synchronization . . . . . 79
3.4.4 Resource Reservation and Connection Mapping . . . . . 80
3.4.5 Online Forwarding Process . . . . . 81
3.4.6 Local Port Interface . . . . . 82
3.5 Global Synchronization . . . . . 83
3.5.1 Services Provided . . . . . 83
3.5.2 Setup Process . . . . . 84
3.5.3 Overview . . . . . 85
3.5.4 Timing Scheme of the Switch . . . . . 87
3.5.5 Time Counter Adjustment . . . . . 88
3.5.6 Frame Alignment and Frame Size . . . . . 90
3.5.7 Synchronization Result . . . . . 98
3.5.8 Upper-Layer Synchronization Service . . . . . 98
3.6 Connection Mapping . . . . . 101
3.6.1 Algorithm Overview . . . . . 102
3.6.2 Bandwidth Quantization . . . . . 104
3.6.3 Connection Routing . . . . . 104
3.6.4 Slot Assignment . . . . . 105
3.6.5 Overall Algorithm Result . . . . . 110
3.6.6 Further Remarks . . . . . 110
3.7 QoS Results for Isochronous Connections . . . . . 111
3.7.1 Throughput and Drop Rate . . . . . 111
3.7.2 Reliability . . . . . 111
3.7.3 Delay . . . . . 112
3.7.4 Jitter . . . . . 114

3.7.5 Summary . . . 119
3.8 Service for Packet-Based Transports . . . 120
3.8.1 Packet Embedding . . . 120
3.8.2 Packet Format . . . 121
3.8.3 The Bypass-Switch . . . 122
3.8.4 Interface to Upper Layers . . . 126
3.8.5 Packet Routing . . . 128
3.9 Scalability and Complexity . . . 130
3.9.1 Space Complexity . . . 130
3.9.2 Time Complexity . . . 130
3.10 Summary . . . 131
3.10.1 Future Work . . . 132

**Chapter 4 - Implementation of the Transport Network 135**
4.1 Overview . . . 136
4.2 Framing and Packet Encoding . . . 138
4.2.1 Format of a Data Frame . . . 138
4.2.2 Format of a Best-Effort Packet . . . 139
4.2.3 Notification of the Slot Usage . . . 140
4.3 The Physical Layer . . . 141
4.3.1 Network Topology . . . 141
4.3.2 Distribution of the Global Reference Clock . . . 142
4.3.3 The Multi-Gigabit Transceiver . . . 144
4.3.4 Configuration of the MGTs . . . 144
4.4 The Synchronization Sublayer . . . 147
4.4.1 Timing of the Network Node . . . 148
4.4.2 Reception of Data . . . 149
4.4.3 Transmission of Data . . . 150
4.4.4 Selection of the Synchronization Parameters . . . 151
4.4.5 Global Synchronous Signals . . . 155
4.5 Implementation of the Bypass-Switch . . . 156
4.5.1 Characterization . . . 157
4.5.2 Overview of the Switch . . . 158
4.5.3 Implementation of the Input Buffers . . . 160
4.5.4 Implementation of the Switch Core . . . 161
4.5.5 Implementation of the Central Crossbar . . . 163
4.5.6 Interface to the Best-Effort Scheduler . . . 165
4.5.7 Interface to Upper Network Layers . . . 165
4.6 Implementation of the Best-Effort Scheduler . . . 166
4.6.1 Implementation of the iSLIP Scheduler . . . 167
4.6.2 Implementation of Two-Dimensional Schedulers . . . 171
4.6.3 Summary . . . 174
4.7 Routing of Best-Effort Packets . . . 175
4.7.1 Description of the Implemented Algorithm . . . 176
4.7.2 Summary . . . 177
4.8 Transport of Neural Network Data . . . 178
4.8.1 Provided Transport Service for Neural Network Data . . . 178

4.8.2 Demonstrator Application for Isochronous Transfers . . . . . 181
4.9 Distributed Shared Memory . . . . . 183
4.9.1 Overview . . . . . 184
4.9.2 Functional Description . . . . . 185
4.9.3 The Client Process and the User Interface . . . . . 187
4.9.4 The Server Process . . . . . 188
4.9.5 The DSM Packet Adaptation Layer . . . . . 189
4.9.6 Transport Control Protocol . . . . . 190
4.9.7 DSM Performance . . . . . 193
4.9.8 Summary . . . . . 197
4.10 Software Development . . . . . 197
4.10.1 Synchronization . . . . . 198
4.10.2 Connection Mapping . . . . . 200
4.10.3 Configuration of the Routing Tables . . . . . 203
4.10.4 Generation of Pseudo-Random Networks . . . . . 204
4.10.5 High-Level Simulation of the Packet-Switch incl. Scheduler . 205
4.11 Summary . . . . . 208

**Chapter 5 - Evaluation** **211**
5.1 Evaluation of the Physical Layer . . . . . 211
5.1.1 Measurement of the Data Reliability . . . . . 211
5.2 Evaluation of the Synchronization Sublayer . . . . . 213
5.2.1 Measurement of the Transmission Delays . . . . . 213
5.2.2 Establishment of the Synchronization . . . . . 214
5.3 Verification of the Transport of Isochronous Data . . . . . 216
5.3.1 Measurement of Application-Layer Delays . . . . . 216
5.3.2 Verification of Isochronous Transfers . . . . . 218
5.4 Discussion of the Neural Network Topologies . . . . . 220
5.4.1 Characterization of Neural Network Topologies . . . . . 221
5.4.2 Calculations for Neural Networks on the Backplane . . . . . 223
5.4.3 Homogeneous Pseudo-Random Networks . . . . . 224
5.4.4 Modified Pseudo-Random Networks . . . . . 226
5.4.5 Summary of the Evaluated Network Topologies . . . . . 233
5.5 Evaluation of the Connection Mapping Algorithm . . . . . 233
5.5.1 Evaluated Qualifiers . . . . . 234
5.5.2 Mapping Results of Pseudo-Random Networks . . . . . 236
5.5.3 Mapping Results of Networks with non-Intrinsic Hop Ratios . 237
5.5.4 Mapping Results of Alternative Hardware Topologies . . . . . 239
5.5.5 Summary . . . . . 243
5.6 High-Level Simulation of the Neural Data Transport . . . . . 243
5.7 Performance of the Best-Effort Schedulers . . . . . 246
5.8 Summary . . . . . 250

**Conclusion and Outlook** **253**

**List of Acronyms** **259**

**List of Symbols** **263**

**List of Figures** **267**

**List of Tables** **270**

**Appendix 1 - Resource Consumption of FPGA Implementation** **271**

**Bibliography** **284**

# Introduction

The human brain is one of the most complex and powerful information processing systems. It is able to solve computational tasks that are far beyond the possibilities of artificial computational systems and still consumes only about 20 Watts of power. Better understanding of this highly efficient system has been a research goal since ancient times. Although the basic principles of its functionality are being researched intensively, a comprehensive understanding of its operation is still not existent. Moreover, no artificial system built by humans has ever reached a level of complexity that is comparable to that of the human brain.

The study of the brain has been an inspiration for the development of computational neuron models and artificial neural networks (ANNs). A first important but rather simple model that uses a threshold function over the sum of binary input values has been developed in 1943 by W. McCulloch and W. Pitts [83]. An important step forward has been made by Rosenblatt in 1958 [119], who introduced the *perceptron* network that consists of neurons that use continuous input and output values and a continuous transfer function. Both models are rather abstract and computational than a correct image of a biological neural network. One of the most important progresses has been made in 1952 by A. L. Hodgkin and A. F. Huxley, who developed a neuron model closer to biology [52]. It is oriented along the physiological processes within the cells and includes the generation and propagation of action potentials, or spikes between the neurons. The model of Hodgkin and Huxley formulates the influence of voltage-gated ion channels, leak channels and ion pumps through the cell membrane on the electric potential with differential equations.

To get a better understanding of neural information processing, it is required to combine a larger number of neurons to an ANN. Doing so, the computing time to solve the differential equations becomes the limiting factor for the size of the modeled networks. Especially if plasticity, diversity and temporal development are part of the model, the available computing power will be quickly insufficient to explore the timescales involved [124]. In contrast to the human nervous system, where 100 billions of neurons and about $10^{16}$ synapses operate in parallel in continuous time, a software simulation reaches complexities in the order of $10^3$ neurons in real-time with a simple Hodgkin-Huxley-based integrate-and-fire model on the fastest available microprocessors [97]. One solution to overcome this issue is to use a powerful parallel computer [80].

Progress in microelectronics makes it possible to implement physical neuron models using very large scale integration (VLSI) circuits with complementary metal oxide semiconductor (CMOS) technology to exploit the parallel nature of a neural network. In a physical model, important physiological quantities like the membrane potential

are assigned an equivalent physical quantity. The variables of the differential equations of the neuron models are represented by electrical counterparts like voltages, currents and capacitances. The implementation of spiking neuron models in VLSI hardware has been done by several scientific groups. The work in [34] implemented Hodgkin-Huxley neurons in hardware. The group in [50] implemented spiking neurons with a time-dependent learning rule. Both designs are limited to a small number of neurons per chip due to the complexity of the model.

A different approach has been taken by the Electronic Vision(s) research group in Heidelberg in the course of the European projects *SenseMaker* [128] and its successor *FACETS* [89]. Two chips have been developed, whose VLSI designs use a less complex neuron model, with the benefit of several hundreds of artificial neurons per chip in a power-efficient design. The chip *HAGEN* [123] provides 256 perceptron-based neurons and the more recent chip *Spikey* [124, 122, 47] comprises 384 leaky integrate-and-fire neurons at an average acceleration factor of about $10^4$ compared to biology. The synaptic interconnections of the neurons are established on-chip within configurable matrices, allowing a wide range of network topologies to be investigated on a single device. The Spikey chip provides plasticity as *spike time dependent plasticity (STDP)* within each individual synapse. Although both chips use different neuron models, they follow the same key idea: combining the advantages of analog VLSI techniques for the implementation of the neural circuits with the advantages of digital communication between the chips. The HAGEN chip has been used for a wide range of experiments including pattern recognition [39, 38], classification experiments based on evolutionary algorithms [55, 54, 53, 125] and the adaptation of liquid computing [127].

The pure digital interface of the developed chips makes it possible to extend the implemented networks beyond chip boundaries. Multiple chips can be interconnected to large-scale neural networks with several thousands of neurons. Connections between neurons on different chips then correspond to axonal connections of biological neurons. The physical infrastructure for this setup is provided by *Stage 1* of the FACETS project, which consists of backplanes equipped with network modules each hosting an ANN chip [46, 39]. The framework requires a *transport network* to establish the neural interconnects, for which it provides programmable logic. Since the data to be transferred is digital, the transport network may use techniques and protocols of existing computer networks to perform this task.

The challenge in designing the transport network is not only a required low online complexity, but also the large number of inter-neuron connections with strict service requirements on the timing and on the throughput of the transmitted neural data. Or to keep to the terms of computer networks: A large-scale ANN requires certain guarantees of *quality of service (QoS)*. In particular, the operation of the chips relies on *isochronous connections* with a low and nearly constant connection delay and a guaranteed throughput due to the biological nature of the connections.

Besides the pure neural data, the experiments to be carried out require additional on-demand transfers for configuration data like synapse parameters, neuron parameters or the transport of network stimuli and monitored neural behavior. These data cannot be transferred within isochronous connections due to the latency and the complexity of their setup. Since a pre-reservation of permanent connections would lead to a bandwidth waste, a packet-based approach is more suitable.

Although a wide range of network architectures have been published, only few of them provide isochronous connections combined with best-effort packet transfers and have only low online complexity. Well-established network architectures like asynchronous transfer mode (ATM) [143] are far too complex to be implemented within the limited space of the programmable logic of the framework. Packet switching approaches seem to be more suitable, but there is currently no switch design or scheduler design that provides a small enough end-to-end delay and jitter for a neural network application combined with low online complexity. Isochronous network architectures have been proposed [159, 40], but do not guarantee the throughput of the reserved bandwidth in all cases. Combined architectures using slotted timing have been presented for networks-on-chips (NoCs) in [35, 117], but the first design is based on asynchronous logic and does not scale, whereas the latter does not provide the required synchronization and slot allocation algorithms for a distributed setup like the Stage 1 framework.

This thesis proposes a network architecture that combines techniques of circuit switching and packet switching to fulfill these requirements. A reference implementation of the architecture operates as the transport network of the FACETS Stage 1 framework for research into large-scale ANNs. Moreover, the network architecture is scalable to be used in the succeeding Stage 2 hardware of the FACETS project [121] for wafer-scale integration of hardware neural networks. Finally, the application of the network architecture is not limited to hardware neural networks, but is a general solution for all environments with demands for isochronous connections and packet-based transfers for which a compact implementation in programmable logic is needed.

### Thesis Overview

The thesis is organized as follows: The first chapter gives a brief survey of the basic principles of computer networks with focus on the provision of end-to-end QoS in packet switching networks. The second chapter presents the Stage 1 framework of the Electronic Vision(s) research group in Heidelberg. It concludes with a list of service requirements on the transport network according to the ANN application. Chapter three introduces the novel network architecture *multi-class gigabit network (MCGN)* and describes the framing strategy, the synchronization concept, the resource reservation algorithm as well as the traffic class multiplexing in detail. Chapter four describes the implementation of the transport network within the Stage 1 framework, which uses the reference implementation of MCGN for its lower network layers. Chapter five covers the evaluation and the verification of the implemented services by means of simulations and measurements. It is further discussed how the properties of different neural network topologies influence the possible average spike frequencies of the artificial neurons according to the bandwidth provided.

# Chapter 1

# Computer Networks

---

*The network architecture presented in this thesis has been developed to operate within a distributed setup of hardware neural networks in an embedded environment. However, its design techniques are borrowed from computer networks. This chapter introduces the basic terms, concepts and protocols of computer networks. The description starts with network topologies and reference models and introduces the concept of network layers. The switching techniques circuit switching and packet switching are described. Since the transport of neural network data requires guaranteed throughput and a low and constant transmission delay, the chapter continues with the description of QoS. The provision of QoS is described with focus on the design of packet switches and scheduling policies. The limited size of the thesis does not allow a detailed discussion of the complex matter. The descriptions are therefore given briefly and the interested reader may refer to [143, 110].*

---

## 1.1 Network Topologies

The nodes of a network can be divided into end-nodes and interconnecting nodes. The end-nodes are called hosts, whereas the interconnected nodes are called routers or switches (see below). Each node of a network is individually addressable via its dedicated network address. The network to be considered can be divided into several subnets. Subnets that contain multiple hosts are called local area networks (LANs). The LANs can be of a different network type and may have their own addressing format for the hosts. Figure 1.1 shows a network of 16 hosts, which are interconnected via a single subnet.

Different network topologies are conceivable: ring, bus, star, cubic, tree, mesh etc. The particular topologies require different methods for the access of the medium

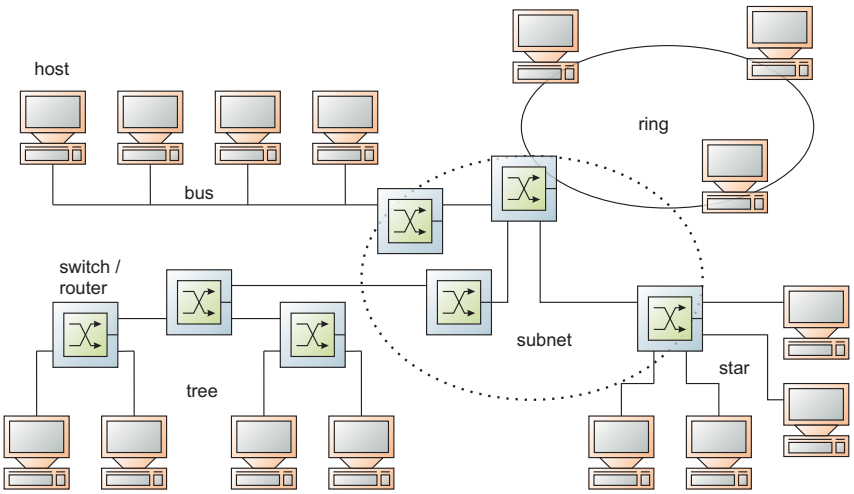



Figure 1.1: Example network of 16 hosts organized in 4 local area networks of different topology. The hosts are interconnected via switches and routers.

and have different trade-offs for the access latency, the throughput per host or the medium costs.

## 1.2 Reference Models

*network layers and nomenclature*

The different functionalities of networks are usually organized into several *network layers* to reduce the design complexity. The layers are ordered by abstraction. The lowest network layer performs the access to the physical transmission medium. Each network layer contains a certain set of *processes*, which provide a set of *services* to the layer above and use the services from the layer below. The services define the functionalities the network layer provides and not how they are implemented. The communication between two stacked layers is performed via the corresponding *interface* between it. The interface defines the the basic operations to use the provided services. The communication between processes at different locations on the same layer is called a *protocol*. The entities at the different locations that communicate via a protocol are called *peers*. Figure 1.2 shows a schematic of a network layer protocol.

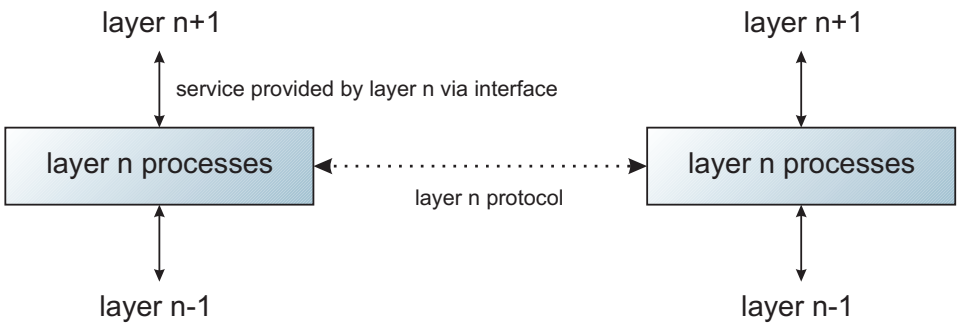



Figure 1.2: A network layer protocol provides a service to its layer above.

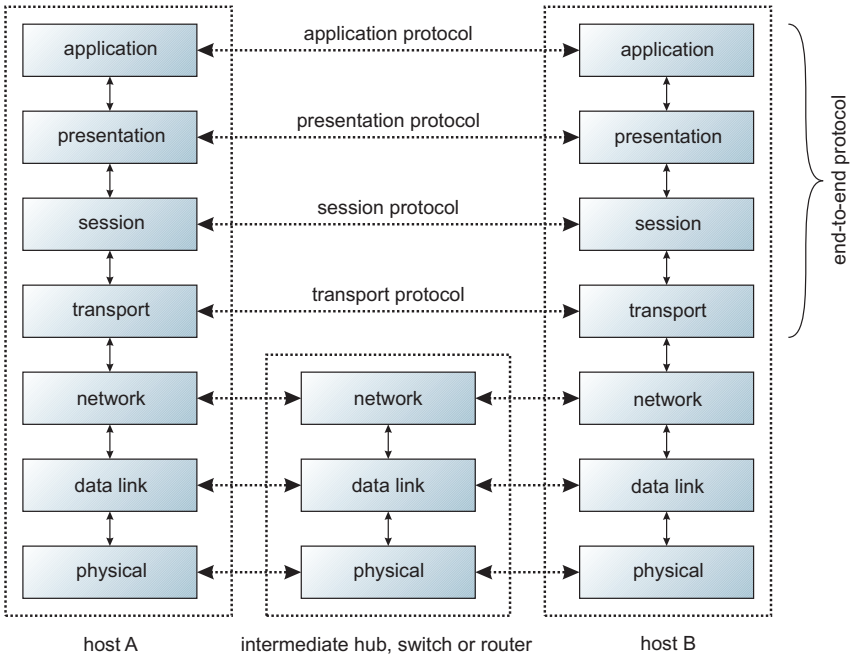



Figure 1.3: The seven-layer OSI reference model

Different reference models exist that define the purpose of the protocols and their basic functionalities within the layers. However, the implementation of certain functionalities is often not strictly limited to a particular layer. As an example, the tasks flow control, error detection or encryption are useful within both lower and higher network layers.

### 1.2.1 The OSI Reference Model

An important and general model to be discussed is the open systems interconnection (OSI) reference model [163], which is shown in figure 1.3. It consists of seven layers with the physical layer being the lowest layer and the application layer on top.

#### The Physical Layer

*electrical and mechanical definitions*

The physical layer accesses the physical medium to perform the transmissions between adjacent network nodes. The physical medium is conceptually located below the physical layer. The physical layer therefore defines mechanical and electrical standards, e.g. how the bits of the data streams are encoded, the pinouts of connectors, the voltage levels of electrical signals or the used transmission frequencies. Example processes of the physical layer are the data encoding and the data decoding.

*network extensions*

Repeaters or hubs can be used to extent the size of a physical medium. Both devices operate basically within the physical layer. Repeaters refresh incoming signals on their way to the destination to enhance the transmission distance, whereas hubs are basically multi-port repeaters and reflect the frames between all ports to interconnect multiple nodes.

### The Data Link Layer

*node-to-node frame delivery*

The purpose of the data link layer is to ensure a reliable communication between adjacent network nodes within the local subnet. For this reason, the data link layer groups the bits that are transmitted via the physical layer to *frames*. The data link layer ensures that the particular frames are unambiguously detectable within the physical bit stream and adds functionalities for the error detection or also error correction. The provided service of the data link layer therefore is to transmit or to receive single frames to adjacent network nodes. Furthermore, the data link layer ensures that a fast node is not flooding a slow node with data. This is called *flow control*.

*media access control*

Another important task is the arbitration of the access to the physical medium in the case that it is shared with other network nodes e.g. via a single cable, a ring or simply the radio frequencies in the case of wireless networks. This task is located within the media access control (MAC) sublayer, which is part of the data link layer. Example MAC sublayer protocols are Ethernet [92] or Token Ring [145].

*frame forwarding, switching*

The interconnection of subnets on the data link layer is done with bridges that also convert the frame formats. Network nodes that interconnect multiple other nodes or subnets are called *switches*. Switches have a larger number of physical ports and forward the frames between the ports according to their data link layer address. The main tasks of switches are multiplexing, queuing and scheduling. The design of packet switches is discussed in more detail in section 1.5.

### The Network Layer

*network-wide packet delivery*

The network layer provides a topology-independent view of the network to the layers above. It provides a network-wide communication between network nodes that are not necessarily directly connected and may even be located in interconnected subnets of different types. The data formats of the network layer are called *packets*. Packets contain network layer addresses.

*routing of packets*

The most important task of the network layer is the *routing*, which is to determine a path (or route) of consecutive adjacent network nodes from the source of a packet to its destination. One issue of this is to avoid live-locks and dead-locks, which means that packets are ensured to reach their destinations without circling or being blocked. The network nodes that perform this task are called *routers*. Routers operate with network layer addresses and are aware of (at least parts of) the network topology. This is in contrast to switches, which perform rather the forwarding of frames between adjacent nodes according to local addresses.

*processes of the network layer*

Besides the routing, further processes of the network layer are the fragmentation of packets into smaller ones to keep to the maximum packet size of the several subnets, as well as the re-assembly of the fragments to packets. An example network layer protocol is the internet protocol (IP) protocol [114] of the internet. Another task is the conversion between the network address and the data link layer addresses of the local subnets. An example protocol is the address resolution protocol (ARP) protocol [112], which is mostly used to translate addresses between IP and Ethernet.

### The Transport Layer

*end-to-end communication*

The transport layer provides multiple and independent end-to-end communication channels to the layers above. This end-to-communication can be either packets (now called *datagrams*) or *connections*. The mostly used transport layer protocols of the internet are user datagram protocol (UDP) [113] for datagrams and transmission control protocol/internet protocol (TCP/IP) [11] for connections.

*TCP/IP connections*

Concerning TCP/IP connections, the protocol hides the packet-based nature of the network layer to the layers above and provides reliable connections. The main tasks are the fragmentation of the data streams to packets and its reassembly at the destinations. Furthermore, the transport layer re-requests erroneous packets or lost packets and also ensures the correct packet order in the case that packets get out-of order due to different routes. The connections of TCP/IP have a client-server architecture. One process (the server) is awaiting connections on a dedicated port and another process (the client) may initiate a connection to that port. Usually, connections require a setup-phase and a take-down phase. This is handled by an internal state-machine which can be of significant complexity.

Another important task is to control the amount of packets sent into the network to avoid an overload of its resources (*congestion control*). This can be detected by the occurrence of packet drops even if the two peers of a connection feature sufficient resources. The transport layer protocol reduces the packet rates in that case.

### The Session Layer

*coordination of multiple connections*

The purpose of the session layer is to manage and coordinate multiple connections between different users or applications. The appearance of the session layer depends on the application, it can even be nonexistent. Its tasks are authentication (password checks), synchronization (e.g. the synchronization of audio streams and video steams within a video conference) or token management (preventing two users of using the same critical resource), but also the checkpointing and the restoration of data streams to resume a session after a crash.

### The Presentation Layer

*definition of transmitted data formats*

The presentation layer defines the semantics and the syntax of the data formates to be exchanged. This is useful to allow machines with different encoding techniques to communicate via a common protocol. The services of the presentation layer perform the conversion of data formats and structures on a higher abstract level and define the bit-patterns that are send via the lower layers. Other presentation layer issues are the encryption or the compression of the transmitted data.

### The Application Layer

*user defined*

The application layer finally contains the application-specific protocols and data formats. Examples for a application-layer protocols are hypertext transfer protocol (HTTP) to fetch web-sides or post office protocol version 3 (POP3) to get emails from the servers.

### 1.2.2 Alternative Models

The described OSI reference model is not the only existing model. As an advantage, it is well suited to understand the basic principles of networking. The model has been deviced before the corresponding protocols have been invented [143].

*TCP/IP reference model*

In fact, the TCP/IP reference model of [11] describes the requirements for internet hosts and its protocols are widely used. The protocol is not formally defined in any publication, but its layered architecture can be extracted from [11]. The TCP/IP model groups the lower layers in a single general network access layer and the upper three layers in a general application layer. The intermediate network and the transport layer use the protocols IP respectively UDP and TCP/IP.

*model used for the thesis*

For the remainder of this theses, the discussion of the network functionality keeps to the layered organization of networks and to the concept of interfaces, services and protocols. The discussion does not use a strict model, but follows the presentation in [143] and uses a hybrid model of the described two for simplicity. It uses the lower levels of the OSI model to describe the physical transmissions and the switching technology in conjunction with the simplifications of the TCP/IP model to group the top three layers to a single abstract application layer. This used hybrid model is illustrated in figure 1.4.

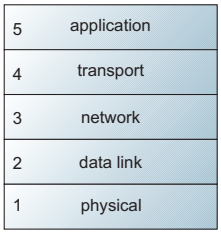



Figure 1.4: The hybrid network model used throughout this thesis.

### 1.2.3 Encapsulation of Data Formats

*headers and trailers*

Most protocols require to transmit a certain amount of data together with the user data from the layer above. This data is added as a header or a trailer to the user data at the source node and is removed at the destination. The user data from the layers above is called the *payload* for the lower layer protocol. This results in the fact that the data from the lower layer protocols is placed at the beginning and at the end of the frame, whereas the user data is enclosed by its lower layer protocols "within". Figure 1.5 illustrates a typical example of a data frame on an Ethernet LAN that belongs to TCP/IP session.

*transparent transportation process*

The encapsulation of the protocols is an important aspect since it makes the implementation of the lower layer transport processes completely transparent to the layers above. As an example, a transport layer protocol that communicates via datagrams does not have to care about *how* exactly the datagrams arrive at the destination. A switching architecture that provides general packet transports can therefore be used by all higher-layer protocols that require this kind of service. Nevertheless, the way packet transports are implemented clearly determines the service

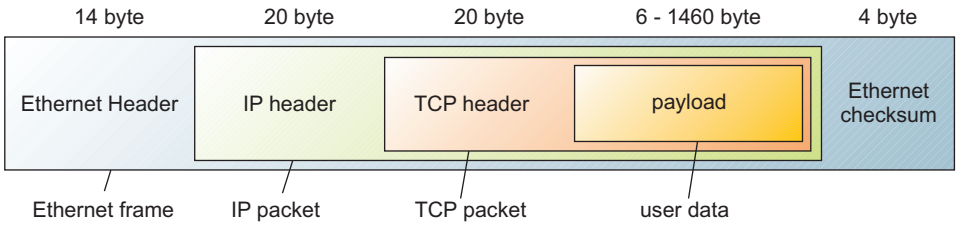



Figure 1.5: Example of an Ethernet data frame that contains a TCP/IP packet.

guarantees that are mad by the lower layers for the transportation process. This is investigated in the succeeding sections.

## 1.3 Circuit Switching and Packet Switching

Most computer networks are not fully connected, but the hosts are interconnected via intermediate switches using a switching technology. Switching reduces the number of physical lines and simplifies the interfacing of single hosts to the network. A fundamental design aspect of each network architecture is the question whether the network provides connection-oriented transport services or connection-less transport services to the layers above. The basic two alternatives are called circuit switching and packet switching.

### Circuit Switching Networks

*functional description*

In circuit switching networks, the communication is performed along dedicated paths (or *circuits*) between the source node and the destination node of a connection. The circuit uses exclusively reserved resources and features a certain guaranteed throughput (or bit rate). A once established circuit remains active for the duration of its connection. The circuits have to be set up in a *signaling phase* or *call setup* before a communication can take place and have to be taken down after the communication completes. The forwarding decision for the data is easy, since the routing decisions have already been made during the signaling phase. This reduces the required amount of buffering within the routers and switches. Examples for circuit switched networks are the old-style telephone networks with the protocol integrated services digital network (ISDN) [9].

*properties*

The advantages of circuit switching is that it allows to reserve network bandwidth to the communicating hosts and that it reduces the complexity of the forwarding task of the data to be transmitted. The reservation of resources further results in a low overall end-to-end delay for the transmitted data. The drawbacks are that the required signaling phase is of high complexity and results in a latency before the transmissions of data starts. Furthermore, reserved but unused bandwidth is wasted, which makes circuit switching more useful in the case that the required bandwidth can be estimated. Circuit switching networks are also less robust, since a failure of a router results in a failure of all of its managed circuits.

### Packet Switching Networks

*functional description*

In packet switching networks, the data to be transported is split into *flows* of multiple packets, which are transported independently. The path or route of the packets is not pre-determined, so its up to each packet to find its way through the network. Packed switching networks therefore require a routing decision to be made for each packet at each intermediate network node. For this reason, each packet contains an additional *header* with the destinations address and usually further administrative information. Since the bandwidth of the packets is not pre-reserved, the network may be required to buffer the packets at the routers and switches before the output lines became free (*store-and-forward* packet switching). An example for a packet switching network is an IP network that connects multiple Ethernet LANs.

*properties*

The advantage of packet switching networks is that they require less administrative overhead by the network, since no connections between end-nodes have to be set up or taken down. Furthermore, packet switching networks feature increased robustness, since the packets can be routed around a failed router along different routes and are also better designed to handle unpredictable traffic. The drawback of packet switching networks is that the packet headers can lead to a waste of bandwidth especially for small payloads. Furthermore, they require a higher administrative overhead by the end-nodes, e.g. for the re-ordering of packets that arrive out-of-order due to different routes. As another drawback, the non-deterministic forwarding of packets in packet switching networks makes it harder to provide deterministic service guarantees.

### Combined Switching Techniques

*transparent to user*

The terms circuit switching and packet switching concern the operations within the lower network layers. The layered protocol stack makes the switching architecture of the network transparent to the user and allows to provide both, connections-oriented services as well as connection-less (datagram) services to upper network layers. Packet flows that belong to higher-layer connections can be transported via both switching techniques. As an example, an ATM [143] network uses fixed-sized packets (or *cells*) that are transmitted along *virtual circuits (VCs)* to provide both, connection-oriented services and also datagram services. Another example is the transport of IP packets over the telephone system or making telephone calls over the internet.

## 1.4 Quality of Service

*definition*

The term QoSs denotes the provision of traffic characteristics or priorities by the network for different levels of performance to the transported data flows. The provided services are usually guaranteed according to a request by the application and thus differ from the general methods to improve the network performance such as flow control or congestion control. Example requirements are:

- A guaranteed reliability of the transmitted data.
- A guaranteed throughput or more specifically, a bit rate.

- Guaranteed bounds for the transmission delay.
- Guaranteed bounds for the delay variation (the *jitter*).
- A guaranteed maximum cell loss ratio.

QoS requests from applications may vary in a wide range. For some applications, a low latency of the traffic is more important than the data reliability. For others, the data reliability is essential and a high throughput is desired etc. Table 1.1 shows an example list of applications and its requirements of QoS [143].

| application | reliability | delay | jitter | bandwidth |
|---|---|---|---|---|
| email | high | low | low | low |
| file transfer | high | low | low | medium |
| web access | high | medium | low | medium |
| remote login | high | medium | medium | low |
| audio on demand | low | low | high | medium |
| video on demand | low | low | high | high |
| telephony | low | high | high | low |
| videoconferencing | low | high | high | high |

Table 1.1: Example applications and how stringent its QoS requirements are [143].

### 1.4.1 Services Guaranteed by the Network

A network does usually not support detailed and fine-grained requests for all combinations of the above variables. In most cases, the network provides a set of traffic classes or service classes to the application. Some networks provide hard guarantees to certain parameters and others provide only soft QoS services. Example end-to-end QoS service classes are:

- *Guaranteed services*, also known as hard QoS. This can be guaranteed bandwidth or guaranteed end-to-end delay or jitter. The network guarantees these services to particular flows independent on the behavior of other flows. This usually requires the absolute reservation of network resources. Examples for guaranteed services are:

  *guarantees services*

  - A *guaranteed throughput*. This can be achieved using a WFQ scheduling algorithm within the switches (see below).
  - A *constant bit rate (CBR)* service. This is required e.g. for telephony, audio, video or other real-time applications, where not the absolute bandwidth, but the smoothness of the traffic is important to avoid gaps. It is e.g. provided by ATM networks [143].
  - *Isochronous connections*. The term isochronous refers to a constant delay between the transmission of the data at the sender and its reception at the receiver. A constant delay ideally equals no jitter. In fact, isochronous

connections are practically achieved with CBR services and the isochronicity of the connections depends on the remaining jitter. Isochronous connections are useful for real-time applications. Example networks that provide isochronous connections are [137, 40].

*best-effort services*

- *Best-effort* services. No QoS is guaranteed and the packets of a best-effort flow simply use the remaining resources not reserved or not used by other traffic classes. This is best characterized by first-in first-out (FIFO) queues, which have no differentiation between flows. The network usually tries to schedule best-effort traffic in a fair way, such that no flow is starved out completely by other best-effort flows.

*integrated services*

- *Integrated services* is a technique to provide fine-grained QoS on IP networks. The QoS guarantees are made per flow. The management of the several flows is done by the network with the resource reservation protocol (RSVP) protocol [12, 151]. The applications have to describe the requested flow with a set of given QoS parameters. A request may be accepted or rejected. The drawbacks are that the complex RSVP protocol has to be implemented within each router of the network, and the required setup phase for each flow.

*differentiated services*

- *Differentiated services* is a technique to provide coarse-grained QoS on IP networks [101, 7]. The QoS guarantees are class-based with pre-defined classes by the network. The guarantees are made statistically and no hard guarantees are made for individual flows. As an advantages to integrated services, this greatly simplifies the network organization and does not require a flow setup phase. Example traffic classes are *expedited forwarding* [27] for prioritized, low-loss or low-latency traffic, *assured forwarding* [51] with 12 sub-classes of different priority and discard probability, and *best-effort* traffic.

It is an important design factor whether the network itself defines the available QoS features or the application may demand a fine-grained QoS with a complex parameter set. Furthermore, the number of service classes is important since it may require the routers and switches of the network to manage multiple queues. A simple implementation to provide QoS is the definition of few priority classes which are served in ascending order.

### 1.4.2 Techniques to Provide QoS

The different traffic classes for QoS are provided with different techniques. Practical solutions often combining several of them:

*error handling*

- The data reliability is guaranteed by *error detection* and/or *error correction* mechanisms. Often used methods are checksums, e.g. the cyclic redundancy check (CRC), which are used in a wide range of protocols on nearly all network layers. Packets (or data in general) that arrive with wrong checksums are simply dropped and re-transmitted. The detection of lost packets and the re-transmissions are usually implemented within the data link layer or the transport layer. Other mechanisms are the usage of redundant data in the case that re-transmissions are too costly.

- A simple possibility to meet throughput requests is the *overprovisioning* of bandwidth and buffer space. In many cases, its more easy to buy additional capacities than to change an existing network architecture to increase the performance. As an example, the telephone system is overprovisioned, since a phone call is rarely rejected. As another example, the development of the dense wavelength division multiplexing (DWDM) technique leads to enormous capacities in fiber-optics communication [110] for wide area networks (WANs). *overprovisioning*

- To reduce the jitter of a packet flow, its is possible to *buffer* the data streams. This can be done either at the receiver of a flow (by using a jitter or playback buffer), at the sender or at intermediate nodes (then called *traffic shaping*). Techniques for traffic shaping are e.g. the *leaky bucket* algorithm or the *token bucket* algorithm [143]. A traffic shaping algorithm controls the average rate of a data stream. It has the advantage that it reduces the burstiness of the traffic, which results in smaller buffer requirements and reduced congestion and packet drop probability at intermediate nodes. *buffering*

- In the case that a strict guaranteed QoS is required for a particular flow, the network has to use *resource reservation*. The resources like bandwidth, buffer space or central processing unit (CPU) cycles are exclusively reserved for the requesting flow and can be provided independently of the network traffic of other resources. Resource reservation can be achieved more easily in circuit switching networks, simply by establishing a circuit between the end-points of the flow with the guaranteed properties. The exclusive reservation of bandwidth can be achieved by a time division multiplexing (TDM) technique, which is a periodic division of the time into time slots. The reservation of time slots further requires a certain synchronization to keep to the correct slot position when passing data from one link to the next. Other possible reservation techniques are the assignment of wavelengths in optical networks or radio frequencies in wireless networks. *resource reservation*

- In packet switching networks, each packet is assigned to a certain flow, which is denoted within the packet header. The flow delimiters are respected by the intermediate packet switches of the network. Incoming packets are stored within the queues of the switches. The switches contain *packet schedulers*, which calculate the queues to be served next according to the flows the packets belong to. This is useful to implement different priorities or to balance the forwarding process of unreserved flows. Packet switches and scheduling algorithms are discussed in more detail in sections 1.5 and 1.6, respectively. *packet scheduling*

- A network that provides QoS in any case must be able to decide whether it can accept a new request for a QoS flow according to its available resources. This process is called *admission control*. The network has to take care to only accept new flows such that the currently accepted flows remain unaffected. For that reason, a request for a new flow has to be specified by the calling process in a *flow specification*. An example flow specification is used with RSVP [12, 151] and contains the token bucket rate, the token bucket size, the peak data rate and the minimum and the maximum packet size [143]. The *admission control*

maximum packet size may be limited for particular network protocols. As an example, an Ethernet frame is limited to 1500 byte payload.

The next sections considers a more practical view of how QoS can be implemented within the network.

## 1.5 Packet Switching Architectures

In circuit switching networks, the information of how to forward incoming data has been pre-determined during the signaling phase of the circuits and is available at the switches at the time the data arrives. Clearly, QoS can easily by provided in circuit switching networks by using the pre-reserved resources for bandwidth or buffering space. In contrast to that, the provision of QoS in packet switching networks is more complicated. This section describes the basic design principles of packet switches in packet switching networks.

*purpose*

The purpose of the switches is to forward the incoming packets to their destinations, which can be determined out of the destination address of the packets. The problem is that the switches may have no information in advance about the arriving packets, neither of their destination nor of their number. Besides the pure packet forwarding, a switch design has to respect the following aspects:

- The main goal is to achieve 100 % throughput. That is, no packet should be dropped due to overcrowded queues or contenting ports.
- The packet average delay, i.e. the average time the packet stays within the switch, should be minimized.
- No packet flow should be starved out. This has to be independent of the behavior of other flows.
- The design should be scalable in terms of area and line speed.
- The switch may provide to multicast packets from a single input to multiple outputs.
- The switch may provide deterministic QoS. As stated in section 1.4, QoS guarantees can be provided in terms of guaranteed bounds for the packet delay and throughput, but also by the provision of different service classes such that the packet flows are forwarded with different traffic characteristics (e.g. at different priorities).

### 1.5.1 General Architecture of a Packet Switch

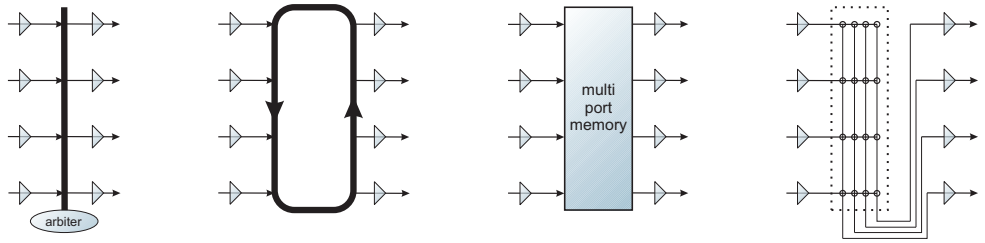
*basic operation*

The following discussion considers switches with the same number $N$ of input and output ports, for simplicity. During the operation of the network, packets arrive sequentially at the input ports and leave the switch sequentially at its output ports. The purpose of the switch is to forward the incoming packets internally from the inputs to the outputs. To handle the packet flows, switches usually consist of the following components:

- A multiplexer functionality. Possible architectures are a central bus, a central ring, a shared memory or a crossbar with $N^2$ crosspoints [148]. Multiple stages are possible.
- A set of buffers (or queues) to store the packets that cannot immediately be forwarded. One possibility is the usage of FIFO queues.
- A scheduling algorithm that decides the time each packet is served. This can be implemented with a centralized scheduler and also with multiple independent schedulers at each port.

*limited scalability of shared architectures*

Figure 1.6 illustrates basic switch architectures. The control of switches that use shared resources like a shared bus (a), a shared ring (b) or a shared memory (c) is less complex then switches that use crossbars (d) [148]. The drawback of the shared architectures is that the bandwidth of the shared medium has to be higher than the bandwidth of the external line. As an example, the central memory of a shared memory switch requires $2N$ times the bandwidth of the external line rate. Furthermore, the memory requires $2N$ independent ports, which increases its complexity. At higher port numbers and higher line rates, shared architectures have only limited scalability. The following discussion therefore focuses on crossbar switches only.




(a) shared bus (b) shared ring (c) shared memory (d) crossbar

Figure 1.6: Schematic overview of basic switch architectures. The input ports are shown on the left, the output ports on the right.

*crossbar architecture*

The fully-connected crossbar is non-blocking and buffer-less and transfers multiple packets in parallel while still operating at the external line rate. The crossbar and the queues may operate at higher rates than the external line rate for increased performance. This is denoted as an internal *speedup* $s > 1$. Different switch architectures of crossbar switches exist, which are mainly distinguished according to the locations of their queuing stages and the scheduling algorithm used.

*contention*

Crossbar switches have to solve the problem of *contention*. That is, each input port can forward data only to a single output port at a time and also each output port can take data only from up to a single input port. The switches have to solve this problem by using queuing stages at the input or output ports or by using a speedup to avoid data losses.

### 1.5.2 Output-Queued Crossbar Switches

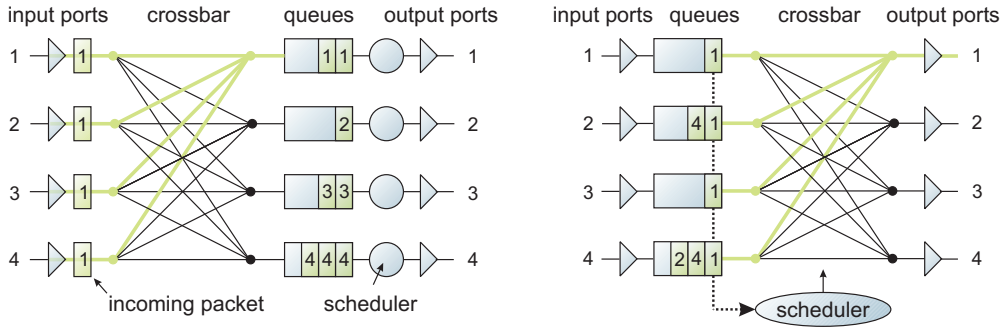
*basic architecture*

Figure 1.7(a) shows the basic design of an output-queued crossbar switch. At this

type of switch, incoming packets are directly forwarded to the crossbar and stored in the queues, which are located in front of the output ports. The queuing stage at each output can be a single FIFO queue or also multiple queues to sort the packet flows. It is also easy to implement multicasts by simply copying the incoming packets.

*limited scalability*

Output queued switches have been investigated in the earlier times due to its conceptual simplicity. Several scheduling algorithms can be used to provide QoS in such a switch (for an overview see e.g. [161]). The main drawback of output-queued switches is the required speedup of $N$ to handle contention at the output ports in the case that multiple packets that arrive at the same time at the input ports have to be forwarded to the same output port. Output-queues switches are therefore not scalable to higher line rates and larger port numbers.




(a) output-queued switch (b) input-queued switch

Figure 1.7: Simplified schematic of an input-queued and an output-queued switch. (a) Output-queued switches require $N$ times the memory bandwidth at the queues to handle worst-case packet arrivals. (b) Input-queued switches require a centralized scheduler for an optimal usage of the crossbar.

### 1.5.3 Input-Queued Crossbar Switches

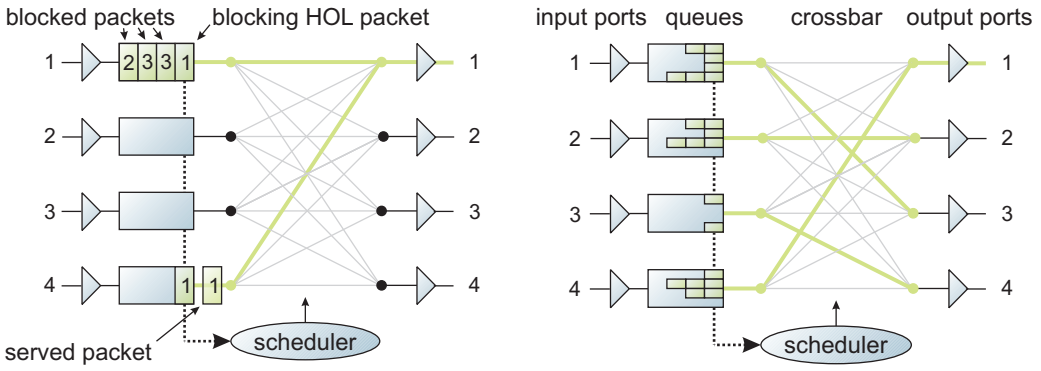
*basic architecture*

Figure 1.7(b) shows the basic design of an input-queued packet switch. Incoming packets are immediately stored within the queues and thus a speedup is not necessarily required. Since the queues of a single input may have packets for different outputs, contention occurs at both, the input ports and the output ports. Without an internal speedup, each output port can take data from only a single input at a time. For that reason, a separate and independent scheduler at each output port is not possible, since multiple output schedulers could select the same input. The operation of the crossbar is therefore controlled by a single scheduler. This is usually achieved with a slotted time and by the use of fixed sized packets (e.g. ATM *cells*). The crossbar then multiplexes up to $N$ packets in parallel from the input ports to the output ports within a single time slot. However, a stable scheduling of variable sized packets can achieved by modifying existing cell-based algorithms [81, 42].

*HOL blocking problem*

The problem of input-queued switches is that the architecture suffers from the head-of-line (HOL) blocking effect in the case that simple FIFO queues are used for the packet storage. The HOL effect is illustrated in figure 1.8(a). The effect occurs in the case that a single packet at the head of a FIFO queue cannot be forwarded to

its dedicated output due to other packets that are taken for this output, first. Due to the FIFO queuing policy, the packet at the head of the queue blocks the other packets even if packets behind the head could be transferred to other, currently unused, outputs. The work in [69] showed that this effect limits the throughput of such a switch to $2 - \sqrt{2} = 58.6\,\%$ for incoming uniform independent, identically distributed (i.i.d.) Bernoulli arrival patterns [1].




(a) head-of-line blocking (b) virtual-output queuing

Figure 1.8: Illustration of the head-of-line locking problem.

*virtual-output queuing*

One opportunity to solve the problem of HOL blocking is to use virtual output queues (VOQs), which contain a separate FIFO queue for each output at each input. The work in [142] presents different architectures for statically and dynamically allocated queues. The queues can have a single write port for incoming packets and a single read port to the crossbar and implement the multiple queues internally. The usage of VOQs requires incoming packets to be pre-routed at the time of their arrival to determine their output port and thus their corresponding VOQ. Since each internal queue can be scheduled individually, the HOL blocking problematic has been removed (cf. figure 1.8(b)).

*complex scheduler, QoS difficult*

As a drawback, this technique significantly complicates the central scheduler, which now has to calculate a scheduling out of $N \times N$ requests from the queues for the $N$ output ports. In particular, it is difficult to implement guaranteed services for an input-queued switch. This shows that it has to be made a trade-off between scalability and the provision of QoS. The output queued switch can provide QoS, but is not scalable. The input-queued switch architecture is scalable, but the provision of QoS is difficult. Example schedulers for input-queued switches with VOQs are described below in section 1.6.2.

### 1.5.4 Combined Input-Output Queued Switches

*OQ emulation with speedup of two*

Switches with queues at both, the input ports and the outputs are called combined input-output queuing (CIOQ) switches. They combine the effects of the two previous

[1] Uniform i.i.d. Bernoulli traffic comprises an arrival pattern such that in any given time slot, the probability that a packet arrives at a particular input is $p$. Each packet has the equal probability of $1/N$ of being addressed to any given output, and successive packets are independent.

architectures by accepting an increased amount of queuing space and an increased complexity of the scheduler. It has been shown in [25, 138] that a CIOQ switch with a speedup of two can emulate the behavior of an output-queued switch and thus provide guaranteed QoS.

### 1.5.5 Further Switch Architectures

There are many variations besides the described switch architectures. Variants feature additional buffers, multiple stages or distributed schedulers in contrast to centralized ones. Newer approaches for designs for high-speed switches are:

*load-balanced*

- *Load-balanced switches* use an additional, unbuffered load-balancing stage with an additional crossbar before the switching stage [20, 21, 72]. The load-balancing stage distributes incoming traffic periodically over the central buffers. This results in a uniform arrival pattern at the second stage. For uniform arrivals, the second stage can be scheduled in a periodic manner of $O(1)$ complexity and results in 100 % throughput, which effectively removes the need of a scheduler. As a drawback, this type of switche suffers from packet disordering [74, 22, 160] and requires an increased space for the second stage. This promising switch type is discussed further below.

*parallel switches*

- A *parallel packet switch* also uses multiple stages, but placed in parallel such that each packet experiences only a single stage of buffering. It consists of multiple packet switches in parallel that operate independently and at a lower rate compared to the external line rate [64]. It can be shown that such a switch can achieve 100 % throughput and can emulate an output queued switch with FCFS scheduling (see below), but it is difficult to implement QoS.

The above sections gave an intention of how the multiplexers and the queues are arranged in different switch architectures. Most of the architectures require a queuing scheduler to decide which packet to be served next. The next sections discusses how this scheduling task can be implemented.

## 1.6 Queuing Schedulers

*scheduling of queued packets*

The purpose of the queuing schedulers is to decide at which time the packets that are stored in the various queues are forwarded to the output ports of the switch for transmission. Concerning packet switches, the scheduling task can be discussed either with reference to the single queues or also with reference to the various packet flows that are stored within the queues. In general, the scheduler gets repeatedly requests from the queues (or the packet flows) and selects one of them. The general requirements for a queuing scheduler are (see e.g. [4]):

*requirements*

- To maximize the throughput.
- To minimize the average delay (the queuing wait time) of the packets.
- To feature fairness and protection. This means that all queues should be served fair under equal traffic conditions and also that no service for a queue should be influenced or even starved out by others.

- To be of limited complexity, i.e. to be simple and fast for an implementation in hardware at high line rates.

Further requirements concern requests to guaranteed services depending on the switch type or application and may include:

- The provision of guaranteed, deterministic QoS.
- A simple admission control to request the guaranteed services.
- To still maintain the fairness for non-prioritized flows.

A wide range of queuing schedulers exist for different switch architectures and also purposes. The next sections give a brief overview of schedulers often used and discuss their advantages and disadvantages.

### 1.6.1 Schedulers to Access a Single Resource

*select 1 out of N*

Schedulers for an arbitration of a single resource can be used in output-queued switches, where the packets are stored in queues directly at the outputs after having already been transferred over the crossbar. The schedulers can therefore be implemented separately and independently at each queue. The purpose of this type of scheduler is to repeatedly select a single queue or packet flow out of $N$ requesting queues (i.e. non-empty queues) to transmit a packet. Two simple scheduling policies are:

- One of the simplest schedulers uses the *first-come first-served (FCFS)* scheduling policy. All packet flows are stored within a single queue and are served within the order in which they arrived. It is simple to implement. *first-come first-serve*
- A commonly used scheduler is a *round-robin* scheduler (cf. figure 1.9). For each selection, all queues have different priorities in a cyclic order. The queue to be served next is selected out of the non-empties according to the current priorities. The priorities are then rotated such that the selected queue gets the least priority next. Round-robin is fair since no queue has to wait longer than $N-1$ selections and can be starved out by other queues. It performs well for comparable service requests from all queues. *round-robin*

A drawback of the above schedulers is that no service guarantees are made for particular queues (or packet flows). In contrast to that, the following schedulers can be used to provide guaranteed services. Most of them provide guaranteed rates [161, 4].

- The simplest way to guarantee the service for a particular queue is using a *static priority*, which serves each queue with a different priority. A particular queue is served only if all higher-priority queues are empty. The unfairness is obvious since only the top-priority queue receives a guaranteed service and packets in lower-prioritized queues can encounter significant delays. *static priority*
- A TDM scheduler selects each queue every $N$ time slots in a periodic manner. This guarantees each queue the fraction $1/N$ of service time. The drawback of this method is that a particular queue cannot get increased service times *TDM*

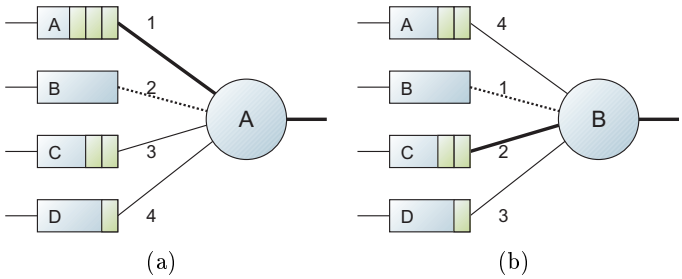



Figure 1.9: Illustration of the operation of a round-robin scheduler. The queues A,C,D have packets and request the scheduler. The current high-priority queue is denoted within the scheduler, depicted as a circle. (a) Since queue A has the highest priority, it gets the grant and transmits a packet. (b) Queue B now has the highest priority and queue C transmits a packet. For the next arbitration, queue D will have the highest priority.

even if all other queues will not request the scheduler. Nevertheless, for equal service request times, the TDM scheduler performs well.

*GPS*

- The generalized processor sharing (GPS) scheduler is a theoretical approach to provide fine-grained control for packet flows [104, 105]. With GPS it is possible to reserve bandwidth to several queues and to guarantee end-to-end delay bounds. Each packet flow has its own queue and an ill-behaving queue is not drowning others. Unfortunately, GPS is not implementable as it assumes infinitesimal service times.

*WRR*

- The weighted round-robin (WRR) scheduler is a simple approximation of GPS. The scheduler requires per-flow queuing. It visits each queue in a turn and the weighting impacts the number of packets released from each queue when it is visited. For equal weights, WRR operates like round robin. The number of packets to be transmitted from each queue is determined by dividing the weight of each queue by its average packet size. The problem is that this average weight has to be known in advance and which may lead to unfairness. WRR can provide guaranteed rates by setting the weight values accordingly. A hierarchical version of WRR is hierarchical round robin (HRR) (see e.g. [68, 58]).

*DRR*

- A comparable scheduler, the deficit round-robin (DRR) scheduler [131], uses a counter for each queue to regulate the flow. It does not need to know the mean packet sizes for each flow in advance. DRR serves a packet if its counter is greater than its size. Otherwise the counter is increased by some given value. The counter is decreased by the size of the packet being served. As a drawback, this scheduler does not allot fair bandwidth at short time scales.

*WFQ*

- The weighted fair queuing (WFQ) scheduler is a close approximation of the GPS [104, 30] scheme. The packets are queued according to their flows and scheduled with a weight. WFQ serves excess traffic in a fair manner according to the amount of bandwidth that has been reserved. The scheduler calculates finishing times to the packet flows that are weighed with the priority of the flows. The prioritized queues are then serviced in a bit-to-bit round-robin manner. The problem of this scheduler is its complexity.

- The earliest deadline first (EDF) scheduler uses a dynamic priority [36, 133]. A deadline is calculated individually for each packet at each switch according to its total delay and the required bandwidth. The priority of a packet increases during the time it waits in the queues. The scheduler selects the packet with the smallest deadline for transmission. This allows both, strict priorities but also a good performance for packets with loose service requirements. EDF performs better than GPS but has a high complexity. *EDF*
- The work in [44] proposed a general scheme, the *stop-and-go queuing*, to provide guaranteed throughput and bounded delay and jitter to packet flows. The technique is used for fixed-sized packets (cells), which are transmitted within equally-sized time frames of an integer multiple of a cell time in analog to TDM. An admission control policy at the source node limits the number of injected cells per flow. The queuing policy at each switch is to postpone the transmission of incoming cells to the next frame, which effectively hinders the generation of bursts (cf. figure 1.10). It is possible to implement multiple hierarchies with different frame sizes for different QoS guarantees. *stop-and-go queuing*

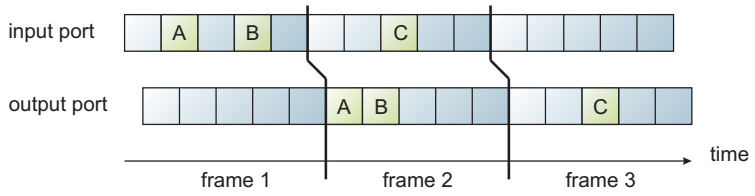


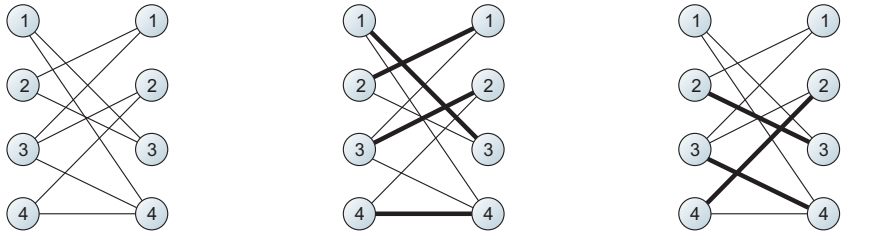

Figure 1.10: Illustration of the stop-and-go queuing discipline for a particular flow. Each incoming cell is postponed until the next frame time before transmission.

### 1.6.2 Crossbar Schedulers

In input queued-switches with VOQs, the scheduling problem is more complex. This is since input-queued switches do not necessarily have to use a speedup so that the queues and the crossbar can operate with the external line rate. The packets have to be transferred in parallel via the crossbar. The scheduler gets $N \times N$ requests from the $N$ input queues to the $N$ output queues. The problem equals the *bipartite matching* problem from graph theory. The scheduler has to find a matching between the sets of input ports and output ports. Each port has to be assigned at most one counterpart. The achieved throughput computes out of the number of matched port pairs (cf. figure 1.11(a)). *bipartite graph matching*

A matching with the highest number of matched ports is called a *maximum size matching (MSM)* (cf. figure 1.11(b)). The scheduling algorithm that finds the maximum matching is stable (and achieves 100 % throughput) for uniform i.i.d traffic but could lead to starvation (and instability) if the arrival patterns are not uniform [87, 85, 75] or to reduced throughput [91]. Furhemore, the algorithm is to complex to be implemented in hardware. The best known algorithm has $O(N^{2.5})$ complexity [56]. *maximum size matching*

Variations of MSM are *maximum weight matching (MWM)*, which assigns a weight to each input queue, and further variations of it, e.g. longest queue first *maximum weight matching*




(a) the matching problem (b) maximum matching (c) maximal matching

Figure 1.11: Illustration of the bipartite graph matching problem: (a) Requests from inputs on the left to outputs on the right are denoted with lines. Each port can only be connected once. (b) A maximum matching comprises the maximum possible number of matches. (c) A maximal matching cannot be improved without removing existing matches.

(LQF) [87], oldest cell fist (OCF) [90] and longest port first (LPF) [91] (for an overview see also [126]). The problem of these algorithms is their high complexity for an implementation in hardware and that they cannot provide deterministic guaranteed services for particular flows.

### Crossbar Schedulers with Parallel Matching

*parallel matching*

An intensively studied class of algorithms is based on a parallel matching [85]. The main idea is to use $2N$ independent single-output schedulers (or arbiters), one for each input and output port (cf. figure 1.12). The arbiters at the input ports are called *accept arbiters*, whereas the arbiters at the output ports are called *grant arbiters*. The scheduling is made in three steps: request, grant, accept.

1. *Request.* All input queues that have packets request all their corresponding grant arbiters at the outputs.
2. *Grant.* The grant arbiters select one out of the requesting inputs.
3. *Accept.* Since multiple inputs can be selected by the grant arbiters in the second step, the accept arbiters at the inputs select one of them.

The resulting matching consists of the input/output port combinations that have been accepted within the last step. The parallel matching can also easily be implemented with multiple iterations in which the currently unselected ports are matched successively. The schedulers find a maxim*al* matching instead of a maxim*um* one. A maximal matching is defined such that it cannot be improved without removing existing matches, which are made in earlier iterations (cf. figure 1.11(c)).

A crucial point of this algorithm is how the single grant and accept arbiters are implemented [85, 84]:

*PIM*

- A randomized selection is called *parallel iterative match (PIM)*. It finds a maximal matching in $O(log(N))$ iterations, but can lead to unfairness for certain flows.

*RRM*

- The algorithm that uses a round-robin selection is called *round robin matching (RRM)*. The round-robin scheme introduces fairness. As a drawback, this leads to a synchronization of the several grant arbiters at the outputs such

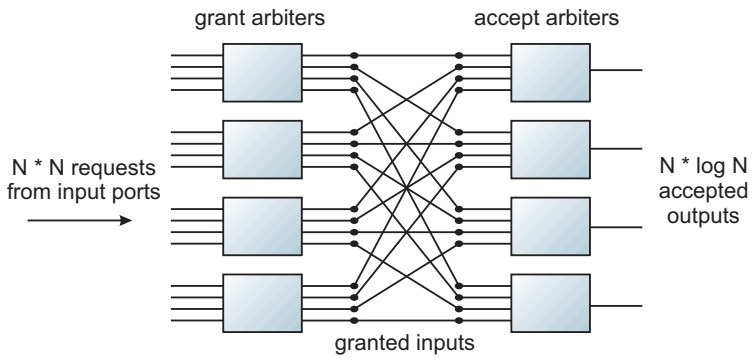



Figure 1.12: Illustration of a crossbar scheduler that performs a parallel matching. The inputs ports denote $N^2$ requests to the grant arbiters. The accept arbiters take the granted inputs and accept an output port for each input.

that different arbiters select the same input. Since the selected input can select only a single output, many possible connections are left unused.

- A very famous algorithm is named *iSLIP* [84, 86, 48]. It also uses round robin arbiters but with a modified update rule for the internal priorities of the grant arbiters: The priority of a grant arbiter is only updated if the granted input on his part accepts the granted output in the third step. This has a significant impact on the performance. The key is that the modified update rule results in a de-synchronization of the several arbiters such that a higher number of independent input/output pairs are matched with few iterations. Unfortunately, the iSLIP algorithm cannot achieve 100 % throughput when the traffic is non-uniform, or when arrivals are correlated. The algorithm has been implemented in hardware for high-speed switches [86] *iSLIP*
- A lot of improved versions of the iSLIP algorithm have been proposed in the literature. Some examples are prioritized iSLIP, weighted iSLIP (because the strict priority may lead to starvation of low priority flows), *FIRM* [129] with slightly improved performance, *static round robin (SRR)* [66], dual round-robin matching (DRRM) which is simpler and faster, or centralized multicast contention resolution (CMCR) [24] with multicast support. *further variants*

### Crossbar Schedulers with 2-Dimensional Cell Arrays

A different approach to implement a crossbar scheduler without requiring multiple iterations uses a 2-dimensional arrangement of elementary *arbiter cells* [141]. Each cell represents a possible input/output pair and gets a dedicated request signal for this pair for which it can denote a grant. To hinder the same input port or output port to be interconnected multiple times, each arbiter cells receives information whether its input or output port has already been selected by a different cell and forwards this information to other cells. As an advantage, this arbitration scheme finds a maximal matching within a single step. *principle*

Multiple types of this scheduler have been published which differ in the type of the basic arbiter cells, in the 2-dimensional arrangement, in the number of required cells and in its performance:

- The work in [141] published the wave front arbiter (WFA) and the wrapped wave front arbiter (WWFA) arbiters. The problem with this type of scheduler is that it requires the use of cyclic combinational logic and does not provide service guarantees. A VLSI implementation has been published in [141, 29].

*RPA, DPA*

- The work in [59] overcomes the logic loop by using an increased number of arbiters to extend the 2-dimensional arrangement. Fairness is introduced by enabling only parts of the cells, which results in different priorities for the particular input/output port combinations. The set of enabled cells is moved each scheduling in a periodic manner. The two presented arbiters rectilinear propagation arbiter (RPA) and diagonal propagation arbiter (DPA) differ in the number of required arbitration cells and the performance of the scheduler.

*PWWFA*

- The recent work in [103] parallelizes the operation of the WWFA scheduler for a faster execution time by accepting an increased amount of logic. The parallel wrapped wave front arbiter (PWWFA) scheduler has the same throughput and fairness properties as WWFA.

The above schedulers are examples of how to implement fairness between the particular port pairs when scheduling an input-queued crossbar switch. However, the main problem with such a switch is to limit the complexity of a scheduler that provides *guaranteed* QoS for particular packet flows, especially bounded end-to-end delay. This is discussed in the next section.

### Crossbar Schedulers with Service Guarantees

Since output-queued switches do not scale, a key issue is how to provide deterministic QoS guaranteed with other switch architectures. CIOQ switches can emulate an output-buffered switch but with the drawback of an increased switch size and a required speedup of two. For purely input-queued switches with VOQs, the situation is more difficult. The provision of deterministic QoS guarantees is indeed a key issue for this switch type:

*framing strategies*

The work in [130] applies the above stop-and-go queuing strategy to input-buffered switches with VOQs. The queues have to store full frames at each port. It is shown that under the condition that the incoming traffic does not overbook the available line rate of the input ports or the output ports, it is possible to re-sort *all* incoming frames such that all cells can be forwarded *without contention* at the input ports or at the output ports within the next frame time. The calculation of the sorting has to be done in advance such that the traffic matrix between the ports has to be known. As an advantage, this design provides guaranteed rates with bounded delay and jitter with a low online complexity. As a drawback, the cells are required to be hold a frame time per switch, which increases the absolute delay. Furthermore, the framing forces a trade-off to be made between the reservation granularity and the packet delay. A similar work based on framing using idling HRR scheduling has been presented in [58].

*Birkhoff-von Neumann scheduling*

Another approach has been presented in [18, 19]. The scheduler provides guaranteed QoS by not requiring a speedup nor the framing of the traffic. The scheduler uses a decomposition approach based on the work of Birkhoff [6] and von Neumann [99] to decompose the traffic matrix between the input and output ports into a series

of permutation matrices. The permutation matrices are then used to schedule the incoming traffic (e.g. by using a WFQ scheduler) without contention between the switch ports. The problem of this scheduler is the required pre-known arrival rates. Furthermore, the scheduler has a scalability problem since it requires $N^2$ permutation matrices and thus the sorting of their finishing times at runtime and the storage capacity. The decomposition has been improved later in [78, 73].

*load-balanced BvN switch*

The work in [20] proposes an input-queued switch that uses the above Birkhoff-von Neumann scheduling, but adds an additional load-balancing stage before the switch. The load balancing stage periodically distributes incoming packets to the center queuing stage. This results in a uniform arrival pattern for the second stage for which the same simple set of scheduling matrices can be used. The Switch has an online complexity of $O(1)$ and higher stability against bursts, but loses the rate guarantee feature and also packets may get out of sequence, which requires increased complexity for re-ordering [22, 74].

*multi-stage load-balanced BvN*

By adding additional queuing stages before and after the switch for per-flow queuing and re-sequencing, the switch keeps the packet order and is able to emulate the ideal FCFS output-buffered switch, but requires a high complexity [21]. By using a framing strategy, such a switch provides guaranteed rates and bounded end-to-end delays with $O(1)$ online complexity without speedup and conflict resolution [23]. However, this again results in a comparably large delay of multiple frame times.

To conclude, there is currently no packet switch architecture that provides guaranteed rates with low delay, low jitter and in-order arrivals with low online complexity and without requiring an internal speedup.

## 1.7 Summary

The chapter introduced the basic concepts of computer networks. Since the motivation of this thesis addresses the transport of neural network data with requirements for the throughput, the delay and the jitter, the focus has been on the provision of these service guarantees. The two basic switching techniques have been presented:

- Circuit switching features guaranteed throughput and constant bit rates by means of exclusively reserved paths between the network nodes. However, circuits require a complex call setup and control to be done by the network.
- Packet switching is more flexible but requires additional headers for the routing process. Moreover, providing guaranteed services like bounded end-to-end delay or in-order delivery is difficult and leads to large or complex designs.

The succeeding chapter first introduces the FACETS Stage 1 framework. The discussion of the available neural network chips `HAGEN` and `Spikey` leads to more specific service requirements for the data transport within the framework. The proposed network architecture of this thesis is presented in chapter 3.

# Chapter 2

# Framework Description

---

*This chapter describes the framework of the Electronic Vision(s) group that is used for the research with VLSI ANNs. The first section introduces the two neural network application specific integrated circuits (ASICs)* ***HAGEN*** *and* ***Spikey****. It discusses the technical data of the chips as well as their underlying neural network models. The chapter continues with the description of a Stage 1 framework, with focus on the* ***Nathan*** *network module, which hosts the ANN chips, and the backplane, which provides the physical interconnects. The following section discusses considerations for the interconnection of multiple ANN chips to a large-scale hardware neural network. The chapter closes with the formulation of requirements to a network protocol implemented in programmable logic that provides the necessary transport services.*

---

The following section first describes the two ANN chips `HAGEN` and `Spikey`. It is shown that besides their internal connectivity, both chips provide the possibility to be interconnected with others of their kind for the creation of large-scale neural networks that are physically distributed over multiple chips. This motivates the definition of requirements to the underlying interconnection network that transports the neural data between the chips. Although both chips require isochronous external connections for a deterministic operation of their network, the timing requirements of the `Spikey` chip are much stronger.

## 2.1 Artificial Neural Network ASICs

The ANN chips `HAGEN` and `Spikey` are the central part of the Stage 1 framework of the Electronic Vision(s) group. Both chips implement a number of neurons and synapses organized in multiple network blocks. The chips combine internal analog computation with external digital communication in *mixed-signal* VLSI designs.

| | |
|---|---|
| process features | 0.35 $\mu$m, 1 poly, 3 metal |
| die/core size | $4.1 \times 3\,\text{mm}^2$ / $3.6 \times 2.5\,\text{mm}^2$ |
| synapse size | $8.7 \times 12\ \mu\text{m}^2$ |
| blocks/neurons/synapses | 4 / 256 / 32768 |
| supply voltage | 3.3 V |
| network frequency $f_{net}$ | 50 MHz typ. |
| connections/s | 1.64 Teracps max. |
| weight update rate | 400 Megaweights/s max. |
| weight resolution | 10 bit (nominal) + sign |
| LVDS bus data transfer rate | 11.4 Gigabit/s max. |

Table 2.1: Nominal specifications of the HAGEN chip [123].

*analog computation*

The analog computation exploits the characteristics of the substrate of the chips for a highly integrated, fast and also power-efficient design. Both chips implement several hundreds of neurons and 32 thousand to 98 thousand individual synapses on a single die. The current-based neural computation allows an execution of their neural network models with a speedup of multiple orders of magnitude compared to biology while consuming only a few watts of power. The programmable synaptic connections provide the flexibility to investigate different neural network topologies within the same chip.

*digital communication*

The external digital communication of both chips allows for the usage of established digital communication techniques for a high-performance interface. Furthermore, digital external communication is less sensitive to noise and crosstalk than analog transmissions. The confinement of the input and output ports of the network blocks to binary values is therefore a key aspect for the scalability of the chips, which support the interconnection of multiple units to form a large-scale neural network in hardware. For that reason, each chip that is operated within the framework is interfaced exclusively by a commercial field programmable gate array (FPGA) that provides the inter-chip connections via a transport network. Since both neuron models and thus the resulting network models are different, both chips demand different QoS guarantees from the underlying transport network.

### 2.1.1 The HAGEN Chip

The Heidelberg AnaloG Evolvable Neural network (HAGEN) chip has been developed by Dr. Johannes Schemmel and Dr. Felix Schürmann of the Electronic Vision(s) group. The chip has been fabricated in a 0.35 $\mu m$ CMOS technology. Figure 2.1 shows a micro photograph of the chip. Table 2.1 lists nominal values concerning the VLSI implementation. For a more detailed description of its VLSI implementation see [123]. For scientific research based on the HAGEN chip refer to [53, 127, 38, 125]. The following paragraphs give a short summary and focus on the details that are relevant for this thesis.

*overview*

The HAGEN chip features 256 neurons and 32768 synapses organized in four equally-sized network blocks. A network block comprises a synaptic array of $128 \times 64$

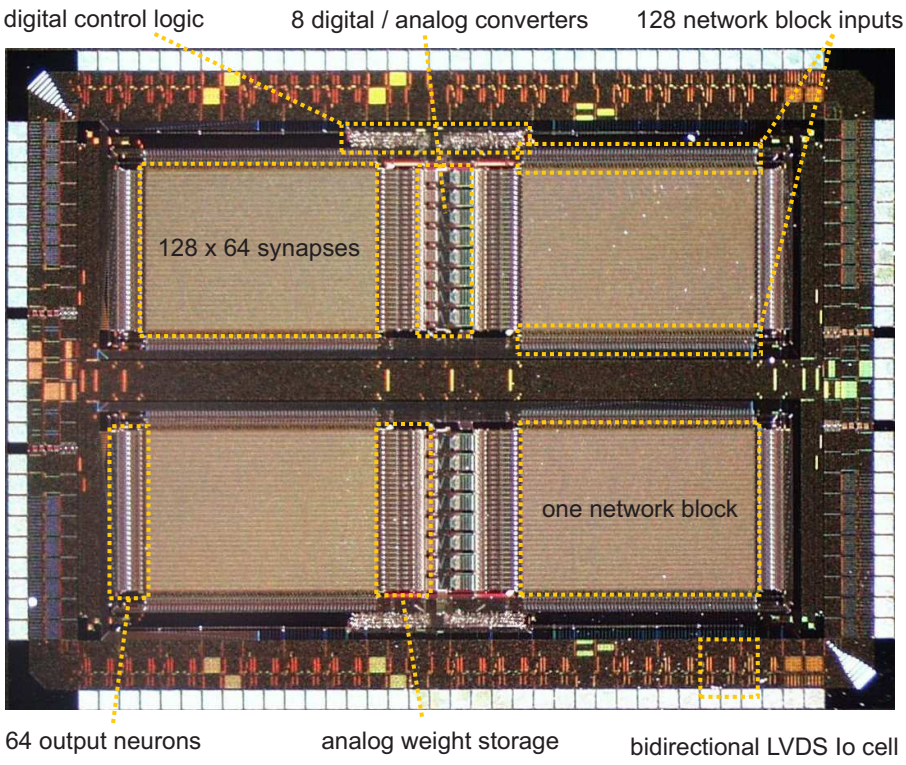



Figure 2.1: Photograph of the HAGEN chip

individual synapses, thus each output neuron takes data from up to 128 synaptic inputs. The analog signals are completely confined within the network blocks such that the communication between the blocks is purely digital. The external interface is required to transport data off-chip and on-chip, to configure the synaptic weights as well as to control the network operation of the chip. It features low voltage differential signaling (LVDS) cells [144] to support a data rate of up to 11.4 Gbit/s.

### Neuron Model

*perceptron*

The neural functionality of the `HAGEN` chip is based on the Perceptron weighted threshold neuron model [119]. The inputs and the output of such a neuron are reduced to binary values. The synaptic weights can have positive or negative real values. The neuron output is activated by a threshold function over the sum of its weighted inputs. By denoting the input vector $I$, the output vector $O$ and the synaptic weights $\omega_{ij}$ of input j and output i, the output function of the neuron can be formulated as:

$$O_i = \Theta\left(\sum_j \omega_{ij} I_j\right) \quad with \quad I_j, O_i \in \{0,1\}, \tag{2.1}$$

where $\Theta$ is the *Heavyside* function.

*synaptic weights*

The binary inputs and outputs of the neuron model facilitate the implementation of equation 2.1: The synaptic weights are stored within current memory cells with

a current proportional to their weight value. The weight value is set digitally and is written to the memory cell by a digital-to-analog converter (DAC) with a precision of 10 bit. The usage of analog design techniques allows a space-efficient implementation of the summation as an addition of currents. Excitatory as well as inhibitory synapses are modeled by an additional sign bit stored at each synapse. The synaptic currents are internally connected to one of two different neuron input lines according to the value of the sign bit.

*temporal behavior*

The 10-bit precision of the synapse weights requires to refresh their values periodically to cope with parasitic leakage currents. Since no neuron operations can be made during the weight update, each two neighbored network blocks share eight DACs for a parallel update process (cf. again, figure 2.1). A modification of the synaptic weight values during runtime changes the connectivity of the network and thus its network answer to input patterns. The modification process has to be done by the experimenter, since the chip does not implement synaptic plasticity for an unsupervised modification of its weights. The update process requires to transfer the modified values via the external interface of the chip.

### Large-Scale Neural Networks

*network topology*

The design of the `HAGEN` chip explicitly supports the creation of multi-layered Perceptrons. The binary output data of a certain network block can be fed to the inputs of the same or other network blocks. On-chip connections are made by using dedicated local connections between the network blocks [123]. Furthermore, multiple chips can be interconnected by transporting the binary data via the digital interface. The topology of the resulting network depends on the values of its synaptic weights and its connectivity. Synaptic connections within a network block are deactivated by setting its appropriate weight to zero (cf. figure 2.2). In the more general case, recurrent networks are created by feeding the outputs of particular neurons to the inputs of neurons of the same or other network layers. In this case, the network output has to be calculated with respect to the neuron outputs of multiple intermediate layers.

*clocked operation*

A such created network needs a certain time $\Delta t$ for the propagation of the data from its inputs to the neurons of its first layer or between the neurons of intermediate layers. This introduces a spatio-temporal behavior to the network operation according to its layered topology. The time $\Delta t$ is denoted as a *network cycle* in the following. The network cycle includes the time for the simultaneous analog operation of the neurons as well as for the transport of the neuron output data to the neuron inputs of the succeeding network layers. The duration of the network cycle imposes an upper bound for the maximum frequency

$$f_{net} := \frac{1}{\Delta t} \tag{2.2}$$

of the operation of the network. The static dependency of the neural outputs from their input values allows a *clocked operation* of the `HAGEN` chip with the clock frequency $f_{net}$ in time-discrete, deterministic steps. The value of $f_{net}$ is determined by $\Delta t$ and thus by the duration of the transportation process. The network operation can even be halted by not updating the input data of the neurons. Since the state of

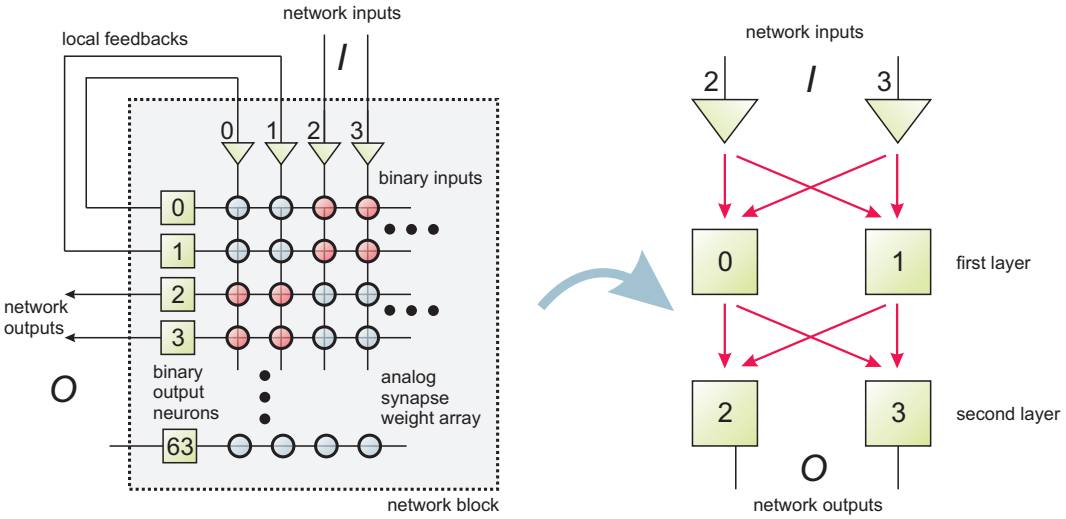



Figure 2.2: Creation of multi-layered feed-forward networks using local feedbacks. Blue synapses are deactivated by setting their weights to zero. Red synapses are activated. Recurrent networks are created accordingly.

the network can be described by the collective digital input and output values of the neurons, this allows to completely read-out the state or to set or modify the state before the execution of the network continues.

*multi-cycle connections*

For multi-layered feed-forward networks, the propagation delay of a certain input pattern to the output of the network equals $n \cdot \Delta t$, where $n$ equals the number of layers. A synchronous and deterministic operation of multiple interconnected HAGEN chips at the frequency $f_{net}$ does not necessarily require the output data of all network blocks to be transported to their corresponding inputs within the duration of a *single* network cycle. The model allows a delay of multiple network cycles for the transport of the neural data as long as the number of cycles remains constant during the network operation. This can be used for a fast operation of the network in the case that the transmission delay between two chips would limit $f_{net}$ otherwise. To calculate the output of a network block $a$ in a network built from multiple, recurrent network blocks, the following equation can be used:

$$O(t+\Delta t)_i^a = \Theta\Bigg(\sum_j \omega_{ij} I(t)_j^a + \sum_k \omega'_{ik} O(t)_k^a + \sum_l \omega''_{il} O(t-n\Delta t)_l^b \ldots\Bigg) \tag{2.3}$$

The first argument of the activation function corresponds to the external network inputs at the block $a$ itself. The second term models the local feedback from the outputs of block $a$ to its inputs. The last term models an exemplary connection from the outputs of a second network block $b$ with a delay of $n$ network cycles in the signal path (cf. figure 2.3).

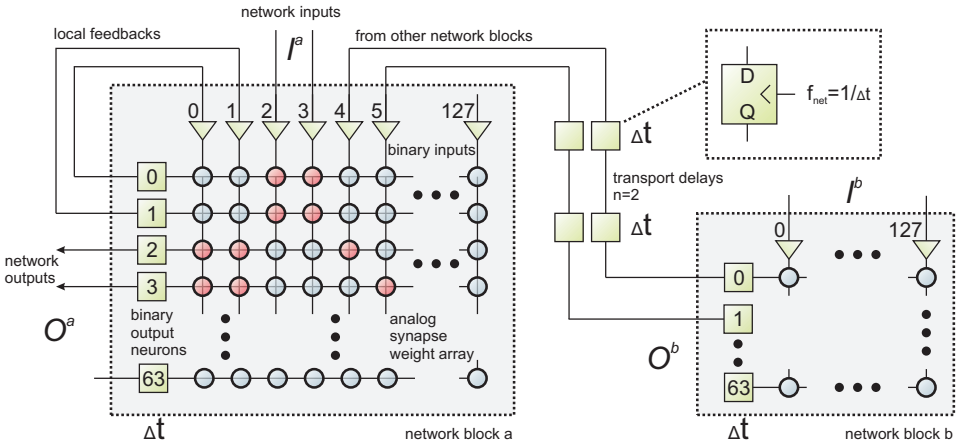



Figure 2.3: Different input sources of a network block (adapted from [123])

### Transport Network Requirements

*transport network requirements*

The interconnection of multiple HAGEN chips requires a *transport network* for the delivery of the neural data. Since the HAGEN chip features 256 neurons, the underlying transport network has to deliver up to 256 bit or 32 byte of neural data per chip within each network cycle. A deterministic operation of the neural network is possible even for a delivery within multiple network cycles, but only if the number of network cycles between each output/input pair remains constant over time as described above. In other words, the external transport network that performs the delivery has to guarantee a constant throughput at a transmission delay with a bounded jitter in the range of a network cycle for the neural data, i.e. a delivery within isochronous connections. The constraints for the transport process are relaxed due to the opportunity to halt the operation of the chips.

## 2.1.2 The Spikey Chip

The artificial neural network ASIC Spikey is the latest chip developed by the Electronic Visions(s) group for the research with artificial neural networks. It has been achieved within the projects *SenseMaker* [128] and *FACETS* [89], both supported by the European Union.

*motivation*

An intention for the development of the chip has been to implement a biologically plausible neuron model within custom-made integrated circuits. The chip features 384 artificial *leaky integrate-and-fire* [43] neurons with 256 synaptic inputs each. The array of 98304 synapses is organized within two distinct network blocks. Due to the acceleration factor of the implemented model of up to $10^5$ compared to biology, the chip allows to explore long-term biological behavior as well as exhaustive parameter sweeps of the implemented neuron model in reasonable time. Furthermore, the network model of the chip supports the coupling of multiple units to create a large-scale neural network.

Although the chip itself is an interesting research object, this thesis focuses on the networking demands of the inter-chip transfers of its neural data and gives an

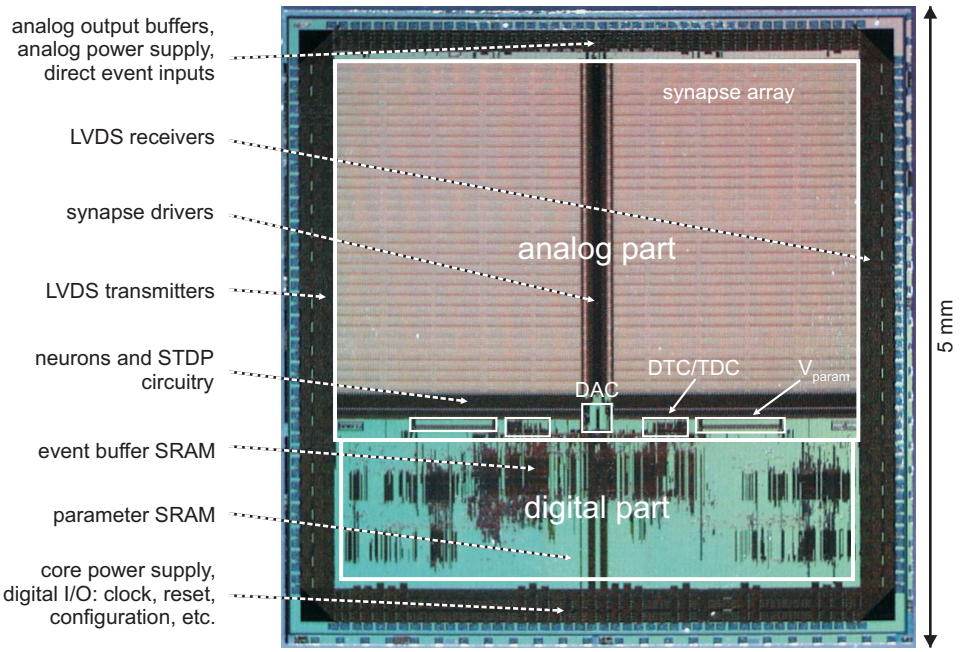



Figure 2.4: Photograph of the `Spikey` die. The top side analog part contains the two network blocks with the 384 neurons and about 100 000 synapses. The digital part below has control functions and contains neural event queues as well as the external interface (picture taken from [47]).

overview of its basic properties. The internals of the chip have been described in more detail in [124, 122, 47].

### Overview

The `Spikey` chip has been fabricated in a 180 nm CMOS technology. The mixed-signal implementation uses both, analog as well as digital design techniques. Figure 2.4 shows a photograph of the chip whereas table 2.2 lists nominal specifications.

*network blocks*

The chip is basically divided into two parts: an analog part and a digital part. The analog part of the chip is divided into two *network blocks*, which occupy roughly the top two thirds of the chip's space and contain the neurons and synapses organized within an array of 192 neurons and 256 synaptic inputs. The synapse array of a block allows to connect each neuron independently to each input. Each synapse provides an individual synaptic weight.

*neuron circuits*

The neuron circuits are located at the bottom edge of a network block. The use of analog design techniques allows for a biologically relevant model within a space efficient implementation. The division of the neurons and synapses into two network blocks has advantages for the analog implementation and also allows for a more flexible mapping of existing neural network topologies onto the chip's resources.

*digital control*

The lower third of the chip is occupied by the digital control part. Its main task is to control the clocking and the data processing of the neural events through the chip. The data flow is organized in three network layers (application, data link, physical), which are clocked at two different clock frequencies of nominal 200 MHz

| | |
|---|---|
| process features | 0.18 $\mu$m, 1 poly, 6 metal |
| die/core size | $5 \times 5\,\mathrm{mm}^2$/ $4.25 \times 4.32\,\mathrm{mm}^2$ |
| synapse size | $10.3 \times 10.5\,\mu\mathrm{m}^2$ |
| neurons/synapses | 384 / 98304 |
| supply voltage (digital and analog) | 1.8 V |
| digital core clock frequency | 200 MHz / 400 MHz (nominal) |
| adjustable analog parameters | 2969 |
| parameter resolution | 10 bit (nominal) |
| event time resolution | 156 ps (nominal, 1/16 clock) |
| event input FIFOs | 16 channels, 64 entries each |
| event output FIFOs | 6 channels, 128 entries each |
| LVDS bus data transfer rate | 2.6 GByte/s (effective) |

Table 2.2: Nominal specifications of the `Spikey` chip [122].

and 400 MHz. The digital part further contains the external interface, which is kept purely digital as well.

### Neuron Model

*spiking neurons*

The neurons of the `Spikey` chip are designed according to a biologically inspired neuron model, which is supposed to represent a major part of the neurons present in the human cortex [28]. As the name indicates, the chip models spiking neurons, more precisely of the *leaky integrate-and-fire* neuron model [43]. The interaction between different neurons is based on the transmission of neural events, or simply *spikes*. Compared to the `HAGEN` chip described in the previous section, the neuron model of the `Spikey` chip can be assumed to be closer to biology.

*VLSI implementation*

The implementation in VLSI technology represents elements of the biological neuron such as the membrane potential or the synapses by electronic counterparts. A neuron receives incoming spikes at its synaptic inputs and sends out spikes according to the state of its membrane potential. The effect of incoming spikes on the membrane potential $V$ of a neuron is modeled by the following differential equation (for a more detailed description see [122]):

$$c_m \frac{dV}{dt} = g_m(V - E_l) + \sum_k p_k(t) g_k(t)(V - E_x) + \sum_l p_l(t) g_l(t)(V - E_i) \qquad (2.4)$$

*current based model*

The three parts of the sum describe the influence of leakage currents of ion channels and the effect of excitatory and inhibitory conductance-based synapses on the membrane capacitance. The leakage reversal potential $E_l$ and the synaptic reversal potentials $E_x$ and $E_i$ can be set for groups of neurons internally. The effect of incoming spikes is described by the parameters $p_k(t)$ for the excitatory synapses and $p_l(t)$ for the inhibitory synapses. Although the synapses are implemented analog, their individual synaptic weights are stored digitally for the 4-bit digital parameters $g_k$ and $g_l$. Plasticity is introduced by changing the values of the weights.

*synaptic plasticity*

The implementation of the synapses features long-term plasticity, or more precise *STDP* [136, 5]. The learning rule implies that the individual synaptic weights are modified depending on the timing correlation of the pre-synaptic spike and the post-synaptic spike. This feature is important since STDP defines a rule to modify the synaptic weights (i.e. the model of plasticity) locally and internally at the synapses without the need of an external supervisor.

*diversity*

The diversity in the synapse and the neuron behavior is achieved by introducing about 3000 freely programmable analog parameters. Some parameters can be set for each neuron individually, whereas others affect groups of neurons. This allows to experiment with networks consisting of a large diversity of neuron properties, but also to test the neuron model by varying parameters for single neurons.

*speedup*

The implementation of equation 2.4 in VLSI technology automatically leads to small time constants of the neuron behavior compared to the biological neuron. The timing acceleration or *speedup* of the chip can be controlled by the adjustment of analog parameters such as the time constants of the membrane potential in the range of roughly $10^4$ to $10^5$ compared to biology. Lower values result in very small currents within the neuron and synapse circuits and may affect the quality of the model. At the value of $10^5$, 1 ms of biological time equals 10 ns of chip time. This speedup not only allows to simulate long term biological behavior in reasonable time, but also to do exhaustive parameter sweeps of the implemented neuron model.

### Large-Scale Neural Networks

*connectivity*

The `Spikey` chip features interconnection capabilities to build large-scale artificial neural networks. This can be done by forwarding the spike events from dedicated neurons to the synaptic inputs of others depending on the synaptic weights $g_{l,k}$. Due to the matrix topology of the synapse array, even a single chip allows to create networks of 384 neurons with 256 inputs each by using all of the about 100 000 synapses. Even more important, the digital interface allows to extend the neural network beyond chip boundaries by interconnecting multiple chips to a large-scale neural network.

*limitations*

Due to the limited input count of a single neuron of 256 synaptic inputs, a neuron cannot get spikes from more than 256 neurons, i.e., *fully connected* networks are possible up to the size of 256 neurons. Fully connected networks that use only on-chip feedbacks are further limited to 192 neurons. The input count limitation and its consequences are further discussed in section 2.3.3.

*interconnecting chips*

Although the neuron implementation uses analog design techniques, the spike events are encoded as digital pulses but in continuous time. Spikes forwarded to synaptic inputs on the same die (to the original or the adjacent network block) stay within the continuous time domain of the analog part of the chip. If multiple chips are to be connected, spike events leaving the die are encoded with address information and time-stamps of a resolution of 156 ps (i.e. 1/16 of the clock period) before being sent via the external interface. After being transported to the destination chip, the incoming spikes are resynchronized to the local clock according to their time stamps within the digital part of the chip before being forwarded to the synaptic inputs [47].

### Transport Network Requirements

The spike transmissions between multiple chips require an underlying network to transport the digital events while ensuring specific timing requirements. These requirements are mainly determined by the analog implementation and the selected speedup of the chip:

*continuous time operation*

1. The analog neuron circuits have no clock, but operate in *continuous time* with their timing determined by reference potentials and current values. This leads to the fact that the network operation cannot be halted as in digital systems, where a halt and a restart of the clock signal allows the system to be paused. During an experiment, the neurons continuously produce spike events depending on the membrane's state. This produces a more or less continuous load on the network being utilized for the transportation process.

*low jitter*

2. Since the inter-neuron connections model biological axons and dendrites, the *delay* of the connections has to be within specific values according to the programmed neuron acceleration factor of $10^4$ to $10^5$. To be more precise, the STDP functionality implemented at the synapses is based on the arrival times of the incoming events. The digital control logic of the ANN is therefore able to re-synchronize incoming events with a precision of 156 ps according to the time stamp of the event. The re-synchronization process requires the events to be transmitted within the correct clock cycle. Delay variation (jitter) in the spike delivery has to be canceled out by the usage of buffers, which again increases the final connection delay. Therefore, jitter has to be avoided as much as possible. The `Spikey` chip thus requires isochronous connections for its external data transport.

*variable bandwidth*

3. The data rates arising for the inter-chip transfer of spike events are not constant, since the generation of a spike at a neuron is based on the timing of the arrival of incoming spikes. The data rate required to transport the spikes of multiple neurons between two chips is therefore non-deterministic. Furthermore, the implemented neuron model has only limited coverage with a biological neuron. Due to this, the resulting spike frequencies can hardly be predicted. As a rough estimation, a mean firing rate of 10 Hz for a biological neuron [1] approximately results in the spike rate of 1 MHz for its chip's counterpart at a speedup of $10^5$. In the case of bursting of a single neuron or a synchronized behavior of multiple neurons, this value can be exceeded significantly.

The reader will notice that the transport requirements of the `Spikey` chip resemble the requirements of the `HAGEN` chip. Both chips require isochronous connections for the transmission of their neural data. However, the requirements of the `Spikey` chip are more strict than the requirements of the `HAGEN` chip, which is operated in a clocked network with a deterministic network load (cf. section 2.1.1).

### External Interface

*physical interface*

The `Spikey` chip has been designed to operate within the Stage 1 framework of the Electronic Vision(s) group and is physically interfaced by the Virtex-II Pro FPGA (cf. section 2.2.1). The external interface is implemented within the digital part of the

chip. The physical layer complies to the HyperTransport [60] I/O link specification. It consists of two bidirectional 8-bit source synchronous links operating at nominal 400 MHz double data rate (800 Mbit) allowing a total transfer rate of 1.6 GByte/s for both input and output. The chip also supports daisy chaining of multiple units on a custom-made board, but this indeed reduces the total bandwidth of the interface to a fraction usable for each chip.

*interface rate*

The ANN-FPGA interface transfers packets of 64 bit in size and combines up to three spike events into a single packet. The interface data rate given in table 2.5 denotes the optimal case that three events can be combined within a single interface packet, which is not always possible especially for low data rates. After passing the interface, the spike events are encoded separately by the ANN control logic within the programmable logic. This process adds additional timing and address information, which enlarges the encoding to 21 bit per single event (cf. chapter 4.3 of [47]). Therefore, the isochronous connections between two distinct ANN chips have to deliver multiples of 21 bit.

*operation control*

The operation of the chip (the queuing of events, the interconnection to the network etc.) is managed by an *ANN controller* implemented within the programmable logic of the FPGA. The high-level control is handled in software running on the control PC of the framework. The description of the interface of the chip, of the internal event transport, as well as of the controller implemented within the FPGA is beyond the scope of this thesis. A detailed description can be found in [47].

## 2.2 The FACETS Stage 1 Framework

*overview*

The FACETS Stage 1 framework has been developed within the Electronic Vision(s) working group in Heidelberg [46, 39]. It consists of one or multiple *backplanes*, which can be equipped with *network modules* for which they provide the interconnections and power supplies. The backplanes are connected to a *control PC* with the Linux [79] operating system, on which a high-level software provides a graphical user interface [38, 53]. Its primary intention is to host and to interconnect multiple ANN ASICs for the research on large-scale hardware neural networks. Besides of this, the framework is a general tool for the operation and the interconnection of custom-made ASICs within a distributed environment.

*scope*

The scope of the following description is reduced to the main aspects of the framework relevant for this thesis, i.e., the operation of the ANN chips `HAGEN` and `Spikey` and the development of a transport network for their neural data. For a more detailed technical description see [46]. A schematic overview of the framework is shown in figure 2.5.

### 2.2.1 Nathan Network Module

The `Nathan` network module is the core component of the framework. The purpose of the module is to host a single ANN chip, which is plugged onto the module via two surface mount technology (SMT) connectors (`Spikey`) or into a dedicated socket (`HAGEN`). A single module features all parts that are necessary for the operation of its ANN. Figure 2.6 shows a photograph of the network module.

*parts*

The central part of the module is a programmable logic FPGA device, which is

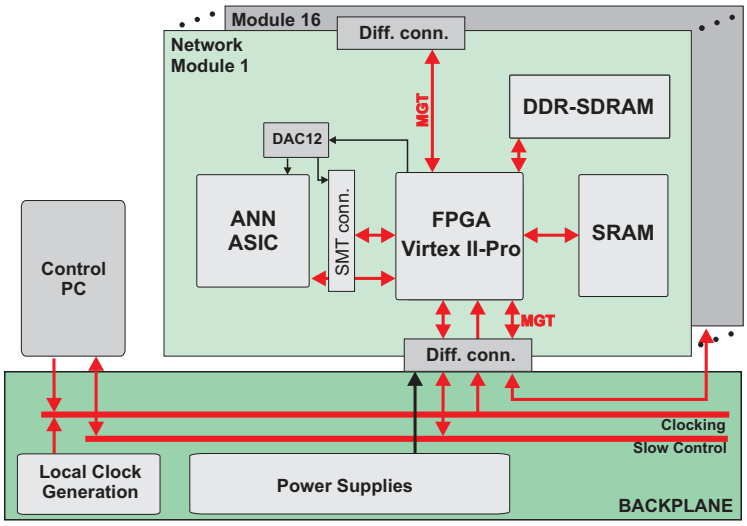



Figure 2.5: Schematic of the FACETS Stage 1 framework. Multiple network modules are plugged into a backplane, which provides the interconnects and the power supply. The control PC hosts the user interface (taken from [46]).

placed directly next to the ANN chip. The usage of programmable logic allows to implement a high-performance and flexible interface to the local ANN. The FPGA is further connected to all other components on the network module and contains the interface logic to the parts within separate functional entities implemented in the programmable logic:

- Two independent ZBT SRAM modules [63] of 512 KByte each are mounted directly onto the Nathan printed circuit board (PCB). The modules have two clock cycles delay for a low-latency access via a $2 \times 32$ bit bus that operates at frequencies of up to 200 MHz.
- An additional DDR SDRAM socket on the module can be equipped with commercial memory modules used on laptop computers to provide larger amounts of local memory of up to 2 GByte. The interface to the controller within the FPGA operates at transfer rates of up to 150 MHz DDR via a 64 bit wide bus [125].
- A temperature sensor is placed directly between the two ANN connectors. It measures the temperatures of the ANN as well as of the FPGA and is able to alert the system if maximum values are exceeded.
- A four-channel DAC with a resolution of 10-bit in the range of 0 V to 3.3 V [82]. The speed of the DAC is about 1 MHz and it is used to provide bias voltages to the ANN device.

*connectivity*

Each backplane can be equipped with up to 16 modules at the same time. The modules are plugged into the PCB in parallel using differential connectors. The connectors do not only provide the necessary power supply to the module, but are also used for the framework-wide interconnections of multiple network modules.

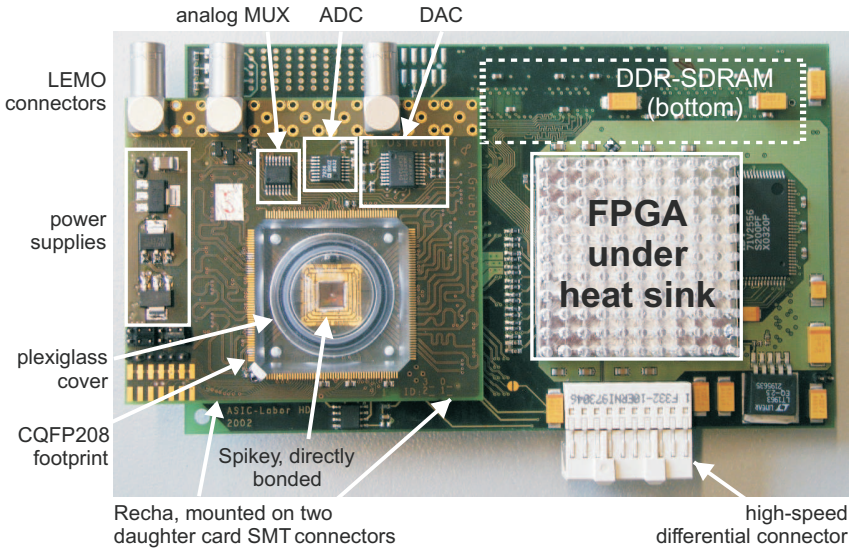



Figure 2.6: Photograph of the Nathan network module. The module is equipped with the carrier board that hosts the ANN ASIC Spikey (taken from [47]).

### Virtex-II Pro FPGA

The central part of the Nathan module is a commercially available Xilinx Virtex-II Pro FPGA of the type `xc2vp7` [154]. It provides programmable logic organized in 11.088 4-input look-up table (LUT) logic cells and 4928 slices of 2 flip-flops each. The presence of programmable logic is a main aspect for the high flexibility and the universality of the whole network module. This allows to implement the ANN control algorithms directly next to the ANN device. Consequently, a transport network that interconnects multiple ANN chips can be implemented directly within the FPGAs to achieve a low-latency transport of the neural data.

To aid the development of complex and high performance functionality, the FPGA furthermore features several integrated cores, which are embedded as ASICs within the programmable logic:

- A single 350 MHz 32 bit IBM PowerPC 405 CPU core.
- Eight multi-gigabit serial links for high-speed communication.
- Four digital clock managers (DCMs) for clock de-skewing and frequency synthesis.
- 44 dual-port block RAM (DPBRAM) elements programmable from 16 K $\times$ 1 bit to 512 $\times$ 36 bit with a total of 792 Kbit per FPGA.
- 44 embedded 18 $\times$ 18 bit multiplier blocks.

The main part of a digital design is usually implemented within the universal 4-input LUT (LUT4) elements of the FPGA. The listed cores can be accessed by internal routing resources for improved performance of particular algorithms and to save LUT4 elements for more general tasks.

*hardware / software co-design*

**PowerPC 405** The embedded 32 bit IBM PowerPC 405 [152] is connected to the surrounding programmable logic with multiple standardized bus interfaces. The FPGA manufacturer provides software tools, libraries and intellectual property (IP) cores for a hardware/software co-design that allows for a convenient programming of the PowerPC using the C/C++ language. The co-existence of high-level software and programmable hardware on the same substrate close to the ANN ASIC allows to execute the chips's training algorithms completely locally on the network module or at least to perform time-critical, calculation intensive operations in parallel within the programmable logic [125].

*Linux on the network module*

To facilitate the execution of existing ANN control software directly on the network module, the Linux [79] operating system has successfully been ported to the `Nathan` network module during this thesis. The connection between the Linux system on the network module and the Linux system on the control PC is performed via the IP protocol (cf. section 2.2.4).

*multi-gigabit transceivers*

**Multi-Gigabit Transceivers** The multi-gigabit transceivers (MGTs) are duplex serial links that operate at speeds of up to 3.125 Gbit/s. The MGTs can be configured to comply to a couple of industrial transmission standards like InfiniBand [62], Gigabit Ethernet and 10-Gigabit Ethernet (XAUI) [92], FibreChannel [37], Serial ATA [120] etc. If all transceivers are used, the FPGA features a total amount of on-chip and off-chip bandwidth of up to 5 GByte/s. The MGTs are interconnected externally via impedance-controlled LVDS transmission lines. The transmission lines of four of the MGTs are routed to the differential connector on the bottom of the network module and further to other network modules via the backplane. The lines of the four remaining MGTs are routed to the differential connector at the top of the module. This can be used to extend the topology of the pre-defined connections of the backplane or to interconnect multiple backplanes. The connections are added by hand using standard serial ATA cables.

### 2.2.2 Backplane and Control PC

*purpose*

A single backplane hosts up to 16 `Nathan` network modules and supplies the modules with the required power and connectivity. The following description refers to the updated version designed by Dan Husmann in 2007. Besides the hosting of the `Nathan` modules, the backplane provides a separate FPGA of the same type as on the network modules with many of its package pins routed to additional connectors (not shown in figure 2.5). The FPGA on the backplane supports direct and independent connections to each network module for configuration and control of its FPGA. The power is provided by a commercial ATX power supply. For details refer to [46]. Figure 2.7 shows a photograph of the new version of the backplane equipped with three modules.

### 2.2.3 Connectivity

An important purpose of the backplane is to provide the necessary connectivity of the `Nathan` modules to each other and to the control PC. Multiple interconnections are provided:

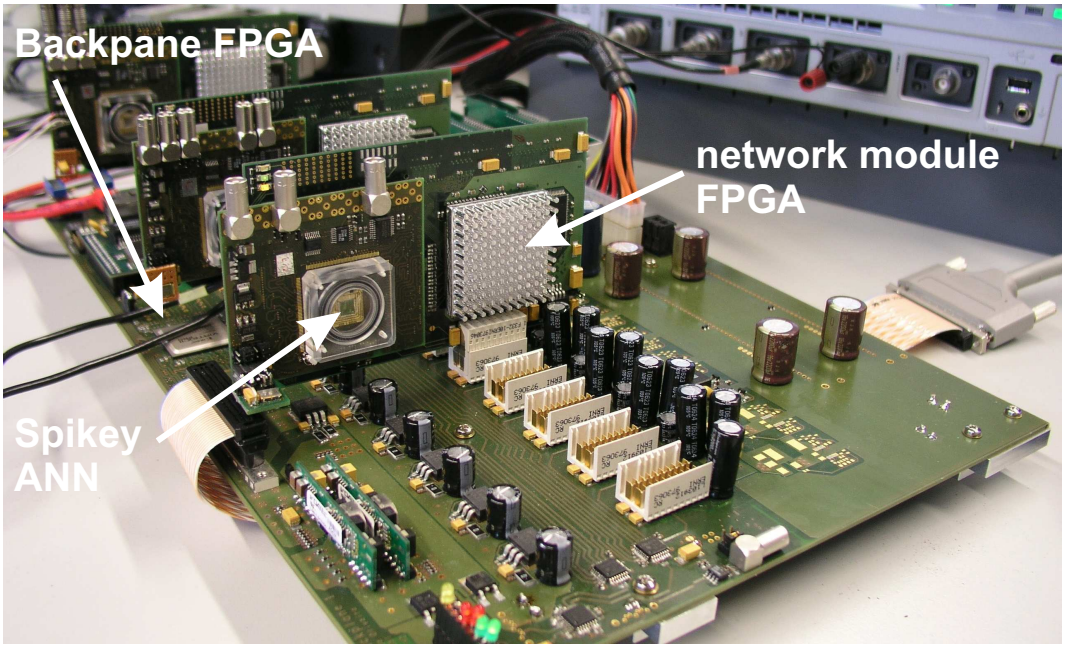



Figure 2.7: Photograph of the backplane with three network modules. The connection between the backplane and the control PC can be seen at the right hand side.

*gigabit network*

1. The main connectivity is provided by the transmission lines between the MGTs for transmissions at data rates of up to 3.125 Gbit/s. These physical interconnects are denoted as the *gigabit network* in the following. The implementation of the transport network of this thesis uses the gigabit network for its data transports.

*SlowControl*

2. The *SlowControl network* interconnects the control PC with each network modules via the backplane FPGA. The SlowControl is designed to commit control commands and read back status information with moderate speed to minimize the consumption of programmable logic within the FPGAs. The connection to the control PC uses a commercial small computer system interface (SCSI) connector to the custom-made Peripheral Component Interconnect (PCI)-board *Darkwing* [3], which is plugged into the PC. The PC can take multiple boards to control multiple backplanes at the same time.

*gigabit ethernet*

3. It is possible to attach a commercial Gigabit Ethernet PHY [49] to the backplane for a high-performance connection to the control PC. The PHY is directly connected to the backplane FPGA. By using an appropriate network switch, a single Gigabit Ethernet card plugged into the control PC is sufficient to access multiple backplanes.

*hyper transport*

4. Since the MGTs of the backplane FPGA supports multiple different high-speed standards, it is further possible to attach the backplane to the control PC using modern high-speed interconnection standards. An interesting opportunity is the connection to the HyperTransport [60] interface of a modern CPU by using appropriate interface cards [57]. This enables a high-speed and low-latency

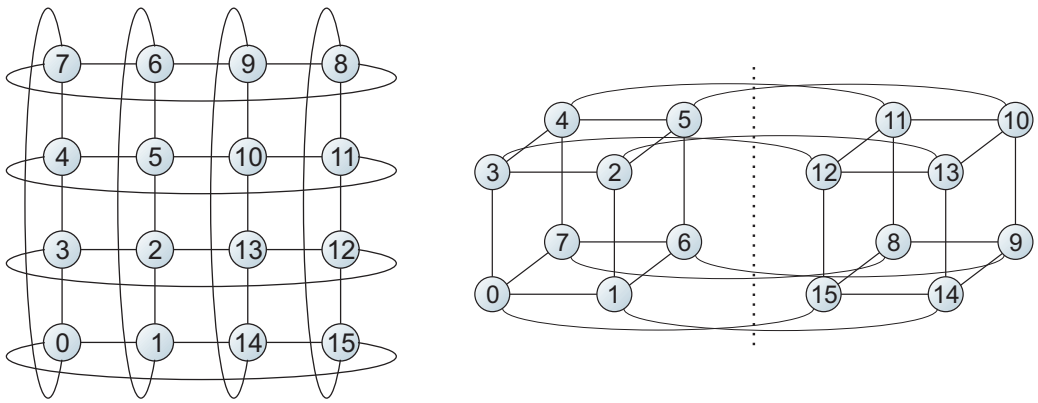
access to the programmable logic within the FPGA by high-level software executed on the PC.

The last two options are to enhance the connectivity of the backplane to the control PC compared to the existing SlowControl network and are not yet available. The first two options are usable for a communication between the network modules. The purpose of the gigabit network is to transport large amounts of data, whereas the SlowControl is better adopted to transfer control and status information. The gigabit network and the SlowControl are explained in more detail in the succeeding paragraphs.

### Gigabit Network

*physical topology*

The backplane interconnects the MGTs on the network modules to create a gigabit network between all modules. The connections between the MGTs are physical transmission lines that are routed directly point-to-point between the differential connectors of the modules. Each network modules is bidirectionally connected to four others, with an independent unidirectional connection in each direction. The topology of the resulting 64 differential connections equals a 2-dimensional toroidal structure or a 4-dimensional binary cube, both with bi-directional edges. It is illustrated in figure 2.8. Since the external pins of four additional MGTs are routed to the top connector of the network modules, the topology can be extended by adding up to four arbitrary connections between the modules.




(a) (b)

Figure 2.8: Hardwired topology of the backplane. Each network module is interconnected to four others. (a) Illustration as a 2-dimensional toroidal structure. (b) Illustration as a 4-dimensional binary cube. The fourth coordinate switches between the two 3-dimensional cubes.

## 2.2.4 SlowControl and PowerPC Operation

*high-level control*

A distributed setup like the Stage 1 framework requires a convenient control mechanism and programming model for higher level software. Two main functionalities have been implemented for this purpose within the scope of this thesis:

- The development of the *SlowControl* network.
- The installation of the Linux operation system on the PowerPCs [152], which are embedded within the FPGAs on the network modules.

Since the high-level control of the setup is not the main objective of this thesis, both mechanisms are only briefly described in the following.

### SlowControl

*purpose*

The purpose of the SlowControl is to provide a simple framework-wide access and control of all functional elements of the framework, as e.g. the ANN chips, the SDRAM memory on the network modules or the PowerPCs. Since all these elements are interfaced by FPGAs, the SlowControl accesses the corresponding modules within the programmable logic. For this reason, the SlowControl provides a set of basic I/O commands (e.g. FPGA configuration, read data, write data) usable from higher-level software. The services of the SlowControl are accessible from both, the control PC and also the PowerPCs.

*connectivity*

The physical interconnects of the SlowControl are illustrated in figure 2.5. The interconnects are serial lines for the data transfers with a clock signal in parallel and operate at speeds of up to 100 MHz. The connection to the FPGA is made via the differential backplane connector of the modules. It is stated here again that the SlowControl network and the gigabit network are separate and independent networks that share no physical interconnection resources. The SlowControl has been initially designed for the first version of the backplane, at which it uses a ring topology to interconnect the FPGAs on all network modules with the control PC. The MAC layer of the SlowControl operates similar to the IEEE 802.5 Token Ring standard [145]. The new version of the backplane replaces this by a parallel access to all network modules via the single backplane FPGA. The present implementation uses the above mentioned SCSI connection between the backplane and the control PC.

*functional parts*

The functional parts of the SlowControl are distributed across several parts of the system. The local access of the programmable logic as well as the MAC layer of the network are implemented within the several FPGAs of the framework (on the network modules, on the backplane and on the PCI card within the PC). Higher-level functions are implemented in software on the control PC and feature a multi-user access to the hardware via a simple and generic software interface. The high-level control of the ANN chips is implemented on top of the SlowControl (see e.g. [53, 38]).

### PowerPC Operation

*Linux installation*

The embedded PowerPC within each FPGA is powerful enough to host a conventional Linux [79] system. The Linux system has been installed by configuring and compiling a *kernel* variant that is dedicated for embedded CPUs [95]. The PowerPC core is connected to the programmable logic via its processor local bus (PLB) [61] and thus has access to the SDRAM memory on the network modules. To start the system, the kernel and a ramdisk with basic executables are written into the memory. The CPU starts the execution of the kernel at a specific address and requires parameters to be set within its processor registers. The ramdisk allows to automatically execute pre-compiled software after the operation system has booted.

*interfaces*

The access of the programmable logic by the software on the PowerPC is performed via the several buses of the CPU. The SlowControl network and thus the functional modules within the programmable logic are accessible via its device control registers (DCR) bus. The exchange of data with the control PC can be achieved via the SDRAM, which is accessible from both systems. Since the PowerPC and the control PC use a different alignment of its data (*endianess*), the exchange of data has been facilitated by a modification in the memory handler of the Linux system. The software on the PowerPC uses this to allocate a memory mapping with the alignment of the PC.

*high-level connection to PC*

Finally, a convenient high-level connection between a PowerPC and the control PC has been established by means of an IP network. Within both systems, a virtual network card has been programmed whose data is physically transported via the SlowControl and the memory modules as described. An IP network is installed on top of this virtual network card and can be used for general purpose TCP/IP connections between software processes on both systems. This technique enables high-level software on both systems to communicate with standard network protocols. This not only facilitates the execution of distributed software [39], but also the debugging during the development phase on standard PCs.

## 2.3 Neural Network Experiments

*overview*

The previous sections described the existing Stage 1 framework and the two analog neural network chips `HAGEN` and `Spikey` built by the Electronic Visions(s) group. Although the framework allows to investigate single neural network chips, its strength lies in the adequate hardware interconnection of multiple chips to create and operate large-scale distributed neural networks. This section now leads to the formulation of the special interconnection demands of neural network applications. The demands arise out of the multiple different tasks to be solved during the execution of the experiments.

### 2.3.1 Experimental Setups

The execution of hardware neural network experiments requires different tasks to be performed prior and during an experiment. As an example, the following tasks are considerable:

- The transfer of a biological network model to the hardware of the framework.
- The configuration of the distributed ANN chips.
- The inter-chip transport of neural data.
- The inter-module communication of the training algorithms.
- The transport of training data, network stimuli or experimental results.
- The execution of the high-level user interface.

The complexity of the framework allows to implement these different tasks in different ways, which are more or less adopted to the corresponding problems. Two main aspects have to be taken into account:

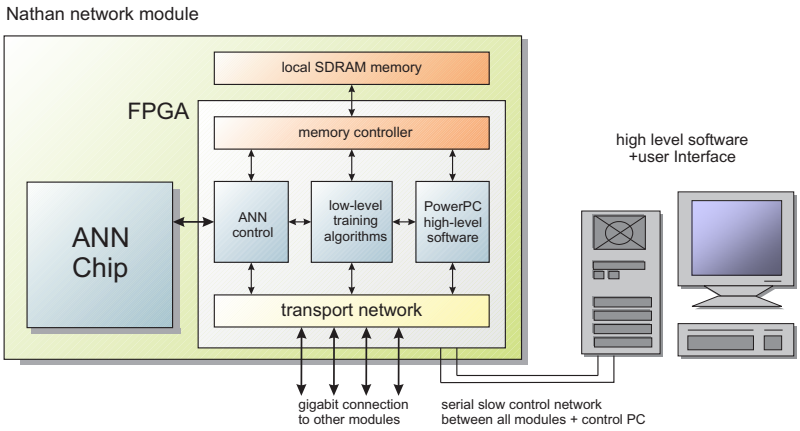



Figure 2.9: Schematic view of the FACETS Stage 1 framework with focus on the substrates, where algorithmic parts could be implemented. The FPGA used provides programmable hardware as well as a PowerPC CPU close to the ANN chip. High-level software can be executed on the control PC. The substrates have different interconnection probabilities and different complexity limitations.

- The distributed nature of the system: The framework can be seen as a collection of equal calculation units, the `Nathan` network modules. A single backplane takes up to 16 modules, but the system can be extended by combining multiple backplanes. For an optimal usage of the given resources, not only the neural network itself, but also the training and evaluation algorithmic tasks have to be parallelized and distributed. The communication bandwidth between the several `Nathan` network modules may limit the speed of the algorithms used.

  *distributed system*

- The existence of multiple substrates: In contrast to a computer cluster that represents a parallel system in software, the Stage 1 framework consists of ANN hardware, programmable hardware, a local embedded PowerPC and the control PC. Although the complexity of the system can be hidden from the user interface on the control PC, it has to be decided on which substrate to implement the algorithmic parts of the experiment.

  *multiple substrates*

Figure 2.9 shows a schematic view of the framework. The focus lies on the different substrates on which algorithms could be implemented. The decision where to implement algorithmic parts depends on the required computational power, the level of parallelism and the required inter-module communication.

### Programmable Logic

Programmable logic is executed in parallel and has a high computing power close to the ANN chip. The main drawback for the usage of programmable logic are the limited logic resources of the FPGA. Furthermore, programmable logic commonly requires a comparably high effort for development and debugging. Parts that have to be executed within the programmable logic are the ANN controller, which interfaces the ANN chip, the implementation of the transport network as well as the SlowControl (cf. section 2.2.4). It is further possible to implement local low-level training

*parallel, fast, limited space*

algorithms within programmable logic to exploit the possible high data rate to the local ANN chip [125].

#### PowerPC Software

*high-level, limited complexity and connectivity*

From the software view, the `Nathan` network module combines a complete embedded computing system around the PowerPC CPU close to the local ANN chip. The buses of the PowerPC are directly connected to the surrounding programmable logic.The support of the Linux operating system on the CPU and its operating frequency of up to 350 MHz makes it possible to implement higher-level training algorithms in software and to port existing algorithms formerly executed on standard PC systems on the network module. Due to the distributed nature of the setup, the PowerPC is more feasible to implement training algorithms with only few need of inter-module communication to not to reduce the performance advantage of local software.

#### Control PC Software

*high-level, complex, low connectivity*

The single control PC mainly hosts the user interface and schedules the different experimental tasks executed on the network modules. The single control PC is a standard PC system, but has only limited connectivity to the network modules via the SlowControl. In a first step, already existing software algorithms can be used immediately and be executed on the control PC. A transfer of the algorithms to the PowerPC on the network modules or a later implementation in programmable logic is possible to speedup the experiment.

#### Example Setups

As an example, in [38, 39, 125] is described how experiments are carried out on the framework using the `HAGEN` chip. The chip is used for pattern recognition and classification experiments. The software `PyNN` [115] is used for high-level control on the control PC. In [125] it is described how an experimental setup using the `HAGEN` chip interfaced by and FPGA directly connected to the control PC via the `Darkwing` card [3] is transferred to the Stage 1 framework. In [14], first experiments with the `Spikey` chip are described.

### 2.3.2 Interconnecting Multiple ANN Chips

The current development of the setup comprises that each `ANN` chip is interfaced exclusively by the FPGA on the Nathan network modules. A fully equipped backplane therefore supports the distributed operation of up to 16 ANN chips. Later developments will introduce the interconnection of multiple ANN chips that are directly connected to a single FPGA [67]. It has been stated in sections 2.1.1 and 2.1.2 that the two chips `HAGEN` and `Spikey` both require a transport network that provides isochronous connections for its interconnection.

*ANN controller*

The two chips are not directly connected to the network by are interfaced by special ANN-controllers implemented within the programmable logic. Both chips have different controllers. The controllers manage the configuration of the chips, the feeding of the chip with neural input stimuli as well as the recording of neural

output data for evaluation. Furthermore, the ANN controllers are interfaced to the transport network for the network-wide interconnection of the chips. The controller are therefore required to implement buffering techniques for the transmit and the receipt of neural data, as well as a re-synchronization logic to provide incoming neural data to the chip at the correct time (cf. section 3.2 of [47]).

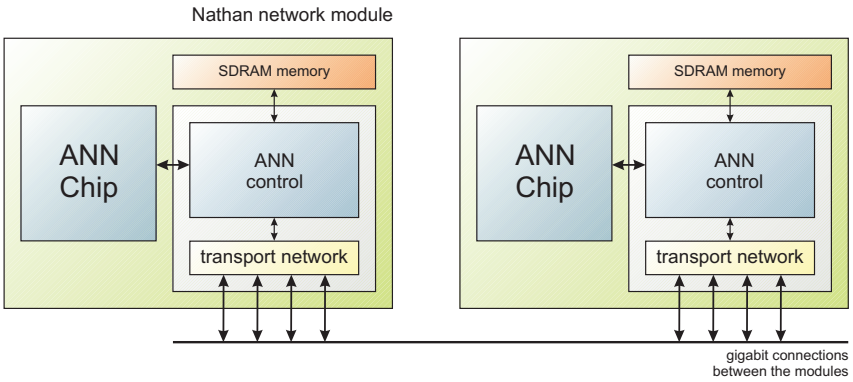



Figure 2.10: Large-scale interconnection of multiple ANN chips. The transport network provides isochronous connections via the multi-gigabit links of the framework. Each chip is interfaced by a dedicated ANN controller that is itself connected to the network.

*levels of connectivity*

The resulting networks can be implemented with different levels of interconnectivity (cf. figure 2.10):

- Local feedback connections on the same ANN chip.
- Local feedback connections provided by the ANN controller.
- Connections between different ANN chips connected to the same FPGA.
- ANN chips interconnected by the transport network.

Note that the different types of interconnects result in different connection delays. Large connection delays either result in the long duration of a network cycle (in case of the `HAGEN` chip) or reduce the possible speedup of the chip or the even quality of the neuron modeling (in case of the `Spikey` chip). The requirements to the transport network are discussed in section 2.4. The considerations to be made when mapping existing neural network netlists to multiple ANN chips is considered first in the succeeding section.

### 2.3.3 Neuron Mapping

*motivation*

The execution of neural network experiments requires a configuration of the synapses and the neurons of the ANN chips. The configuration of the synapses equals the definition of the topology of the neural network to be investigated. A possible scenario might be to model existing neural network topologies on the framework that have been previously simulated in software to compare the results [14]. The neural network topologies can be represented as a list of interconnected neurons, the *netlist*.

*mapping process*

Since the present ANN chips model neural networks on the neuron level, a *mapping process* has to be executed prior to the experiment, which assigns each neuron

of the netlist to be modeled a physical counterpart on an ANN chip. The result of this mapping operation is an assignment table defining the hardware position at which each modeled neuron has to be placed. The table is then used to configure the ANN hardware accordingly (cf. figure 2.11). The resulting mapping can also be represented as an adjacency matrix of the hardware neurons with the synaptic connections of the ANN chips as the matrix elements.

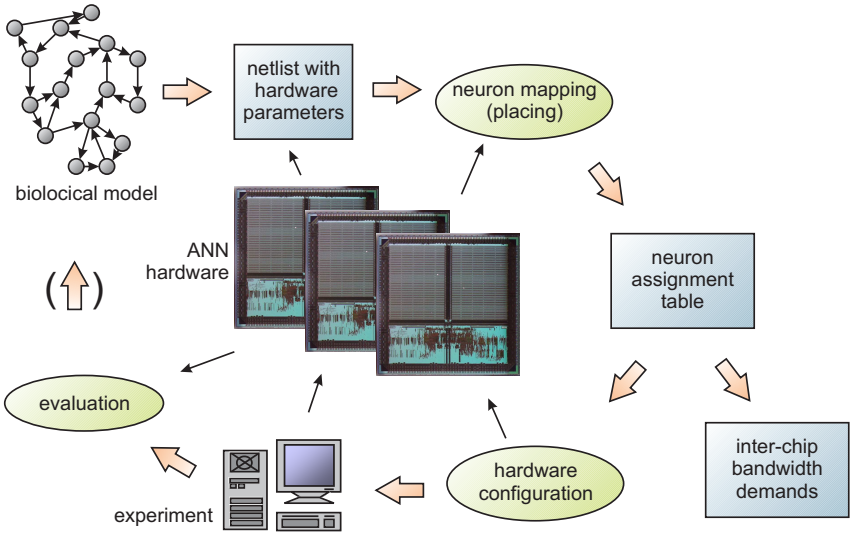



Figure 2.11: Schematic view of the pre-experimental mapping process. The netlist of the neural network topology has to be mapped on the available hardware. Thin arrows represent the usage of hardware information or hardware interactions.

*connectivity limitations*

It has been shown in the sections 2.1.1 and 2.1.2 that the topology of the possible networks is limited by the interconnection properties of the chips. When placing the neurons, the mapping process has the limited hardware resources to take into account: on-chip connectivity and inter-chip connectivity. On-chip connectivity is limited by the number of local feedbacks as well as the maximum number of synapse drivers per neuron (the input count). Limitations may hinder particular neurons to be connected or may require the neurons to be removed from the network such that the given netlist cannot be mapped. Inter-chip connectivity is limited by the interface of the chip and by the available bandwidth of the physical links between the chips.

*constraints*

The following paragraphs discuss the hardware constraints, a mapping algorithm has to consider. Hard constraints of the mapping process include the physical properties as the limited input count of the neurons or the number of available ANN chips. This either hinders to exactly map particular biological network models on the hardware or results on drops of neurons or synapses. Besides of this, the mapping algorithm has to perform optimizations like to minimize the number of chips used,to minimize the inter-chip bandwidth needed for communication as well as to minimize the connection delay. The quality of the algorithm and the difficulty of the mapping task for a given netlist and hardware setup determine the resulting congruence of the mapped network to the original netlist.

### Network Blocks

The two ANN chips `HAGEN` and `SPIKEY` consist of groups of neurons within multiple network blocks with the identical number of neurons, synaptic inputs and synapses for each block. A `HAGEN` network block contains 64 neurons with 128 inputs, whereas a `Spikey` network block contains 192 neurons with 256 inputs each. Each network block can be fully connected, i.e. each synaptic input can independently be connected to each neuron with an individual synaptic weight.

*hierarchical level*

The resulting hardware topology can be viewed as an accumulation of interconnected network blocks, or in other words, the network blocks introduce an additional level of hierarchic between the single neurons and the whole network. Note that this hierarchic level does usually not exist in the initial biological model that is to be mapped. In the case of using multiple blocks, the mapping algorithm has to cluster the modeled network into several subnetworks, which are then assigned to the network blocks of the ANN chips (cf. figure 2.12).

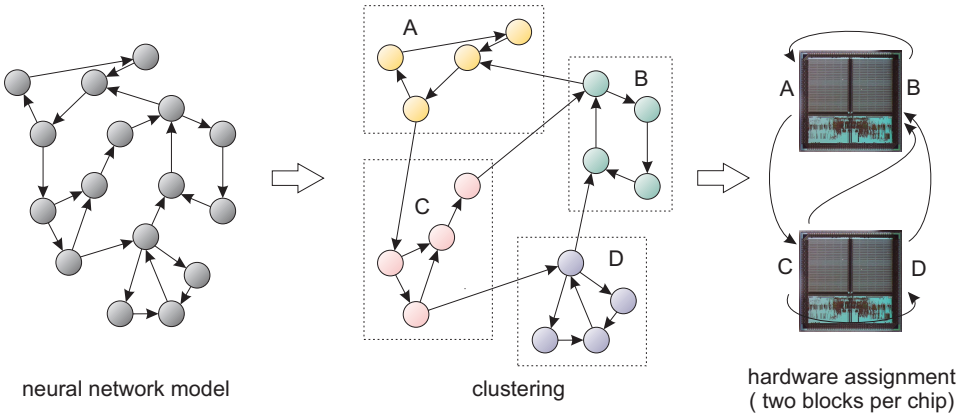



Figure 2.12: Networks occupying multiple hardware blocks have to be partitioned into clusters and to be mapped one by one. The clustering process has to minimize inter-block communication. The example uses a maximum neuron count of four per network block for simplification.

### Input Count Limitation

*limited number of synaptic inputs*

On both presented ANN chips, the number of synaptic inputs (the input count) to a hardware neuron is limited. Due to the analog implementation of the neuron, this limitation cannot be exceeded by interconnecting multiple chips. As a result, fully connected networks are possible only up to the size of 128 neurons (`HAGEN`) or 256 neurons (`Spikey`). Network models including single neurons with higher input count cannot be implemented on the chips. Fully connected networks that use only on-chip feedbacks are further limited to 192 neurons. This is since the on-chip connections feed the output of each neuron to only a single dedicated input at each network block, which corresponds to its own block position (0 . . . 191). This limitation has consequences for the *randomness* when creating networks consisting of multiple `Spikey` chips. If the total number of chips is high, only a fraction of neurons from each chip can be used as inputs at any other chip. Note also that

this limitation holds for a whole network block, thus the inputs of *all* neurons of a network block are limited to *the same* 256 neuron outputs that feed the synapse drivers of the block. Consequently, the input count limitation also affects networks being only sparsely connected.

*example calculation*

Consider a network of 1000 neurons randomly connected with an interconnection probability of 5 % between any two neurons to be mapped on several `Spikey` chips. Each neuron will have about 50 other neurons from which it receives input. Imagine that the 1000 neurons are clustered into 4 chips with 8 blocks of 125 neurons. Although each individual neuron of a block receives input from only a mean of 50 other neurons, all neurons of a block have their inputs *independently* assigned out of the total number of 1000. To calculate the required input count of a network block, note that the output of a particular neuron is requested as input by each of the neurons of the network block with a probability of $p = 0.05$. The probability that this neuron is requested by at least any of the 125 neurons as input is then

$$P_{\text{any}} = 1 - P_{\text{none}} = 1 - 0.95^{125} = 99.84\,\% \tag{2.5}$$

for each block. Each block therefore requires the mean number of

$$P_{\text{any}} \cdot 1000 = 998 \tag{2.6}$$

neurons as input. Since a network block has a limited number of 256 inputs, this network cannot be achieved using only four ANN chips.

According to the input count of 256 synaptic inputs, the mapping algorithm could avoid to exceed the input count by placing only five neurons onto each network block. Although possible to map, this solution, would require a total number of 200 network blocks or 100 ANN chips. This would not only result in a low hardware efficiency, but also in a required high interconnectivity as well as in a high resulting connection delay since 100 chips can hardly be fully connected.

### Bandwidth and Delay

The limited input count of a neuron or network block is a hard constraint for the mapping algorithm and may force to drop neurons and synapses of the netlist to be mapped. In contrast to that, the mapping algorithm has to minimize the required bandwidth as well as the delay of the connections between the placed neurons as an additional side condition.

*inter-chip bandwidth*

The physical topology of the gigabit network of the Stage 1 framework is not fully connected, but represents a 2-dimensional toroidal structure that can also bee seen as a 4-dimensional binary cube (cf. figure 2.8). The placement calculated by the mapping process will require multiple synaptic connections between neurons on different chips to share the same physical links. Neuron connections between distant chips occupy the bandwidth of all links on the route between the source and the destination chip. Since the link capacity is limited, the mapping algorithm should optimize the placement for a minimum and balanced bandwidth by combining groups of highly connected neurons and placing the group on the same chip.

*inter-chip delays*

Furthermore, the transmission delays between different chips will be much higher than the local feedback connections on the ANN chips itself. In the case of the

Stage 1 framework, internal delays of the Spikey chip require about 2 ns [47] whereas the FPGA-embedded gigabit transceivers introduce a delay of about $>200$ ns per network hop without further modifications (cf. section 4.3.4). This results in significant gaps in the transmission delays depending on the number of intermediate network hops between the source chip and the destination chip. Very large delays may not be acceptable for an adequate modeling of the given netlist. In analog to the bandwidth optimizations, the mapping algorithm has to optimize the connection delays by combining clusters of highly interconnected neurons to local groups.

### Implementation of the Mapping Process

A detailed discussion of the mapping process or the presentation of a reference algorithm is beyond the scope of this thesis. The following describes some considerations to be made for the implementation. The description of an example algorithm based on a graph model can be found in [149]. A simple algorithm of the mapping process would enumerate and map the modeled neurons one by one to the existing hardware neurons of the framework in incremental order. This clearly does not respect the given structure and topology of the hardware.

*clustering process*

The clustering of neurons is possible only for a subset of netlists. Concerning the representation of the mapping result as an adjacency matrix, the clustering process equals a modification of the matrix to accumulate enabled synapses near the diagonal by exchanging the corresponding lines and columns accordingly.[1] Clearly, fully connected networks cannot be optimized, since all synapses are enabled. Sparsely, but randomly connected networks are also hard to be optimized as the calculation example of equation 2.6 showed. The difficulty of the clustering process is caused by the random character of the network and the resulting low probability that two neurons share the same inputs. The neuron placement can be optimized best for feed-forward or recurrent networks with a (multi-)layered structure, in which the clustering process can be oriented at the layers.

*keep statistics*

To take these topics into account, more sophisticated algorithms have to be used. One option is to modify the initial netlist of the network model (e.g. to interchange synaptic connections, to remove neurons completely or to reduce the number of synaptic inputs) by keeping important statistical parameters of the network, as e.g. the average number of synaptic inputs per neuron. As an example, long-range synaptic connections can be interchanged by short-range connections to relax the link usage.

*network generation*

In the case that the mapping process results in significant losses of connectivity, it is also conceivable to characterize the neural network to be investigated by its statistical properties in terms of diversity, randomness and interconnectivity in contrast to a definition with an exact netlist. In this case, another approach is to *generate* networks that fulfill the hardware constraints automatically by the design of the generation mechanism. The mapping of such networks to the hardware can then be accomplished without any losses of neurons or synaptic connections. This strategy has been used in this thesis to generate a set of representative networks

[1] In fact, the process is more complex due to the block structure of the ANN chips and the multi-dimensional topology of the physical network.

with different bandwidth requirements to test the performance of the transport network. The generated networks have pseudo-random character and use all neurons and synapses available of a backplane (cf. section 4.10.4).

## 2.4 The Transport Network

*motivation*

It has been shown in section 2.3.1 that many of the tasks to be done during an experiment rely on a framework-wide *transport network* with high data rates and a connectivity with low latency between the several `Nathan` modules. The network is required during the setup of an experiment (e.g. to transport network stimuli to the local SDRAM memory chips) as well as during its execution (e.g. to transfer neural network data) and after (e.g. to transfer network outputs for further evaluation). The SlowControl network, which exists anyhow for configuration and control, cannot be used for these purposes, since its physical layer provides serial transmissions of up to 200 Mbit/s only. In contrast to that, the gigabit network between the `Nathan` modules features speeds of multiple magnitudes higher.

*purpose*

The remainder of this thesis discusses the design, the implementation and the evaluation of the gigabit transport network. Its main characteristics are the provision of QoS transfers combined with a compact design well suited for an implementation within the limited space of programmable logic. The transport network has initially been developed for the specific requirements that arise at the research with hardware neural network experiments. Although many design decisions of the reference implementation have been optimized for this kind of application, the network architecture features a universal character and can be used with other applications on the Stage 1 framework or even within other environments as well. This section discusses the requirements for the operation within the Stage 1 framework. The following chapter 3 describes the general architecture of the network. The reference implementation within programmable logic is presented in chapter 4.

### 2.4.1 Design Considerations

*payload data*

According to the operations necessary during the execution of a neural network experiment on the Stage 1 framework, the transport network between the `Nathan` network modules has to deliver multiple different types of payload data:

- Data concerning the neural network operation: neural network data (neuron outputs), training data (network stimuli), network answers to stimuli, synaptic weights etc.
- Other higher-level communication data: data required for the coordination of distributed algorithms, for high-level control functions or for the transfer of status and monitoring information.

*traffic classes*

Another distinction of the data to be transported is better adopted to the problem from the network point of view: the question if the data type to be transported has demands for QoS guarantees or not. In sections 2.1.1 and 2.1.2, it has been shown that the transport of ANN neural network data between the chips has strong timing demands to the underlying network to keep to the network model of the chips.

Although the Perceptron-based chip `HAGEN` and the spiking chip `Spikey` model different types of neurons and thus have different network models, both require QoS guarantees for the neural data transport: both network models require nearly constant delay and a certain guaranteed throughput or, in other words, require the neural data to be transported within *isochronous connections*. All other data types mentioned above are expected to not to have such strong delay, throughput or jitter demands. The following discussion of the data types therefore distinguishes between neural network data and non-neural network data.

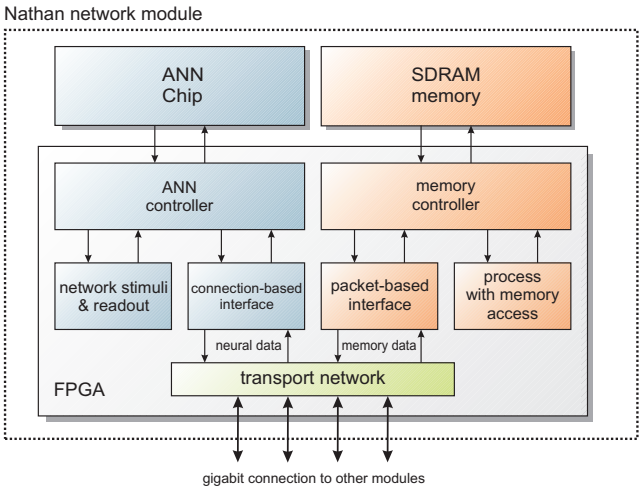



Figure 2.13: Simplified schematic of the transport network. Neural network data as well as memory data is transported between the neural network modules via the same physical gigabit links. The network provides separate interfaces to upper layer processes.

### 2.4.2 Transport of Neural Network Data

*fixed connection topology*

The requirement for isochronous inter-chip connections lead to a connection-based approach for the transport of neural network data similar to the use of circuits in circuit switching networks (cf. section 1.3). Although the transport of neural network data has stronger timing demands, the transport process can exploit the pre-knowledge of the particular network nodes that communicate and the estimation of the expected data rates: during the execution of experiments, the synaptic weights of the neural connections change, but the overall topology of the neural network itself typically remains fixed. Furthermore, the data rates can roughly be estimated since the number of neuron outputs to be transmitted between the chips depends on the pre-known network topology. Consequently, this allows to move the setup of the isochronous connections to the setup time of the experiment and keep the connections throughout its execution.

*QoS Requirements*

The service requirements for the transport network in terms of throughput, delay and jitter arise out of the neuron models and its network dynamics. The requirements depend on the implemented models of the chips and the desired speed of the experiment. Table 2.3 compares the neuron models of both chips. The QoS demands to the transport network are discussed separately for each chip in the following.

*data integrity*

An important aspect concerns the fault tolerance of the neural connections. Due

| specification | HAGEN | Spikey |
|---|---|---|
| neuron model | Perceptron | leaky integrate & fire |
| neurons | 256 in 4 blocks | 384 in 2 blocks |
| synapse weights | 10 bit+ 1 bit sign | 4 bit + 1 bit sign |
| weight stability | periodic update required | stable |
| synapse plasticity | external modification | STDP |
| network timing | clocked operation | continuous time |
| network halt | possible | not possible |
| speed | $f_{net} \leq 50\,\text{MHz}$ | speedup $10^4$ to $10^5$ to biology |
| data rate | deterministic | statistical |
| data / neuron | 1 bit / cycle | 21 bit / event |

Table 2.3: Comparison of HAGEN and Spikey with focus on the network activity.

to the parallel nature of the neural network experiments, the gigabit network has to serve the data of a large number of neural connections in parallel. If hardware errors occur, the network cannot re-request the correct data, especially in the case that the error occurred at the end point of a connection after a significant amount of time after its generation. Since the two ANN chips HAGEN and Spikey both use analog computation for the summation of the synapse currents, it is not possible to re-generate the exact neural output data.

*fault tolerance*

This leads to the fact, that the network implementation either has to be able to buffer and retransmit erroneous data by itself or to introduce sufficient redundancy, which would reduce the overall bandwidth available. Since the nature of neural networks implies some kind of fault tolerance, the design of the transport network can trade the error handling against a compact and efficient implementation, at which the appearance of rare errors is accepted. Although the acceptance of errors is unusual, the isochronicity of the connections is more important for the application.

### QoS Requirements of the HAGEN Chip

*deterministic behavior*

The network of the HAGEN chip operates at a clocked operation with the network frequency $f_{net}$. Each network cycle, each neuron transmits a single bit into the network according to its stimuli and synaptic weights. Although the bit can be interpreted as an indicator for the occurrence of spikes within a certain biological time, the neuron model is not very close to biology such that a mapping of biological time to ANN time can hardly be defined. As an advantage, the deterministic data rate as well as the fact that inter-chip connection remain fixed during an experiment allows to exactly calculate the required bandwidth between the chips prior to the execution of an experiment depending on the neural network topology to be investigated. The maximum possible execution speed is then determined by the bandwidth usage of the physical links.

*data rates*

Table 2.4 lists the bandwidth requirements depending on the selected network frequency $f_{net}$ of 10 MHz to 50 MHz for a different fraction of neuron outputs to be transported off-chip. It can be seen that th transmission of all neuron outputs

off-chip at the maximum possible network frequency of 50 MHz exceeds the transfer rate of the interface of 11.4 Gbit/s (cf. table 2.1). This limits the connectivity of the neural networks to be investigated independently of the data rate of the inter-chip transport network.

*synaptic weights*

The available data rate at the link further decreases if not only neural input and output data are to be transferred, but also the synaptic weights should be modified during an experiment. This is the case during the training phase of a neural network experiments. It is therefore considerable to train the neural network with a moderate speed and increase the network frequency at the time the synaptic weights remain fixed.

| $f_{net}$ | $\Delta t$ | fraction off-chip | | | | |
|---|---|---|---|---|---|---|
| [MHz] | [ns] | single | 10 % | 20 % | 50 % | 100 % |
| 10 | 100 | 1.25 MB/s | 32 MB/s | 64 MB/s | 160 MB/s | 320 MB/s |
| 20 | 50 | 2.5 MB/s | 64 MB/s | 128 MB/s | 320 MB/s | 640 MB/s |
| 50 | 20 | 6.25 MB/s | 160 MB/s | 320 MB/s | 800 MB/s | (1600 MB/s) |

Table 2.4: Estimated data rates for the `HAGEN` chip

*delay, jitter*

Concerning the connection delay and jitter, the clocked operation requires the data to be transported with the precision of a network cycle $\Delta t$. On the one hand, this requires a carefully designed transport, since the clock cycles are in the range of few tenth of nanoseconds at the highest possible network frequencies. On the other hand, the clocked operation also allows the `HAGEN` chips to be halted until the transport network completes the transfer of neuron data for the current cycle. Since the network operation of the `HAGEN` chip does not introduce further timing requirements, the network operation can be slowed down to a frequency at which the transport network is capable to fulfill the QoS demands. To conclude, the required QoS bandwidth, delay and jitter depends on the (arbitrary) selected speed of the chips.

### QoS Requirements of the Spikey Chip

*continuous time operation*

The leaky integrate-and-fire neurons of the `Spikey` chip are generating spike events, which are transmitted between the chips as digital event packets (cf. section 2.1.2). The fact that the `Spikey` network of the chip cannot be halted and continued without affecting the model results in more specific demands for QoS guarantees to the transport network that the operation of the `HAGEN` chip.

*speedup*

Since the analog neuron and synapse circuits operate in continuous time, not a dedicated network frequency but the configuration of neuron and synapse parameters determines the spike event rate of the chip. The more biologically plausible neuron model of the `Spikey` chip allows to define an acceleration factor (or *speedup*) that roughly compares the timing behavior of the implemented neurons to its biological counterpart. According to the configuration of the chip, the speedup can be adjusted between the values $10^4$ and $10^5$ [124, 122]. The following discussion of

| | | biological neuron | ANN neuron $10^4$ | ANN neuron $10^5$ | ANN chip $10^4$ | ANN chip $10^5$ |
|---|---|---|---|---|---|---|
| spike rate | avg. | 10 Hz | 100 kHz | 1 MHz | 38 MHz | 384 MHz |
| | peak | 40 Hz | 400 kHz | 4 MHz | 154 MHz | 1536 MHz |
| interface | avg. | - | 0.27 MB/s | 2.7 MB/s | 100 KB/s | 1 GB/s |
| data rate | peak | - | 1.07 MB/s | 10.7 MB/s | 400 KB/s | (4 GB/s) |
| inter-chip | avg. | - | 0.4 MB/s | 4 MB/s | 150 MB/s | 1.5 GB/s |
| connections | peak | - | 1.6 MB/s | 16 MB/s | 610 MB/s | (6.1 GB/s) |
| axonal delay | | 0.5–10 ms | 50-1000 ns | 5–100 ns | - | - |

Table 2.5: Expected data rates and delay requirements of the `Spikey` chip. The data rates are calculated for speedups of $10^4$ and $10^5$ and a biological mean firing rate of 10 Hz and 40 Hz. The limited ANN-FPGA interface data rate of 1.6 GByte/s inhibits the data of all neurons to be transferred off-chip at the highest possible speedup.

timing requirements ignores the existence of multiple parameters for simplification, but assumes a single speedup

$$10^4 \leq \mu \leq 10^5 \tag{2.7}$$

of the `Spikey` chip that results out of the configuration of the neuron and synapse parameters. The timing requirements are compared to the timing of biological neurons according to the selected speedup of the chip. A mean firing rate of

$$\overline{\nu}_{bio} = 10\,\text{Hz} \tag{2.8}$$

is assumed [1]. The effect of synchronized behavior or bursting is modeled by assuming a short-time mean frequency of a set of neurons of up to 40 Hz. Table 2.5 shows an overview of the expected throughput and delay requirements to the transport network depending on the selected speedups. The table also contains delay requirements. A biological transmission delay of 1 ms results in a required physical transmission delay of 100 ns to 10 ns at the possible speedups.

*data rates*

An important aspect of the `Spikey` chip is the fact that its neural spike events are generated statistically, i.e. non-deterministic. If the required bandwidth to transport the neural events exceeds the physical resources, this will not necessarily prevent the execution of the experiment, but will lead to high drop rates at the affected links. The physical bandwidth that is assigned for its inter-chip connections has therefore to be over-provisioned to minimize event losses in the case of synchronized neuron behavior. The peak values of table 2.5 denote an activity of four times the mean value. The interface rate of the chip has been calculated for the case that three events are packet into a single 64-bit interface packet. The external event contains 21 bits for address and time-stamp information. The resulting data rate for the physical links has been estimated by considering the internal 16-bit data path of the physical transceivers of the gigabit network (the MGTs within the FPGAs on the network modules). A single event therefore requires 32 bit for transmission to not to merge multiple events beyond data word boundaries. The limitation of the ANN-FPGA interface to 1.6 GB/s in single direction does only allow a simultaneous bursting of about 150 neurons or 39 % at the speedup of $10^5$. Therefore, the maximal bandwidth

requirement for neuron data of a single chip to be transported within isochronous connections sums up to 2.4 GByte/s.

*delay*

The calculated transfer times for the modeled axonal delay between two neurons is in the range of few 100 ns for the overall inter-neuron transport of spike events that has to be provided by the network. On-chip local feedback connections introduce only a physical delay of about $\leq 2$ ns, but off-chip connections require multiple processes for encoding and decoding. Furthermore, the backplane-based topology is not fully-connected but equals a 4-dimensional binary cube. The transport network therefore requires multiple physical hops to interconnect distinct ANN chips. This shows that the transport network has to be optimized for low-delay connections in any case for a biological relevant modeling of the inter-neuron connections.

*delay variation*

Due to the continuous-time operation of the chip, the delay has furthermore to be as constant as possible. Any delay variation introduced by the transport network is required to be canceled out by additional buffers within the controller of the chip according to the time-stamp within the events. The buffers not only consume programmable logic, but also increase the overall delay. Consequently, the low delay requirement to the transport network also implies the requirement for a low delay variation.

### 2.4.3 Transport of Non-Neural Network Data

*best-effort traffic class*

As described in the previous section, the execution of neural network experiments does not only require the transport of neural network data, but also several additional payload data to be transported over the framework. This includes training data as neural input stimuli or recorded output activity, as well as communication data for low-level training algorithms in programmable logic or for the software running on the distributed embedded PowerPCs. Depending on the payload type, the data to be transported can be large data blocks or small packets within continuous data streams. Unlike the transport of neural network data, the payload of this kind has no strict QoS requirements, but simply the throughput has to be maximized. Data of this kind can be combined within an additional single traffic class, whose data is transported as *best-effort traffic* using the remaining network bandwidth not needed by neural network data.

*packet switching approach*

In contrast to neural network data, the bandwidth requirements of non-neural network data are not likely to be known prior to the experiments. The transport requests are non-deterministic and arise during the execution of an experiment depending on its current state. A connection-based approach would need to implement the complex connection handling (call, setup, shut down) completely within the FPGAs. The significant delay that is introduced in the signaling phase of connections is not acceptable, particularly for small amounts of data like e.g. software messages. Therefore, a packet-based approach as it is used in packet switching networks is more feasible (cf. section 1.3).

*based on memory data transport*

The support of different payloads requires a convenient and easy-to-use interface and addressing scheme, which allows a compact implementation within the programmable logic of the FPGAs. Most processes operating on data structures store their data into the local SDRAM memory of the network module, which suggests to implement the transport of this data type based on an exchange mechanism between

the local memory modules. The implementation of a general packet transportation mechanism also allows to define higher-level data structures within the transported data and to transport different payload types without any adaptation of the packet transport network.

*error-free delivery*

Concerning the required QoS, the packet-based part of the transport network should indeed try to maximize the throughput and to minimize the delay of the delivery, but more important is the error handling. Since all best-effort data is combined into this traffic class, the network has to guarantee the error-free delivery in any case. In contrast to neural network data, the network has to re-request packets that get lost due to network congestion or physical link distortions. Furthermore, all packets have to be delivered in order.

### 2.4.4 Summary of the Service Requirements

The above discussion showed that the execution of experiments with large-scale artificial neural networks within the FACETS Stage 1 framework has requirements for two fundamentally different traffic classes to an interconnecting transport network:

*priority traffic*

- Network data with QoS requirements for throughput, delay and jitter. This traffic class is needed for the transport of data from the outputs of the artificial neurons to the synaptic inputs on distant ANN chips. The network models of the chips require isochronous connections, which implies throughput guarantees as well as guarantees in the timing of delivery. The connections can be set up early during the network startup. Data losses are acceptable to ensure a correct timing. Data of this traffic class has to be transported as *priority traffic*.

*best-effort traffic*

- Network data with QoS requirements for the data integrity. This typically includes the transport of large blocks of data. Examples for this traffic class are experimental input patterns (e.g. training data), output patterns (recorded experimental results), synaptic weights, parameter sets as well as communication data of the PowerPC software. A packet-based implementation is needed to reduce the implementation effort and to optimize the delay. Data of this kind uses remaining bandwidth and is transported as *best-effort traffic*.

Both traffic classes are required to be transported in parallel via the same physical links. Note that the acceptance of erroneous data to be delivered within isochronous connections is no hard constraint. If an error-free delivery is required, the protocols located in upper network layers can implement data redundancy or error correction functionality on their own.

*application specific demands*

Besides the two traffic classes, there are further aspects to be considered for the implementation of the transport network:

- A compact implementation within programmable logic of the existing Virtex-II Pro FPGA. This includes the usage of the existing embedded MGTs for the physical layer.
- A scalable approach. This is needed to interconnect multiple backplanes to a large system.

- A general architecture usable with different payload types. The complexity of programmable logic limits the adaptation of the network to upcoming applications if this feature is not included by design.

### 2.4.5 Existing Solutions

The following paragraphs discuss existing solutions according their ability to provide combined services with low online complexity. The provision of QoS has been discussed in chapter 1 for the two basic switching technologies circuit switching and packet switching within the scope of computer networks. It has been shown that established architectures or protocols for computer networks cannot be used:

*basic switching technologies*

- Circuit switching features guaranteed throughput and constant bit rates by reserving fixed paths between the network nodes. The problem is that the required call setup and control is complex and causes a latency, which is not acceptable for on-demand transfers of best-effort data.
- Packet switching is more flexible but requires additional headers for the routing process within each packet. Moreover, providing guaranteed services like bounded end-to-end delay is difficult and leads to large or complex designs.

However, several network architectures exist for computer networks that feature combined services:

- ATM [143] networks are well-established and currently widely used (although being more and more displaced by Gigabit Ethernet and multiprotocol label switching (MPLS) [118]). The network uses fixed-sized cells of 53 byte and provides different levels of QoS. However, it is far too complex to be implemented within the limited space of the programmable logic of the framework. *ATM*
- The work in [159, 40, 41] proposes a mixed architecture between circuit switching and packet switching to provide isochronous connections. Packets are routed along *routing trees*. A routing tree covers up to all nodes of the network and determines the pre-calculated routing decision to reach the single destination node at its root. To be able to send packets to all network nodes, multiple routing trees are periodically scheduled in *green bands* in parallel by the synchronized network nodes. If a routing tree becomes active, the network resources are reserved solely for packets to the corresponding destination. Although the architecture uses reserved resources, it is not possible to provide guaranteed rates for certain flows since contention can still occur between packets within the same tree on their way to the same destination. *Isochronets*

Concerning embedded systems or NoCs, these designs are developed and optimized for low online computational complexity and a compact, space efficient design. Two approaches are considered here that have been proposed in the literature to implement QoS within such networks:

- The work in [35] presents a router for NoCs that uses asynchronous logic to provide different levels of QoS. The design uses a fixed priority scheduler, which reserves buffer space for a particular connection using virtual channels. *asynchronous router*

Dynamic allocation of bandwidth enables the router to accommodate bursty traffic with lower end-to-end latency. The drawback is that the complexity of the design increases rapidly with the number of virtual channels. The fixed priority further may starve out flows of lower priority completely. Concerning the implementation, the router requires a speedup of two for the crossbar and is designed using asynchronous logic, which is difficult to handle with standard FPGA design flows.

*Æthernet*

- In [117, 45], a NoC design developed for multimedia applications is presented. It features a combined architecture that uses circuit switching with TDM to divide the time periodically into frames of 256 time slots. The data is transmitted deterministically between synchronized network nodes. Guaranteed services can be provided by reserving single time slots. Best-effort data uses the remaining bandwidth. The switches are input-queued switches with separate buffers for guaranteed and best-effort traffic. However the difficulty of this approach is the synchronization technique and the requirement for an efficient slot allocation mechanism, which are both not proposed. Furthermore, the fixed frame size of 256 data slots is inflexible and leads to either a high jitter or to a waste of bandwidth for connections with low bandwidth requirements.

### 2.4.6 Concept of the Transport Network

*guaranteed services*

The main issue for the transport process is the provision of isochronous connections for the neural data, which are set up at the beginning of experiments and remain fixed during its operation. Guaranteed throughput can be provided with low online complexity by the use of TDM and the periodic framing of bandwidth as in [58, 130, 23]. Framing also results in a bounded jitter and thus in the isochronicity of the connections.

*buffer-less design*

A second issue for the transport network is the required low delay. Delay is introduced by buffering, which is itself required to avoid data losses in the case of contention at the switch ports. The framing techniques mentioned above that use buffering introduce one to multiple frame times of delay per switch for the reordering process. This is not acceptable for neural data since even a frame time of e.g. 20 neural network events of each 32 bit would result in an additional delay of 256 ns at the physical clock speed of the framework (cf. section 4.3.1). The solution for the transport network therefore has been to consequently avoiding *all* buffering for isochronous data (except for the basic input and output functions of the switches). This requires a contention-free arrival pattern of the traffic as well as a global synchronization of all network nodes as in [117].

*algorithms for synchronization and resource reservation*

The proposed network architecture explicitly proposes a synchronization technique usable with arbitrary topologies and arbitrary frame sizes. Buffering is only required for the fine adjusting of delay elements in the time scale of a single time slot. Furthermore, part of the proposed switching technology is an efficient resource reservation algorithm to calculate contention-free traffic patterns that occupy up to 100 % of the available bandwidth.

*bypass-switch architecture*

The combined transfer of the two fundamental traffic classes via the same physical medium is performed by a novel switch architecture, the bypass-switch. The switch implements queues only for best-effort traffic for which it equals an input-buffered

switch with VOQs. The switch further doubles the inputs of the crossbar to remove contention between reserved traffic and best-effort traffic at the input ports. This significantly increases the performance for the latter at the presence of both classes and does not require internal speedup.

## 2.5 Summary

The chapter discussed the specific interconnection demands that arise from the research with hardware neural networks within the FACETS Stage 1 framework. It has been shown that the operation of the used ANN chips `HAGEN` and `Spikey` relies on a transport network that is capable to fulfill strict timing requirements. The main challenge for the design of the transport network is to handle the different types of data to be transported by considering their individual QoS requirements: isochronous connections for neural data and best-effort packet transfers for additional on-demand data transfers. The complexity of the transport network is bounded by the limited space of the available programmable logic.

The discussion of these requirements resulted in the concept for a network architecture that is able to fulfill these needs. The following chapter presents the specification of the developed network architecture, the *multi-class gigabit network*. The succeeding chapter 4 describes its reference implementation within programmable logic as part of the transport network.

# Chapter 3

# The Multi-Class Gigabit Network Architecture

---

*This chapter specifies the MCGN architecture and defines its protocols for operation. The first section presents an overview of the main concepts. The network architecture is able to transport guaranteed traffic as well as best-effort traffic by combining techniques of circuit switching and packet switching. The next section describes in detail how MCGN guarantees QoS for isochronous connections between network nodes by the use of three mechanisms: the network-wide synchronization, the reservation of resources and a compact and efficient online forwarding algorithm. It is further described how the novel bypass-switch architecture is able to merge packet-based transports in between the connection-based data flow. A reference implementation of MCGN is presented in the succeeding chapter.*

---

*motivation*

The design of the MCGN architecture has been motivated by the research with ANNs within the FACETS Stage 1 framework developed at the Electronic Vision(s) group as described in the previous chapter. It has been shown that the interconnection of multiple ANN chips to a large-scale neural network relies on a global network that transports different data types with different needs of QoS. *Global* in this context means network-wide, concerning the data transfer between all network modules that are connected within the gigabit network of the framework. The requirements to the network are the support of isochronous connections with low delay and jitter as well as the support of packet-based traffic. The network is designed for a low online complexity for a usage especially within the limited space of the programmable logic of the FPGAs on the network modules.

*universal architecture*

The MCGN architecture has been designed to reflect the flexible and universal structure of the Stage 1 framework, whose application is not limited to ANN research. Consequently, the developed network architecture features a universal character as well. It can be used with other applications within the presented framework and

is also a general solution for other networking environments, that require multiple traffic classes to be served.

### Characteristic Features

The characteristic features of the MCGN architecture are:

*combination of traffic classes*

1. The support of multiple traffic classes with different levels of QoS. Priority traffic is supported as well as best-effort traffic. The combination of techniques of circuit switching and packet switching ensures an optimal service for each class.

*QoS guarantees*

2. The transport of QoS traffic within isochronous connections. Isochronous connections feature nearly constant throughput and very low variations of its transmission delay. MCGN achieves good QoS results for connection-based traffic due to the avoidance of internal buffers. This makes the network architecture feasible to be used in real-time environments.

*multi-protocol*

3. The minimization of internal header processing. This allows to support multiple higher-level protocols within each class and without the need of adaptation to the protocol stack. Multiple protocols can be used in parallel.

*application layer synchronization*

4. The service of global (network-wide) synchronization for the upper layer processes up to the application layer with the precision of a cycle of a global reference clock.

*scalability*

5. A scalable approach in terms of physical line speed, the network size and the number of ports (depending on the selected scheduler, see below). The architecture can therefore be used in large networks at multi-gigabit speeds.

*configurability*

6. A wide range of configuration parameters. Frame sizes for synchronization and bandwidth reservation can be optimized for the specific working network environment. The opportunity to use the architecture with very tiny reservation unit makes it ideal for the use in embedded computer environments or systems-on-chip.

*modularity*

7. A modular design oriented along the network layers. This allows a convenient adaptation or selection of particular parts like the routing algorithm or the scheduling policy.

*universality*

8. MCGN makes only slight requirements to the physical network layer. This allows to use the architecture on a wide range of networks.

*compact online design*

9. A design optimized for the implementation in programmable logic able to be used with commercial FPGAs.

*application fields*

As a consequence of the features, MCGN is well-suited to integrate multiple protocols with different QoS demands in an embedded environment. Furthermore, the resulting low delay and jitter for connection-based traffic as well as the support of small data chunks makes the MCGN architecture most feasible for applications with strong real-time requirements. Possible application fields of MCGN are NoCs respect. system-on-chips (SoCs) in embedded environments with QoS requirements, like multi-processor environments, integrated systems or even consumer electronic

devices. Although the techniques of MCGN can be applied to computer networks as well, the administrative overhead and the requirement of its switching technology to be implemented in every node of the network may hinder its application on business or home computing areas. The reference implementation presented in chapter 4 has initially been developed for the research with hardware neural networks.

### Functional Parts

An MCGN network consists of the following parts: *parts*

1. A framing strategy combined with a novel switch type. Its switching functionality is implemented in each network node of MCGN, including the end-nodes. The switch integrates the two traffic classes - isochronous connections and packets - on the same transport medium.
2. A switching functionality to be implemented within each network node.
3. A synchronization sublayer. A global synchronization of all network nodes is required since the forwarding process of connection-based data is deterministic.
4. A connection mapping algorithm. The mapping algorithm pre-calculates the routes of all isochronous connections and calculates a reservation pattern without contention between the switch ports.[1]
5. A specification of the interface to upper network layers for the exchange of data to be transported within isochronous connections or packets. The transmission of connection-based data is limited by an admission policy.

*organization*

The processes of MCGN are located within the lower network layers, namely the data link layer as well as the network layer. The requirements to the physical layer are discussed below. The global synchronization, the connection mapping as well as the configuration of the switch is calculated off-line during a network initialization phase.

*multi-class bypass-switch*

The integration of packet-based transports and connections-based transports is performed by a novel switch type, the multi-class *bypass-switch*. It is part of the data link layer and the core component of the presented architecture. The operation of the bypass-switch has only few requirements to the physical layer and thus can be operated in a wide range of networks. The switch provides separate interfaces to the upper network layers for connection-based transports and packet-based transports. Due to that, multiple existing protocols using either of the two transport techniques can be operated in parallel on the network without further adaptation.

*organization of the chapter*

The description of MCGN continues with an overview of its architecture. The transport of payload data within isochronous connections or packets is described separately in the succeeding sections. The description has been oriented on the usual discussion of protocols along the network layer stack, but not strictly: since QoS is an aspect naturally affecting multiple network layers, the description of the isochronous connections focuses more on the principles and design techniques used. Furthermore, the MCGN architecture is specified by the functionality of its protocols and not by

[1] The mapping of connections to physical bandwidth should not be confused with the mapping of neurons from netlists to the physical neurons within the ANN chips as described in section 2.3.3.

the exact numeric definition of data structures or timing diagrams. Chapter 4 may be consulted for the reference implementation concerning the research with hardware neural networks.

## 3.1 Overview

This section introduces the MCGN architecture and its protocol stack. The section summarizes the basic concepts of the architecture and gives a first impression of the overall functionality.

### 3.1.1 Merging of Traffic Classes

*traffic classes*

The basic concept of the MCGN architecture concerns the assignment of bandwidth to transfers of the two traffic classes, which are:

1. Network traffic with QoS requirements. This traffic is transported as *priority traffic* within reserved bandwidth. The network guarantees QoS by implementing isochronous connections between network nodes.
2. Network traffic without QoS requirements. This traffic is transported as *best-effort traffic* within packets comparably to packet switched networks. Best-effort packets use remaining bandwidth not reserved or not used by QoS connections.

*design focus on priority traffic*

The design of the network architecture has been focused on an optimal transport of connection-based priority traffic. The transport of packets is expected for data, for which not the exact timing, but the overall throughput and the reliability are the transport qualifiers of interest. At some points, the design of the architecture required to balance and trade-off between the design of both traffic classes. In this cases, an optimum transport of priority traffic has been given advantage.

*bandwidth reservation*

The network architecture uses a hierarchical approach for the combination of the two traffic classes. The bandwidth reservation for priority traffic uses TDM reservation scheme. The scheme reserves the bandwidth of each physical link by dividing the time axis into succeeding *time slots* of the same duration. A fixed number of time slots is then aggregated to a *time frame*. The framing scheme is applied on every physical link in a periodic manner. Priority traffic is implemented by reserving time slots for dedicated connections along global routes between the end-nodes of the connections. Best-effort packets are not only placed into unreserved slots, but also in slots currently unused by priority traffic to enhance the overall bandwidth efficiency.

*deterministic traffic qualifiers*

The bandwidth reservation is based on a synchronization of all network nodes, which is both performed during a network initialization phase prior to the network operation. The periodic framing and slotting scheme introduces a deterministic component to the traffic management and simplifies the administration of the reserved bandwidth. The throughput of the connections between its end-nodes is guaranteed due to the bandwidth reservation. The connection delay and jitter values are calculated out of the deterministic assignment. The slotted assignment scheme also allows an efficient and compact implementation of the online switching and routing

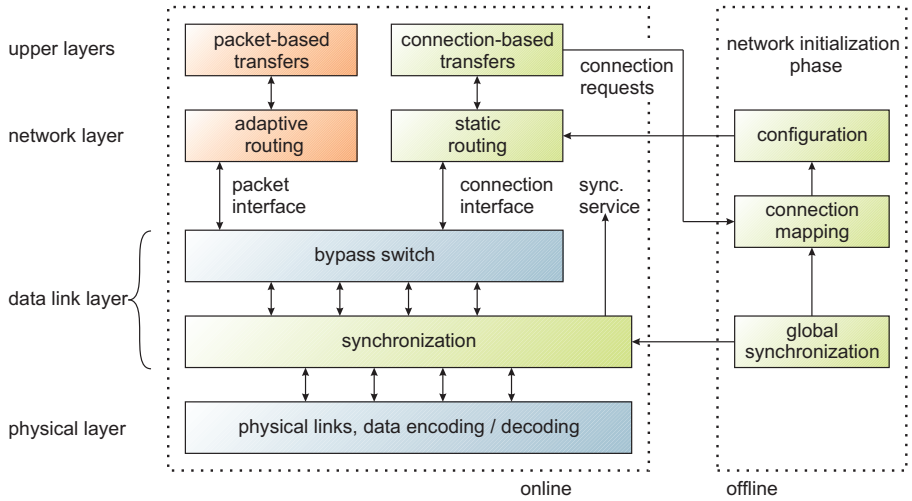



Figure 3.1: The network protocol stack of the MCGN architecture. Processes marked as offline are executed prior to network operation (without user data to be transferred). The central switch serves packet-based transfers as well as isochronous connections. Upper-layer processes may implement any network protocol using either connection-based or packet-based transfers.

algorithms as it deals with periodic traffic patterns. This results in the fact that the end-to-end delay itself is nearly constant as well, resulting in the isochronicity of the connections.

### 3.1.2 Network Protocol Stack

Figure 3.1 shows the network protocol stack of the MCGN architecture. It consists of five layers. The protocols of MCGN mainly belong to the data link layer, which implements external framing, timing and global synchronization.

*bypass-switch*

The core component of the data link layer is the novel switch type, the bypass-switch. It integrates the transport of both traffic classes, connections and packets, by combining techniques from circuit switching and packet switching. The switch has a hierarchic design. Connection-oriented priority traffic bypasses all buffering to establish isochronous connections with minimum possible delay. Best-effort packets use remaining bandwidth and are handled comparably as in common packet switching networks: Concerning this traffic class, the bypass-switch behaves like an input-buffered packet switch with VOQs, a crossbar and a multi-port packet scheduler. The bypass-switch offers separate interfaces for the service of each traffic class to protocols in upper network layers.

#### Lower Network Layers

*synchronization sub-layer*

The global synchronization, the external framing and the local timing are implemented within a lower sub-layer of the data link layer. This is required since the operation of the bypass-switch relies on a proper timing of the arrival pattern of the network traffic. Global synchronization ensures a deterministic timing of all network

nodes as well as of the transmitted frames. This is a key feature of the network, since it allows the avoidance of buffers for connection-based traffic for good QoS results.

*physical layer*

Concerning the physical layer, the MCGN architecture does not make strong requirements to the network it is used with. The physical layer merely has to provide point-to-point bidirectional links with constant data rates and the opportunity to transmit special control characters for framing purposes. This holds true for most network hardware. As an example, networks using 8b/10b encoding schemes [150] or similar techniques can be used.

#### Upper Network layers

*connection-oriented traffic*

Concerning the transport of connection-oriented traffic, the MCGN architecture requires an admission policy to be observed by upper layer protocols to control the traffic rate of data slots sent to the network. Despite of this, the usage of certain protocols or data formats is not constrained. Since the content of connection-based data is not evaluated during the switching process, arbitrary and multiple protocols that require real-time data transmissions are operated with MCGN in parallel.

*packet-based traffic*

Processes requiring packet-based transfers are implemented comparably. Since the bypass-switch uses an internal packet format, the architecture is able to serve the requests of arbitrary existing packet-based protocols as e.g. the IP protocol. As a minimum requirement, packet-based applications need to provide a small adaptation layer for the conversion of global addresses to the internal address format of the switch. No strict restrictions on the packet size or header format is made.

## 3.2 Framing Strategy

The MCGN architecture reserves network resources by applying a framing strategy on each physical link. The following paragraphs first introduce the formal description of the network topology. The description continues with the definition of the framing parameters and the discussion of an optimal selection of the parameters according to the protocols to be served on the network.

### 3.2.1 Network Topology

*definition of network nodes*

The network topology is described as a set of *network nodes* interconnected with independent *links*. The usual description of networks in the literature distinguishes the network nodes being either *hosts* or *switches*[2]. Hosts represent user end-points of the network hosting the application's functionality and are usually connected to a single switch, whereas switches interconnect multiple hosts or subnets [143, 110].

*MCGN network model*

The description used in the following makes a slight modification without loss of generality, but to simplify the discussion. The MCGN architecture uses a synchronization and switching functionality that is required to be implemented within every node of the network: intermediate switches as well as end-nodes hosting the user applications. Furthermore, switches are allowed to implement application functionality as well, hence the difference of switches and hosts is inexistent within MCGN.

*two logical nodes per host*

The network topology is modeled with the use of two parts for a single physical

[2] the following discussion does not require to further distinguish between switches and routers

host or switch, a separate node to represent the upper layer processes and to represent the switching functionality, respectively. Interconnections between switching nodes model external *physical links* whereas interconnections between switching nodes and upper layer processes model internal *local links*, i.e. internal interfaces between the data link layer and processes within upper network layers of the same host (cf. figure 3.2 as well as figure 3.3). Classical end-nodes are modeled simply by the use of a single external link.

The model also allows to construct networks without any pure switches. Instead, a network may consist of multiple interconnected hosts, each itself represented by a separate node for the switching process as well as for the local application. The topology of distributed applications interconnected by localized switches is well adopted to represent e.g. interconnected processes of NoCs or multiple interconnected FPGA devices.

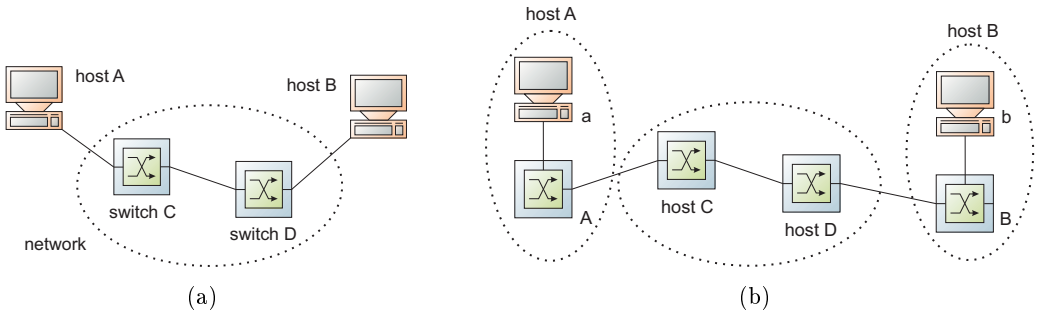



Figure 3.2: (a) Classical illustration of networks with nodes being either switches or hosts. (b) The same network in the model used for discussion. The proposed network requires each host to implement the MCGN switching functionality. The implementation of upper layer functionality is optional. Switch ports are either global physical links or local internal interfaces. All interconnections are bidirectional.

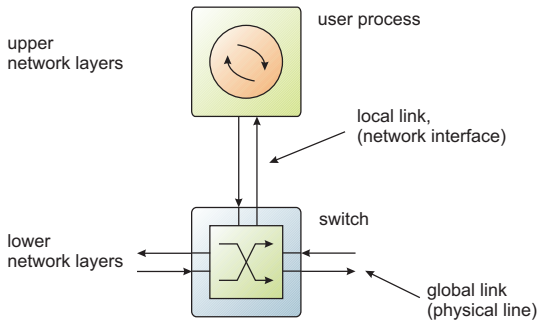



Figure 3.3: The network node of MCGN combines switching functionality and also local application processes within upper network layers.

### 3.2.2 Formal Description

The network topology is formally described as a directed graph $G = (V, E)$, with the network nodes as a set of graph vertices $v \in V$ and the interconnection links as

*graph formulation*

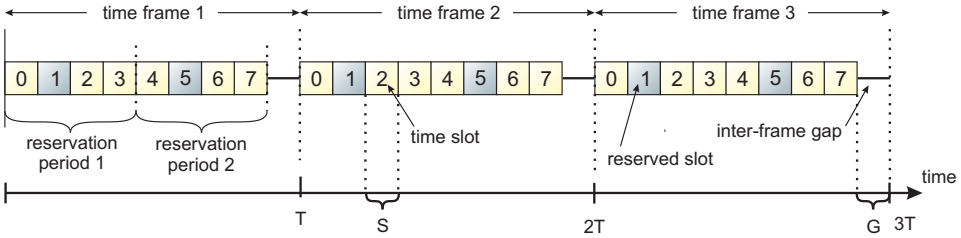



Figure 3.4: The framing strategy of MCGN uses a time division into succeeding time frames of size $T$. Each frame contains $f = m \cdot n$ time slots of size $S$ and the final frame gap of size $G$. The example parameter set equals $m = 4$ and $n = 2$. Slot 1 is reserved for priority data, which includes a reservation of slot 5 due to the size of $m$.

a set $E$ of graph edges $e$. Each edge $e \in E$ is denoted as an ordered pair $e = (v_s, v_d)$ where $v_s \in V$ is the source node and $v_d \in V$ is the destination node. All connections between any two network nodes are assumed to be bidirectional, that is G holds $G = (V, E) : \forall e = (v_s, v_d) \in E \Rightarrow \exists (v_d, v_s) \in E$.

*global and local connections*

According to the network layer stack, the set of network nodes is divided into two disjoint subsets $V_g$ of global and $V_l$ of local nodes that hold $V = V_g \cup V_l$ and $V_g \cap V_l = \emptyset$. The set $V_g$ represents the *switching processes* or *switches* of the data link layer, whereas the set $V_l$ represents the *upper layer processes* implementing the user application functionality. In analog to this, the set of edges $E$ is divided into two disjoint subsets $E_g$ and $E_l$ that hold $E = E_g \cup E_l$ and $E_g \cap E_l = \emptyset$. The set $E_g$ represents global *physical links*, whereas the set $E_l$ represents local *internal interfaces* between the data link layer and upper network layers. A valid graph $G(V, E)$ must hold the relations: $\forall e = (v_s, v_d) \in E_g \Rightarrow v_s, v_d \in V_g$ and $\nexists e = (v_s, v_d) \in E_l : v_s, v_d \in V_l$ that is, all communication between different upper layer processes uses intermediate switching processes. Since the description does not enforce the set of all global nodes $V_g$ to be fully interconnected, the routing of data along intermediate nodes may be required. As an example, the network of figure 3.2(b) is formally described by:

$$\begin{aligned} V_l &= (a, b) \\ V_g &= (A, B, C, D) \\ E_l &= ((a, A), (A, a), (b, B), (B, b)) \\ E_g &= ((A, B), (B, A), (B, C), (C, B), (C, D), (D, C)) \end{aligned}$$

*bandwidth*

It is assumed for simplification that each link between any two nodes transports the same amount of data per time, i.e. provides the same bandwidth $w$. This is no hard restriction, since physical links with different bandwidth can be modeled using multiple links of the same common bandwidth in parallel. In analog to that, the connectivity of a network application to its local switch can be modeled using multiple local links.

### 3.2.3 Framing of Bandwidth

*time frames*

The framing of bandwidth used in MCGN is based on a periodic and continuous division comparable to a TDM scheme. It is illustrated in figure 3.4. The time is

divided into periodic *time frames* of the same duration $T$. Since each link is assumed to have the same physical bandwidth, the time division equals a bandwidth division. The time frame $T$ is divided into a data part as well as an *inter-frame gap* of duration $G$. The frame gap is part of the frame itself such that the time frames follow back-to-back on each other in time. The gap takes respect to the fact that the physical layer of a network may require to insert special characters into the data stream, e.g. to detect the start of a frame as well as to store a checksum at its end. The particular data transmitted during the gap is not specified. During the data part, each frame transports the same amount of user data $F = (T - G) \cdot w$ within the fraction of bandwidth usable for data

$$w_f = w \cdot \frac{T - G}{T}. \tag{3.1}$$

The data part of the frame is divided into $f$ *time slots* of duration $S$ such that

*time slots, reservation periods*

$$T = f \cdot S + G. \tag{3.2}$$

The periodic reservation of data for connection-based traffic is made by grouping each $m$ time slots to a *reservation period* of duration $M$. The number of reservation periods of a frame is denoted as $n$ such that

$$f = \frac{T - G}{S} = m \cdot n. \tag{3.3}$$

The reservation of time slots is done with reference to a single reservation period. That is, each reservation period of a dedicated physical link gets the same reservation pattern. A connection with a single data slot $s \in \{0 \ldots m - 1\}$ reserved for the transport of its data therefore transports the bandwidth

$$w_s = \frac{w_f}{m} = \frac{w}{m} \cdot \frac{T - G}{T} = w \cdot \frac{n \cdot S}{T} \tag{3.4}$$

The reservation mechanism comprises that the maximum number of connections to be transported on a physical link at a time equals the number $m$ of time slots per reservation period.

## 3.3 Network Initialization Phase

*preparation of network functionality*

An MCGN network needs to be prepared before it can be used by applications. These preparations are denoted as the *network initialization phase* in the following. It is required for the transport of connection-based traffic only. The transport of packet-based traffic is performed purely at runtime. The initialization phase is illustrated in the figures 3.1 and 3.5. It consists of the following steps:

1. The selection of the global parameters for the framing of external bandwidth.
2. The global synchronization of all network nodes.
3. The execution of a connection mapping algorithm. The mapping algorithm reserves the required bandwidth according to the connection requests. It calculates an exclusive slot assignment pattern for each connection and each physical link of the network.

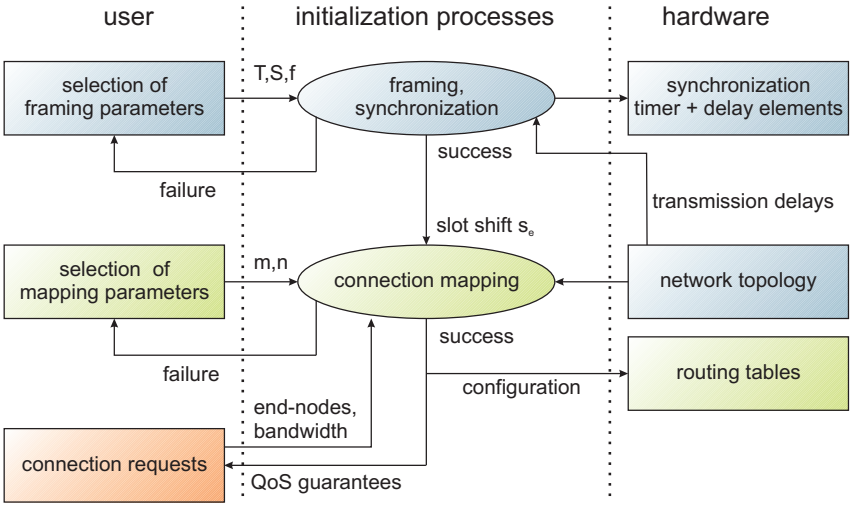



Figure 3.5: Illustration of the network initialization phase. It is executed offline and consists of synchronization and connection mapping according to user-requested constraints and hardware constraints. As a result, the routing tables of the switches are calculated (see text).

4. The configuration of the local routing tables within the switch with the calculated reservation patterns. The tables are read during the network operation by the forwarding algorithm of the switch.

*user level view*

Note that the synchronization and the configuration of the routing tables indeed require the network hardware to operate during the initialization phase, but the network is *offline* for the user during this time. To execute the initialization phase, the above tasks require a list of connection-requests in terms of end-points and bandwidth requirements by the user. After the network initialization phase is completed, the network is ready to operate. All established connections then remain unchanged throughout the operation of the network. A modification of the bandwidth reservation due to changing connection requests requires to repeat the steps three and four of the above list. Since no data can be transferred during the reconfiguration process, this equals a restart of the network. The transport of packets is performed purely online without pre-knowledge of its traffic patterns.

*complex algorithms*

The network initialization phase allows to solve the global optimization problem in software using complex algorithms. This not only increases the optimization result, but also reduces the complexity of the online algorithms and increases the scalability of the whole network. This is an important aspect for the implementation of the proposed architecture in programmable logic or the usage within SoCs.

### 3.3.1 Parameter Selection

*slot size*

The size $S$ of the time slot and thus the number $f$ of time slots per frame are in principle arbitrary and can be optimized for the application. Small values for the slot size allow for a fine reservation of bandwidth and a large number of connections to be transported via a single physical link. MCGN allows to select small slot sizes, since

it does not investigate the content of the transmitted data and uses a low-complex forwarding process.

*frame size*

Large values for $f$ minimize the waste of bandwidth caused by the frame gap, but the size of the frame may be bounded by the physical network layer (e.g. by the requirement to insert error detection codes after a finite time). The size $m$ of the reservation period depends on the complexity of the bandwidth reservation. Small values for $m$ minimize the resulting connection jitter, but may not allow to reserve data slots for all requested connections during the connection mapping process. The optimal selection will therefore tend to large frame sizes $f$ and the smallest possible values for $m$ that lead to a successful connection mapping. The influence of the framing parameters on the mapping and reservation process as well as on the resulting connection jitter is further discussed in detail in section 3.7.

*parameter updates*

The initialization phase requires to fix the framing parameters. The values are the same on all network nodes and remain throughout the execution of the network. A change in the set of the connection requests requires different interactions:

- The new connection requests require to halt the network operation during the new configuration of the routing tables. No data can be transferred during the update process.
- A valid mapping of the new requests that keeps $m$ or a required change in $m$ that keeps $f = mn$ does not require a new synchronization of the network.
- If $f$, $S$ or $T$ are to be changed, the network needs to be re-synchronized.

## 3.4 Service for Isochronous Connections

*definition*

MCGN provides *isochronous connections*, or just *connections* in the following, to serve user requests for connection-oriented data transports with guaranteed QoS. The connections are *isochronous* in terms of the fact that a constant throughput is guaranteed and the overall transmission time (the delay) is kept as constant as possible (the delay jitter is minimized). The network model does not make any assumptions about the type of payload to be transported.

*deterministic online behavior*

After the network operation started, the transport of payload data along the connections is controlled by two techniques:

- A fast, scalable and buffer-less *forwarding process*. It transports connection data throughout the network within the reserved data slots.
- A *slot admission policy*, which is to be observed at the transmit interface at the source node of each connection. It regulates the traffic sent into the network and ensures that the reserved bandwidth is not exceeded by the user application.

The preliminary calculations effectively move the switching complexity from on-line to offline. The online forwarding process is therefore of low complexity and well-suited for a compact implementation in programmable logic. The reservation scheme ensures that no physical link nor local link is congested with data.

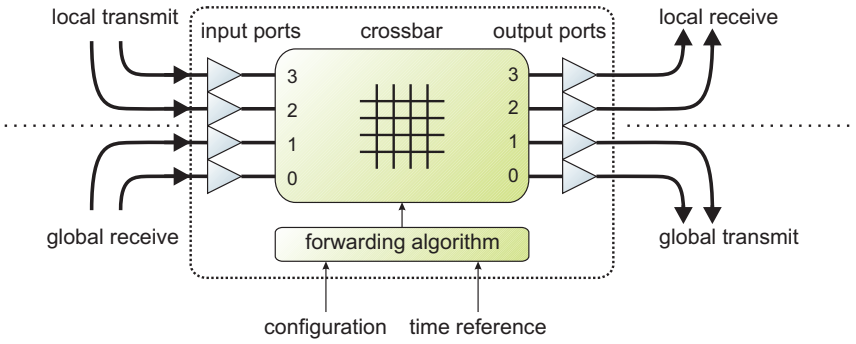



Figure 3.6: Schematic of a four-port isochronous switch with two local ports and two global ports. The input ports receive local transmit-data or global receive-data. The data at the output ports is globally transmitted or locally received. No internal buffering is performed.

### 3.4.1 Model of the Isochronous Switch

*reduction to connection-based parts*

It has been stated above that the MCGN architecture uses the bypass-switch within the data link layer to merge the two traffic classes (connections and packets). This section concerns only the connection-based traffic class. The hierarchic design of the bypass-switch as well as the fact that connection-based traffic is transported with priority within MCGN allows to reduce the following discussion of the switch model to the functional parts responsible for the forwarding of connection data. A switch implementing only the connection-based parts of the bypass-switch is called an *isochronous switch*. To simplify the following discussion, the term *switch* always corresponds to such an isochronous switch.

*topology*

An $N$-port isochronous switch features $N$ input ports as well as $N$ output ports. In analog to the above definition of network edges, each port can be either a global port or a local port. The number of global ports $N_g$ as well as the number of local ports $N_l$ equals for input ports and output ports, respectively. The input ports and the output ports are interconnected via a central crossbar. The crossbar is scheduled by the online forwarding process. The switch operates without any buffering. According to this, this is achieved without requiring internal speedup and the crossbar directly interconnects the ports. Figure 3.6 illustrates an isochronous switch with $N = 4$ and $N_g = N_l = 2$.

*interfaces*

The particular interface of the switch ports may be designed differently for global and local ports depending on the network environment. All data ports feature a common and continuous data rate. Besides of the data ports, the switch receives synchronization information about the timing of the incoming frames from the underlying synchronization sublayer during runtime. Furthermore, the forwarding process of the switch is configurable to store the reservation pattern calculated during the initialization phase.

### 3.4.2 Contention Resolution

*contention at switch ports*

It has already been stated that MCGN guarantees QoS for connection-oriented traffic by the framing of bandwidth and a reservation on a time slot basis. During each time slot, the same amount of a data (a data slot) is transported on all physical

links, local links and switch ports. However, section 1.5 showed that data losses not only occur due to the limited bandwidth of overcrowded links, but also during the internal data forwarding process of the switches in the case that data from multiple inputs has to be forwarded to the same output, which can accept data only of a single input at the same time. If this *contention* is not resolved, this results in collisions and therefore data losses.

### Proposed Solution

*deterministic arrivals*

MCGN exploits the pre-reserved determinism of the traffic arrival pattern to not only guarantee the throughput and to reduce the delay variation, but also to reduce the absolute value of the delay to a minimum. Comparable to the work mentioned above, it relies on the synchronization of all incoming and outgoing frames up to the accuracy of the duration of a time slot $S$. To be more precise, the slot numbers of all incoming frames as well as the slot numbers of all outgoing frames are transferred according to an alignment strategy (see below). This alignment is established during the synchronization process within the network initialization phase and remains stable throughout the network operation.

*contention-free assignment*

The data slot reservation pattern of every link is chosen in a way that at each switch and within each time slot, only at most one input port has to be assigned to an output port. In other words, no two input ports have data slots to be forwarded to the same output at the same time. Therefore, every data slot entering a switch at an input port is forwarded to its dedicated output port immediately without any buffering. This principle is called *contention-free assignment* or *contention-free forwarding* in the following.

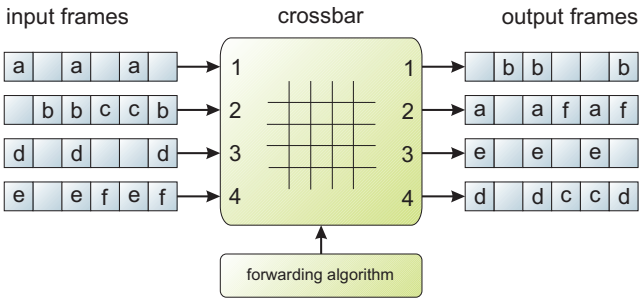



Figure 3.7: Contention resolution for reserved traffic. Example of a 4-port switch with 6 time slots per frame and a single reservation period ($N = 4$, $f = 6$, $m = 1$). Reserved slots belong to connections $a$-$f$. Conflicts at output ports are removed by the slot reservation pattern to allow the forwarding of incoming data without buffering.

*example*

Figure 3.7 shows an example reservation pattern to illustrate the contention-free forwarding process. The time frames enter the switch at the input ports shown on the left. The data slots of the frames are assigned exclusively to multiple connections named from $a$ to $f$. The reservation has been made such that no physical link has been overbooked with data, i.e. each slot is assigned to at most one connection. Furthermore, the slot assignment removed port contentions completely and the forwarding process is achieved by a look-up of the pre-calculated forwarding table.

**Advantages**

The contention-free assignment of bandwidth has the following advantages:

*low time complexity*

1. It consequently moves all possible calculations to the connection setup process at the network initialization phase. This not only allows to use calculation-intensive algorithms to achieve the best reservation pattern possible, but also minimizes the online forwarding process to a simple table look-up. To be more precise, the online forwarding process is of *constant* time, that is O(1) complexity. The low online complexity further allows for the selection of small slot durations $S$ for a fine-grained reservation pattern.

*low space complexity*

2. The forwarding process totally avoids the need of local queuing. A low space requirement of the forwarding process is an important aspect, especially if the network is to be implemented in programmable logic. Since no memory elements for buffers or queues are needed at all, the main part of the logic is reduced to the implementation of the central crossbar.

*good QoS results*

3. The determinism of the traffic pattern automatically guarantees the QoS for the connections. It obviously guarantees the connection throughput as it totally avoids data losses due to collisions. The omit of buffers greatly reduces the overall connection delay mainly to the time the data slots need to traverse the crossbar. Since the delay introduced by the forwarding process is *constant*, this makes the overall connection jitter independent from the number of intermediate network hops, which results in the scalability of the network (cf. section 3.7).

*multi protocol support*

4. Since the forwarding process does not require the look-up of headers, it operates independently of incoming data contents, which allows to transport data of multiple protocols without any adaptation to upper network layers. This greatly simplifies the installation of the switching technique within existing networks to supply users with isochronous connection services.

5. The reservation of slots for connections facilitates the combined transport of additional unreserved best-effort traffic within unreserved or unused slots on the same physical resources. The switching task for best-effort traffic can be implemented hierarchically after the reserved slots have been processed.

*calculation intensive*

The proposed design relies on an initial calculation of the contention-free reservation pattern by the connection mapping process. The complexity of the switching process is therefore moved from online to offline. Concerning the implementation, the local reordering algorithm of [130] cannot be used, since it requires the storage of all input data for the duration of a whole time frame. Although this decouples the input and the output assignment and is thus done at each node independently, it also introduces the delay of a reservation period $M$ as stated above. In contrast to that, MCGN couples the slot assignment at the input ports to the assignment at the output ports and therefore leads to a *global* assignment problem of *all* global links.

*QoS results*

The isochronicity of the connections is achieved by the guaranteed throughput due to the slot reservation, which avoids data losses in case the reserved bandwidth is not exceeded. Furthermore, the transmission delay is kept nearly constant due

to the deterministic, buffer-less forwarding process. Concerning the data reliability, error detection can be performed, but error correction cannot be provided due to the implementation of the switching process. This is no hard restriction since this can be also be implemented in the upper network layers.

### 3.4.3 Synchronization

*deterministic timing*

The global synchronization of all network nodes is established separately by the synchronization sub-layer during the network initialization phase. Its task is to synchronize the timing of the switches and thus to control the timing of the data slots transmitted between the switches. Doing so, the arrival times of incoming data slots at each switch are known in advance at runtime.

*global clock reference*

The global synchronization requires a globally unique clock source that is taken as a reference clock on all network nodes. The signal of the reference clock is distributed throughout the network either with separate dedicated signal lines or encoded within the data streams depending on its data reliability. Local clocks at each network node are derived from the global clock reference to avoid phase shifting effects at the timing between the network nodes. Each switch contains a separate timer, which is driven by the local clock. The timer is also provided for upper network layers to support the synchronization of upper-layer network processes with the precision of the global clock cycle. The synchronization layer further provides a *global synchronous signaling* service for upper network layers to facilitates the exchange of synchronized data between applications (cf. section 3.5.8). Figure 3.8 illustrates the synchronized timing of an MCGN network.

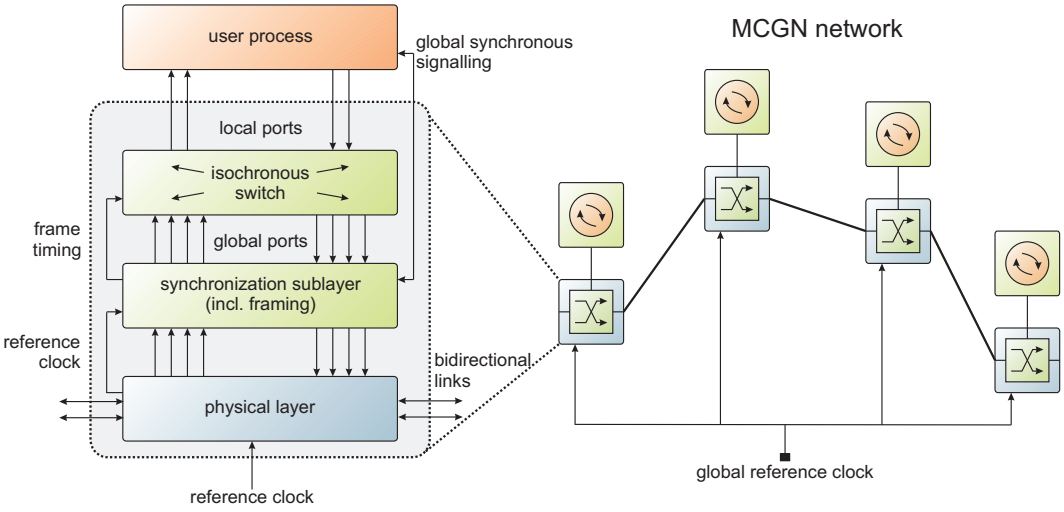



Figure 3.8: Example network of four physical nodes. The global reference clock is distributed via separate dedicated connections. The left side shows an exemplary node with four global ports and two local ports. The timing and framing is performed within the synchronization sublayer. The frame timing is provided to the isochronous switch to support its buffer-less operation.

*establishment*

To establish the synchronization of all network nodes, two tasks are required:

- The adjustment of the local timers such that the timers of all switches of the network run synchronously during operation. The value of the timer can then be taken to identify the data slot that currently arrives at the switch ports.
- The grouping of data slots to a data frame at the switch outputs. This is further adjusted according to the *synchronization condition*: The frame boundaries are selected such that the first data slot of all incoming frames at all switch input ports arrive at the local times zero. This implies a shift in the logical positions of reserved data when traveling the links of the network.

Using the two adjustments, the synchronization can be established independently of the network topology and of the selected frame size. No additional buffers are required within the synchronization sublayer except an adjustable delay element with the duration of up to the frame gap $G$.

*complexity*

The complexity to establish a global synchronization of all network nodes strongly depends on the physical layer of the network. In networking environments already synchronous, the synchronization sublayer can be far more easily implemented than in unsynchronized distributed systems, in which the synchronization is a major task. Due to this, the synchronization sublayer is described separately in section 3.5.

### 3.4.4 Resource Reservation and Connection Mapping

*inputs*

The resource reservation of MCGN is performed by the connection mapping process. The process maps the user connection requests to the physical connection resources of the network. The input to the process is the set $C$ of connection requests $c = (v_s, v_d, w_c)$ as well as the definition of the selected framing parameters, most important, the number $m$ of time slots per reservation period. Furthermore, the mapping process requires a description of the network topology including the local ports at each network node as described in section 3.2.2.

*algorithmic steps*

The mapping process calculates the global routes for each connection from its source node to its destination node via intermediate switches if necessary. For each connection, the process assigns one or multiple data slots of every link on the global path between the source and destination node exclusively for the connection according to its bandwidth demands. Since the number of time slots in a reservation period is limited by $m$, the mapping algorithm takes care not to overbook the links. The process performs the following steps:

- The quantization of bandwidth according to the connection requests.
- The routing of the connections.
- The calculation of the contention-free slot assignment for each affected link.
- The generation of the corresponding routing tables to configure the switches.

The most complex step is the calculation of the contention-free slot assignment. It can be shown that this problem equals the common *vertex-color* [13] problem from graph theory, which is known to be *NP*-hard [70].

*results*

After the connection mapping process succeeds, the requests are granted or rejected depending on its particular bandwidth requirements and the network resources available. Each granted connection is assigned its route between the network

nodes as a set $E_c \subset E$ of consecutive physical links between the source node and the destination node. The reservation pattern is denoted as a set of slot numbers $s_i \in \{0 \dots m-1\}$ of reserved slots. The set denotes the reserved time slots at the source interface of the connection at which data can be injected into the network with respect to the first reservation period (succeeding reservation periods use the slot numbers $m \leq s_i + \mathbb{N} \cdot m < f$). The forwarding tables for the switches are generated according to the calculated reservation patterns of each link. Due to its complexity, the mapping process is described in detail in section 3.6.

### 3.4.5 Online Forwarding Process

*timing*

The division of bandwidth into frames and slots allows to describe the forwarding process of each switch accordingly. The timing of the forwarding is based on a slot basis with the time reference from the underlying synchronization sublayer. Since the synchronization sublayer also performs the framing, it signals the arrival of the data slots and the appearance of the frame gap to the isochronous switch. The forwarding process of the switch can therefore be implemented without any knowledge about the framing parameters $f$, $m$ etc.

*merging of data slots*

The forwarding is performed in the same way for global and local ports (cf. again, figure 3.6). The process periodically controls the crossbar to match pairs of input ports and output ports. During each time slot, $N$ data slots arrive at the input ports and also $N$ data slots leave the switch at its output ports. Concerning a particular output port, the switch merges outgoing data slots of frames at different input ports successively into a new frame. Note that the transmission delay of data slots that traverse the switch equals for all input/output port combinations.

*low complex forwarding algorithm*

Since the switch calculates a new port mapping each time slot, its duration $S$ imposes an upper bound on the complexity of the implemented algorithm. Due to the resource reservation process, the forwarding of priority traffic is reduced to a table look-up of the relative slot positions within the frame, which can be done with only O(1) time complexity. Furthermore, the periodic assignment allows to look up the local output port of incoming data *in advance*, so that the result is known at the time the data arrives. As an advantage, the low complexity of the forwarding process allows the selection of small data slot sizes $S$ for a fine bandwidth granulation as well as it ensures the scalability of the switch for larger port numbers $N$.

*headerless, protocol independent operation*

As a second advantage, the transport of connection-based data is performed without the need of headers to mark the connection to which a time slots belongs. This does not only improve the bandwidth efficiency, but also allows the usage of small slot sizes $S$. Due to the resulting protocol-independence of the transport service, MCGN can not only serve multiple protocols independently and in parallel within a single connection, but also allows to change the protocol during the network operation without further administration.

*switching and routing*

Note that MCGN does not require to pass the connection data to the network layer at intermediate switches to perform the forwarding process. The routing information of all granted connections is already stored *into the network*, i.e. in the routing table of the switches. Since this completely reduces the online process to the switching task, an online network layer is effectively not existing within a MCGN network segment.

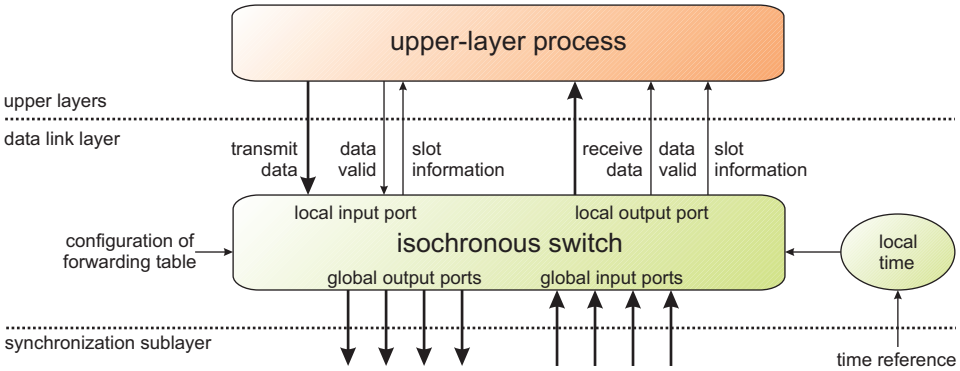



Figure 3.9: Illustration of the interface between upper network layers and the isochronous switch for connection-based traffic. The slot admission policy controls the transmission of data into the network at the source node of an isochronous connection.

### 3.4.6 Local Port Interface

*slot admission policy*

The local port of an isochronous switch is connected to the upper-layer network protocols via a dedicated interface. To regulate the transmission of local data into the network, MCGN requires a *slot admission policy* to be respected at the interface at the source node of the connection. The policy ensures that the reserved resources are not exceeded. The switch denotes the upper layer process for each connection, when its corresponding data slots are available. The deterministic forwarding scheme ensures that the data arrives at the destination automatically. The specific signals of the interface depend on the implementation. It can use a single interface for all local ports or a separate interface for each port according to the type of the upper-layer protocols. An example implementation of an interface is illustrated in figure 3.9.

#### Traffic Control

*transmission*

To transmit connection data, the switch denotes the upper-layer process of data slots that have been reserved for its connections. The reservation scheme implies that the achievable data rate is indeed guaranteed, but can also not be exceeded. The interface protocol therefore requires the upper layer process to keep to traffic smoothness conditions as e.g. the $(r, T)$ traffic model of [44]. Since excess data results in losses, the upper-layer processes may use internal FIFO queues to store transmit data to balance the network load in the case of peak rates or bursts. A traffic shaping algorithm like leaky bucket [143] is feasible to be used. This releases the user process from an over-reservation of bandwidth and results in a better overall bandwidth usage.

*receipt*

At the destination node of a connection, the switch denotes valid receive data for the appropriate connections. The upper-layer process cannot delay the receipt of the data, such that buffering has to be implemented solely within the upper network layers. The transmission delay between the time the data is sent via the transmit interface and the receipt of the data depends on the synchronization mechanism as well as on the slot reservation pattern. In the worst case, its transmission delay is constant with a variation of the duration of the frame gap (c.f, section 3.7).

#### Interface Jitter

*wait time*

The admission policy is the main origin for the resulting QoS delay variation of connection-based data. This is due to the wait time of data to be transmitted until an appropriate data slot becomes available. The time to wait depends on the number of slots reserved for the corresponding connection and the slot allocation pattern within the reservation period of size $M = m \cdot S$. The exact calculation of the wait time further has to add the duration of the frame gap $G$.

*jitter*

In the best case, the wait time equals zero. The maximum value of $M - S + G$ occurs if only a single slot is reserved and the transmit data becomes valid just after the data slot passed by within the last reservation period of a frame. If transmit data is expected to appear randomly at the interface, this results in an unpredictable delay variation of $J = M - S + G$.

*improvements*

The amount of delay variation is a measure for the isochronicity of the connection. It is improved for a dense and periodic reservation scheme, i.e. for a small number $m$ of time slots per reservation period and a short duration of the time slot $S$. To cancel the jitter completely, a user process can implement further buffering and re-synchronization to the exact clock cycle at the receive interface at the destination network node. This requires additional timing information to be submitted within the data stream of the connections.

An exact calculation of the QoS delay and jitter results is given in section 3.7. Since this requires a better understanding of the synchronization process and the connection mapping process, both tasks are described first in more detail in the two following sections.

## 3.5 Global Synchronization

*motivation*

Section 3.4 showed how MCGN uses a slotting and framing strategy as well as a contention-free bandwidth reservation scheme to guarantee QoS for isochronous connections. This further reduces the complexity of the online forwarding algorithm at the switches. The forwarding is done without the need of investigating the connection data, but depends only on the frame slot number the data arrives. A buffer-less implementation of this process requires that the slot numbers of data arriving at the input ports are deterministic at all times. This must hold for all nodes on an connections's route. MCGN uses a global synchronization of all network nodes to satisfy this requirement.

### 3.5.1 Services Provided

As shown in figure 3.1, the synchronization sublayer is achieved below the switching process on the top of the physical layer. The timing of the processes within the layers above adapt to the timing of the synchronized network. In particular, the sublayer provides the following services:

- The framing of data slots to continuous data frames according to the parameters $T$, $f$ and $S$.
- The alignment of the frames in time at the input ports of the switches to ease the switching process.

- A global synchronization service usable for processes in upper network layers.

*synchronous signaling service*

The latter service is to support applications in embedded environments or applications with real-time requirements. For that reason, MCGN provides *global synchronous signals (GSS)* to be used by upper-layer processes to transport synchronized events with the time precision of the synchronization itself (i.e. a cycle of the global reference clock). The signals can be transmitted to all nodes of the network and also between the two end-nodes of an isochronous connection only. The synchronization service is designed to be an orthogonal extension to the protocol stack. It can be added to any upper-layer protocol used with the MCGN architecture.

### 3.5.2 Setup Process

*requirements*

The difficulty of establishing a global synchronization depends on the physical layer of the application environment. This thesis proposes a general solution requiring the following conditions:

- The presence of a global time reference. At NoCs or within embedded systems, a single global oscillator is applicable. On larger networks, a clock distribution strategy is required.
- A constant delay of the physical layer for data encoding, decoding and transmission during operation at any link. Both directions of a bidirectional link must have equal delay values. Delays of different links may have different, but constant, values (e.g. different length or physical media). This includes a constant data rate of the transmissions.
- The ability of the network to verify the synchronization state, e.g. by detecting the arrival time of a data frame or the ability to transfer out-of-band synchronization characters. This task is delegated to the physical layer as it depends on the specific encoding and decoding mechanisms used.

*initialization phase*

The synchronization is established once during the network initialization phase (cf. section 3.3). The execution of the process is not time critical and independent of the network speed. Prior to synchronization, the user has to define the global frame size $T$. Due to equation 3.2, this also fixes the number $f$ of time slots per frame in the case that the data slot size $S$ is fixed. Furthermore, only discrete values for $T$ may be possible depending on the selected framing strategy and the network topology (see below). As a result of the synchronization process, the physical transmission delays $D_0$ as well as the logical slot shifts $s_e \in \mathbb{N}$ at each link $e \in E$ of reserved data are defined. The latter is used as an input to the connection mapping algorithm, which is afterward executed during the initialization phase.

*interference with reservation period*

Note that the logical subdivision of a frame into reservation periods (parameters $m$, $n$) is not necessarily a synchronization issue. A change in the values of $m$ and $n$ does *not* require a new synchronization so far as $f = m \cdot n$ still holds. A re-mapping of the connections that leads to a value $m$ not fulfilling this relationship indeed requires a new synchronization. However, the following discussion within this section reduces the description of the frame structure to the time frame $T$ or the number $f$ of data slots per frame without a loss of generality.

### 3.5.3 Overview

*local time counters*

The proposed solution uses a local periodic time counter at each switch, which is counting modulo[3] the duration of a time frame $T$. The counter does not count on incoming frame data, but runs continuously. The value of the time counter is used as a look-up index into the switches forwarding table to serve input ports and output ports accordingly. This forwarding technique requires an alignment such that the slot positions of receiving frames match the value of the counter at any time slot. To avoid variations and time shifts between the clocks of different switches during the network operation, all counters are driven by the single globally unique time reference. The global reference clock ensures that the synchronization state lasts the whole operating time of the network (cf. again, figure 3.8).

*input alignment*

The definition of a network being in synchronization state is based on the alignment of incoming frame data at the switch input ports. The physical data is usually enframed by additional delimiters encoded into the data stream to mark the start of a frame and to store a checksum at its end. The process of framing requires additional bandwidth that results in an inter-frame gap not usable for data transports. The frame gap is therefore defined as the periodic part of the physically transmitted data that does not contain data slots. The gap is located between the last slot of a frame and the first slot of the succeeding frame. Since the switch forwards the data slots of incoming frames in parallel to the output ports, this requires an alignment of the frames such that the frame gaps and the data slots are not intermixed (cf. figure 3.10).

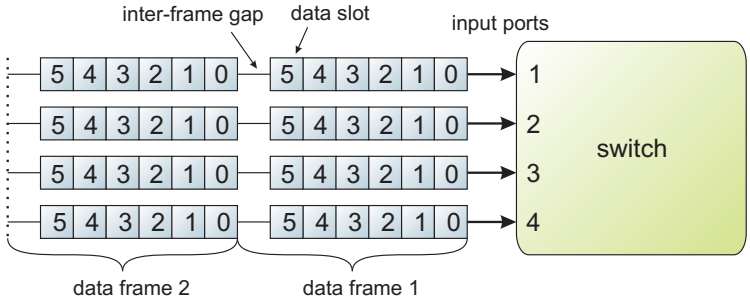



Figure 3.10: Schematic illustration of the required frame alignment at the input ports of a switch with $N = 4$ ports and $f = 6$ slots per frame. The data slots are numbered from right to left to illustrate the arrival times at the switch. The frames are aligned with respect to the frame gap for a parallel forwarding process.

This leads to the following definition for a globally synchronized network:

**Definition 3.5.1 (synchronization)** *Consider a network operating according to the proposed framing and reservation scheme. All data frames transmitted are of the same global frame size $F = f \cdot S \cdot w$, with $w$ as the common link bandwidth and the number $f$ of time slots of duration $S$. All local periodic time counters count modulo the same time period $T$.*

[3]The modulo function $a \ mod \ b$ is implemented differently in various programming languages and office tools for negative arguments $a$. This discussion uses the periodic version of $mod$ for which $(a + b) \ mod \ b = a \ mod \ b$ for all values $a \in \mathbb{Z}$, $b \in \mathbb{N}$.

*This network is in synchronization state, if the first data slot of each incoming frame at each switch input arrives at the local times* $0 \mod T$ *(input alignment).*

It is important to understand that this definition does not necessarily require the timers of the particular switches to be strictly synchronized to each other according to the usual meaning of synchronization. Definition 3.5.1 only requires to adjust the timing of the arriving data streams at the input ports to the time counters of the corresponding switches. It is shown throughout this section that it is indeed possible to achieve both, a synchronization according to definition 3.5.1 as well as a synchronization of all time counters in terms of equal timer values. Furthermore, this can be achieved with arbitrary frame sizes and arbitrary network topologies.

*adjustable delays*

To fine-tune the synchronization process, each switch features adjustable delay elements placed into the data path before each output port (c.f, figure 3.11). An adjustment of the time counter at a certain switch can be compensated by a shift of the delay elements at all links ending at the switch, but also requires a time shift of all delay elements located at the switch itself to the opposite direction. The delay elements are required to align incoming data on frame slot boundaries and to compensate for different link delays. The synchronization process therefore includes two steps:

*synchronization process*

1. The adjustment of the local periodic counters.
2. The adjustment of the local delay elements.

Two framing strategies are applicable at the switch outputs: fixed framing and shifted framing. The latter allows a fully synchronous operation of all local counters as well as a minimum value for all delay elements independent of the physical network topology (cf. section 3.5.6 below).

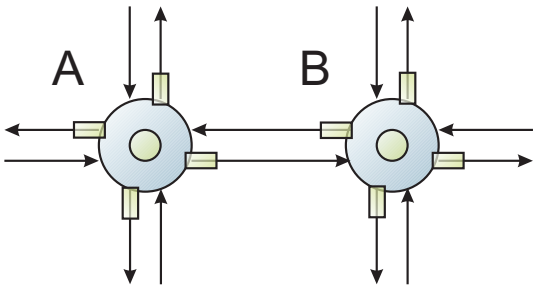



Figure 3.11: Illustration of the adjustable elements at each network node. The time counter is illustrated as a single central circle, the adjustable delay elements are illustrated as small boxes located at each link output.

*further presentation*

The following section first revises the timing of the switch according to the synchronization process. The next section discusses how the time counters of adjacent switches can be synchronized and the physical delays can be calculated. The succeeding section discusses to which values the time counters and delay elements are adjusted depending on the output frame alignment strategy. It is shown that the network can be operated fully synchronous and also use arbitrary values for the global time frame $T$ by shifting the frame boundaries at the switch outputs to match the required synchronization alignment at the switch inputs. The last section describes the GSS service to upper network layers.

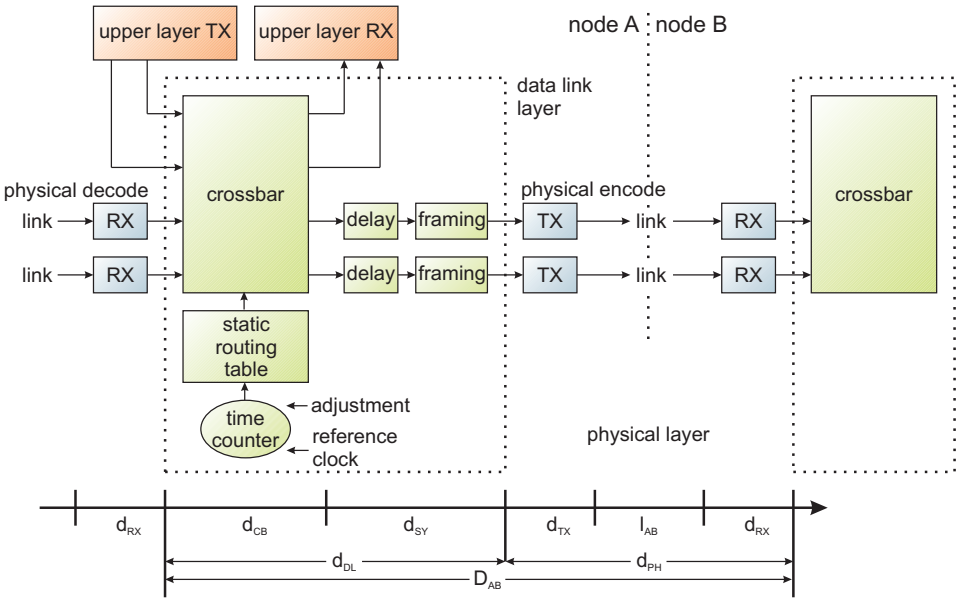



Figure 3.12: Timing scheme of connection-based traffic transported between two switches A and B (B shown only partly). The local time counter and the delay elements are adjusted during the synchronization process.

### 3.5.4 Timing Scheme of the Switch

*transmission time D*

The timing scheme of the switch is illustrated in figure 3.12. Data slots traversing the network are forwarded only within the physical layer and the data link layer. Online routing decisions are not necessary. The *transmission delay* or *transmission time* $D$ is defined as the total time between the switch inputs of two adjacent switches. To be more precise, it is denoted as $D_e$ or $D_{AB}$ for the transmission time of the link $e \in E$ from the switch A to the switch B. The values of $D$ can differ between the various links, but are constant. The transmission delay can be separated into two parts:

$$D_{AB} = d_{DL} + d_{PH} \tag{3.5}$$

The delay $d_{DL}$ of the data link layer is caused by the switching process plus the synchronization logic $d_{SY}$. The delay $d_{SY}$ includes the variable delay $\epsilon$ of the adjustable delay element plus the delay of the framing logic, which are both placed at the output ports of the switch. The delay introduced by the forwarding process simply equals the time $d_{CR}$ needed to traverse the crossbar. It is the same for all data slots, independent of its input port, output port or the connection it belongs to. The time $d_{PH}$ of the physical layer includes the delay caused by the serial data encoding and decoding ($d_{RX}$ and $d_{TX}$ in Figure 3.12) as well as the transmission time $l$ of the physical links. Since the individual parts of $d_{PH}$ are assumed to be constant, $d_{PH}$ is constant over time as well. The global distribution of the absolute values of $D_{AB}$ is mainly determined by the distribution of the link delays $l$, i.e. the physical network topology.

*determinism*

A deterministic arrival pattern over time requires that

$$D_{AB} = d_{DL} + d_{PH} = d_{CB} + d_{SY} + d_{RX} + d_{TX} + l_{AB} = const \tag{3.6}$$

for the whole operation time of the network. Due to the constant values of $d_{PH}$ and $d_{CB}$, the only variable part is $d_{SY}$ which includes the delay element. Its value is fixed during synchronization or is at least varied only by the duration of the frame gap $G$ depending on the selected framing strategy (see below). The transmission time that corresponds to the smallest possible value of the element is denoted as

$$D_0 = D(\epsilon = 0). \tag{3.7}$$

Concerning the synchronization definition 3.5.1, a constant value of the transmission delay $D$ over time results in the fact that the alignment of frames between the input ports of A and B is constantly shifted by this value.

### 3.5.5 Time Counter Adjustment

*the problem*

It has been stated that the establishment of a global synchronization requires to adjust the time counters and the delay elements at the switches to achieve an alignment at the switch inputs according to definition 3.5.1. The particular values of the counters required for proper synchronization depend on the transmission times $D$ between the input ports of two adjacent switches as well as on the selected framing strategy at the *output* ports of the switches (cf. section 3.5.6).

The following description first concerns the problem of *how* to adjust the counters. In contrast to the internal delay elements (which contain static values and can be set simply by writing the required value in it), the internal time counters change its values with every cycle of the global reference clock. An adjustment of the counters therefore requires either one of two methods:

1. The synchronous start of the counters on all network nodes after adjustment.
2. The synchronous set of the counters to dedicated values at runtime.

The difficulty of these two possibilities depends on the physical environment, the substrate and the size of the network. The first method can be achieved easily in embedded environments or NoCs by using a dedicated clock-enable signal with a known transmission time to all network nodes. In contrast to that, larger networks like e.g. WANs do not provide that possibility.

#### Timer Adjustment at Runtime

*three-step algorithm*

The following presents a more general solution for the timer adjustment at runtime. It is based on the correction of an unwanted offset of the timers of two adjacent switches. For that reason, it is assumed that a time counter can be shifted during runtime by a dedicated value. The adjustment consists of three steps:

1. Measurement of the transmission times of all links.
2. Measurement of the clock shifts between adjacent switches.
3. Adjustment of the counters according to the selected output framing strategy.

The first two steps use a bidirectional measurement between two adjacent switches. Both steps are first repeated for any pair of adjacent switches of the network until

the algorithm proceeds with step three. The measurement of the transmission times has to be executed only once for a given network, whereas step two is required at each synchronization. The time precision reached equals the precision of the global reference clock. The following paragraphs describe each step in more detail.

*bidirectional measurement*

**Measurement of Transmission Times** The bidirectional measurements are performed on the physical links of two adjacent switches. The adjustable delay elements on both switches are set zero at the beginning ($D = D_0$). Since the switches can be assumed to use similar encoding and decoding techniques on the bidirectional link, the values $D_0$ of both directions can assumed to be equal except for a small error. The local time counters on A and B are started using the same value for the time period $T$. Both switches A and B transmit frames with the frame size $T$ at each *local* time $t_A = 0$ and $t_B = 0$, respectively. Let

$$\delta_{AB} = (t_A - t_B) \mod T \tag{3.8}$$

be the time shift of the local time counters of A and B. The exact value of $\delta_{AB}$ is unknown in the beginning.

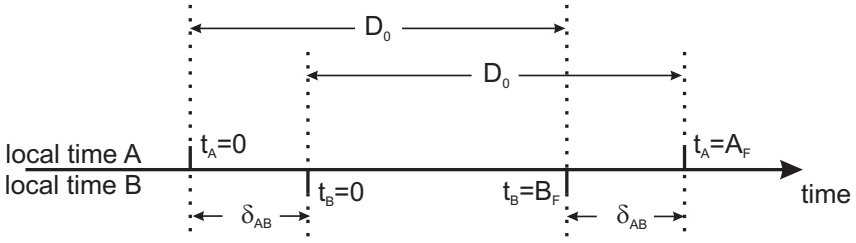



Figure 3.13: Adjustment of the local time counters and measurement of the transmission times. The initial time axes at A and B are shifted by $\delta_{AB}$. Time shift and transmission time $D_0$ are calculated using two bidirectional frame delay measurements.

Figure 3.13 illustrates the timing of the measurement. The local time of A is noted above the time axis whereas the time of B is noted below. The arrival times of the frames sent from A and B are measured at the switches B and A at the local times $t_B = B_F$ and $t_A = A_F$, respectively. The frame sent out at A arrives at local B-time $B_F$ which corresponds to $(B_F + \delta_{AB}) \mod T$ local A-time. Therefore,

$$B_F = (D_0 - \delta_{AB}) \mod T \tag{3.9}$$

holds for the frame transmitted at A, as well as

$$A_F = (D_0 + \delta_{AB}) \mod T \tag{3.10}$$

for the frame transmitted at B and measured at A. Since $\delta_{AB}$ is unknown, $A_F$ and $B_F$ can take all values $0 \leq \{A_F, B_F\} < T$. If the counters on both switches run synchronized, $\delta_{AB} = 0$ and thus $A_F$ and $B_F$ are measured to the same value $D_0$.

*calculation of transmission times*

**Evaluation** To calculate the transmission times, the frame size $T$ is set to $T > 2D_0$ of the (expected) transmission times $D_0$ to avoid ambiguous values due to the modulo

function. Doing so, the transmission time can be calculated by adding equations 3.9 and 3.10 to:

$$D_0 = \frac{1}{2} \cdot ((A_F + B_F) \mod T) \tag{3.11}$$

which is the mean delay measured. If $D_0$ calculates to a non-integer number, the transmission time in terms of clock cycles is asymmetric in both directions. In this case, one of the two delay elements has to be increased by a single clock cycle. This measurement of $D_0$ is then similarly executed on all remaining links of the network, resulting in a value $D_{0,e}$ of each link $e \in E$ of the network.

*calculation of time counter shifts*

In the next step, the frame size $T$ is set to the values requested by the user. The measurement described above is repeated. The arrival times $A_F$ and $B_F$ can again take all values modulo the requested frame size $T$. The transmission times $D_{0,e}$ are independent of the selected frame size $T$ and thus known from the first step for each link. Therefore, the time shifts of the time counters between two adjacent switches $A$ and $B$ can be computed to:

$$\delta_{AB} = (A_F - D_{0,AB}) \mod T \tag{3.12}$$

In analog to the first step, the measurement is repeated for each pair of adjacent switches, resulting in the values $\delta_{AB}$ for any adjacent switches A and B of the network.

*adjusting the time counters*

**Final Adjustment** In the last step, the counters as well as the delay elements of all switches are set to its final values. Synchronization is established by shifting the counters until the required value of $\delta_{AB}$ between adjacent switches has been reached. Again, the required value of $\delta_{AB}$ depends on the framing strategy at the output ports of the switches as well as on the network topology. Particularly, $\delta_{AB}$ does not necessarily have to be zero. The selection of $\delta_{AB}$ is discussed in the next section.

*adjusting the delay elements*

Since data in the network is transported within time slots, the local delay elements $\epsilon$ at each output port of the switches are finally set to fine-tune the alignment of incoming data to time slot boundaries to fulfill the synchronization condition of section 3.5.3. This requires an individual adjustment in the range of $0 \leq \epsilon < S$. The delay elements can also be used to equal the link delays in case of differences in the bidirectional transmission times. The correct adjustment is ensured by verifying the alignment of incoming frames at the input ports of the switches according to definition 3.5.1. Due to the usage of the global reference clock, the synchronization state lasts for the remaining operation time of the network.

### 3.5.6 Frame Alignment and Frame Size

*frame size selection*

The frame size $T$, the size of a time slot $S$ and thus the number of slots $f$ per frame are globally constant during the operation of the network. The values are fixed by the user prior to the synchronization process. On the one hand, a large value of $T$ is generally desired as it allows a fine grained QoS reservation pattern for the connections and also reduces the bandwidth waste caused by the frame gap. On the other hand, large values for $T$ are problematic on erroneous links, since large amounts

of data have to be considered as being invalid after frame errors are detected within the physical layer.

The user selects the optimal framing constants $T$, $S$ and $f$ according to the number of connections required for the application. After $T$ has been selected, the selection of the slot size $S$ and thus the number of time slots $f$ per frame is independent for the synchronization and not further discussed here. Both values can be chosen freely by the user so far as equation 3.2 is fulfilled according to the required bandwidth granulation. The selection of $T$ is also constrained by the physical transmission times $D$ of the network, i.e. the physical network topology, which is discussed in the following.

*globally coupled timing*

The constrains for the framing constants result out of the method of contention-free buffer-less forwarding of synchronized frames. An important focus on the development of the switching process has been to reduce the overall connection delay. Buffers within the switch have been avoided in front of as well as after the central crossbar. This not only requires incoming frames to be aligned according to the synchronization definition 3.5.1, but also determines the timing at the output ports. Since the total internal forwarding delay of the switch is constant, this effectively couples the input port timing to the output port timing (which is the input port timing at the next node). Concerning the whole network, this results in a globally coupled timing of *all* network nodes.

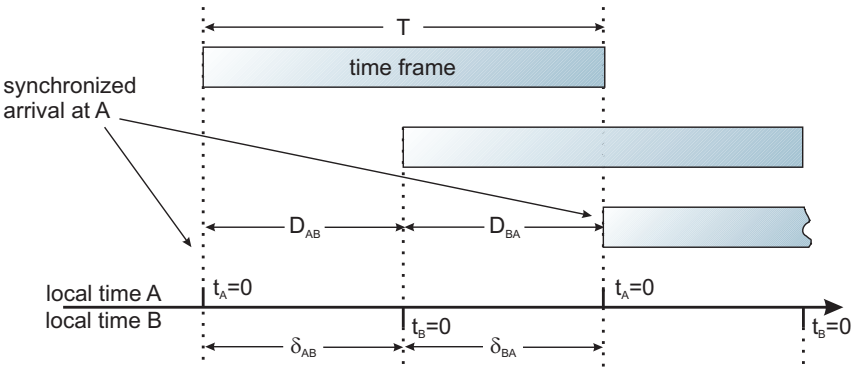



Figure 3.14: Dependency of frame size $T$, transmission time $D$ and time shift $\delta$ between two adjacent switches A and B for the case $T = 2D = 2\delta$. A frame sent from A to B and back to A arrives at both switches properly aligned according to the synchronization definition.

*alignment constraints*

Consider figure 3.14 that illustrates the timing of the synchronization of two directly connected switches. The forwarding process requires an alignment of frames at the input ports of *both* switches according to definition 3.5.1. This results in a dependence of the shifts of the local time counters of both switches and the transmission times $D$ in between. Since this condition has to be met for frames transmitted in both directions, this reduces the possible values for $T$ as well as for the shifts $\delta$ between the two time counters to discrete values according to:

$$\delta_{AB} = D_{AB} \mod T. \tag{3.13}$$

The following paragraphs discuss the synchronization process with focus on the selection of the frame size $T$ as well as the selection of the time counter shifts $\delta$

*framing strategies*

in dependence of the physical transmission times $D$. Two framing strategies are considered:

- When using *fixed framing*, the grouping of data slots to a frame is preserved. Data entering the switch at a particular slot number leaves the switch within the same number. Fixed framing does only allow discrete values for the framing parameters $T$ (or $f$, $S$) depending on the network topology.
- To increase the framing flexibility, the grouping of data slots to a frame can be changed at the output ports of the switch. In this case, data slots of two parts of succeeding frames are aggregated to a new frame, which shifts the frame boundaries, but allows arbitrary values for $T$ ($f$, $S$). This method is referred to as *shifted framing* in the following. It has the effect that the positions of the reserved time slots change when traversing the switch. The number of shifted positions is denoted with $s_e$ for each link $e$.

Both framing strategies are considered in the following. The discussion starts with the more simple case of keeping the frame boundaries.

### Fixed Framing

Fixed framing keeps the frame boundaries during the forwarding process. In this case, a proper synchronization between two switches A and B according to definition 3.5.1 requires equation 3.13 to hold. In principle, the transmission time $D_{AB}$ can be tuned by modifying the value $\epsilon$ of the local delay element to allow for larger frame times $T$. However, additional delay in the data path is generally unwanted and purpose of the elements is to fine-tune the synchronization.

*time counter shift*

The bidirectional nature of the network requires equation 3.13 to hold not only for the link from A to B, but also for the link from B to A. On this link, $\delta_{AB} = (-\delta_{BA}) \mod T$. Since $0 \leq \delta < T$, the allowed values of the time shifts between two adjacent switches reduce to

$$\delta \in \left\{0, \frac{T}{2}\right\}. \tag{3.14}$$

*frame size constraint*

The transmission times in both directions can be assumed to be the same, thus $D_{AB} = D_{BA}$ (inequalities in the transmission times are to be equalized using the adjustable delay element $\epsilon$). Let $D = D_{AB} = D_{BA}$ be the common transmission time in both directions, then the constraint for $T$ can be written as:

$$2D \mod T = 0 \tag{3.15}$$

or

$$T = \frac{2D}{n}, n \in \mathbb{N}. \tag{3.16}$$

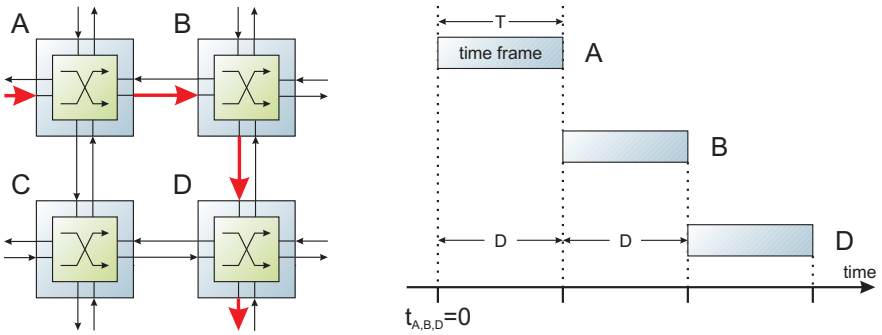
*frame size selection*

Synchronization requires equation 3.16 to hold globally, which limits the (global) frame size $T$ depending on the (local) transmission times $D$ between two adjacent switches. The fixed framing method has the disadvantage that the value of $D$ may be rather small depending on the physical layer, especially in NoCs or embedded systems. On networks with large values of $D$, this is not an issue. A limited value of $T$ limits the number of possible time slots $f$ of size $S$ aggregated to a frame and

results in a low granularity for the QoS reservation pattern. Large values of $T$ require both, $D_{AB}$ and $D_{BA}$ to be artificially increased via the delay elements $\epsilon$, but this reduces the proposed framing and buffer-less forwarding scheme to absurdity.

*simple setting using fixed framing*

**Example 1:** Figure 3.15 shows the synchronization case for a network with regular 2-dimensional meshed topology, $D = T$ and $\delta = 0$ for all switches. In this case, all periodic counters run strictly synchronous and the same slot number appears at all inputs of all switches at the same time. All switches belong to the same timing diagram of Figure 3.15(b). Since $T$ is a globally unique number, this also enforces $D$ to be globally of the same value. In the case of different physical link delays $D_{AB}$, the delay elements have to be adjusted such that $D$ is set to the globally same value of $D = \max(D_{AB})$ that is the worst case transmission time of the whole network.




(a) (b)

Figure 3.15: (a) Regular network topology with similar transmission times $D$. The route of a forwarded frame is shown. (b) Timing diagram of the forwarding process. The network is synchronized with the global frame size $T = D$ and $\delta = 0$. All nodes operate strictly synchronous.

*largest frame size with fixed framing*

**Example 2:** A different setup usable with the same 2-dimensional meshed network is shown in Figure 3.16. The network nodes are separated into two disjoint classes A and B such that not two switches of the same class are directly connected. The example uses a time shift of $\delta = D$ and a frame size $T = 2D$, which is the largest usable value. As a disadvantage, the network nodes are not all globally strictly synchronized, i.e. the time counters run only synchronous within the same class A or B of network nodes. Large frame sizes $T$ are better implemented using the shifted framing technique that is described below.

### Shifted Framing

*synchronization condition*

Equation 3.16 limits the selection of the frame size $T$ to discrete values if the frame boundaries are preserved. The constraint arises out of the synchronization condition that has to be fulfilled on all network nodes according to definition 3.5.1. The timing at a certain switch input is determined by the synchronization condition at the *local* network node, whereas the timing at the switch output is determined by the synchronization condition at the *destination* node. The synchronization conditions

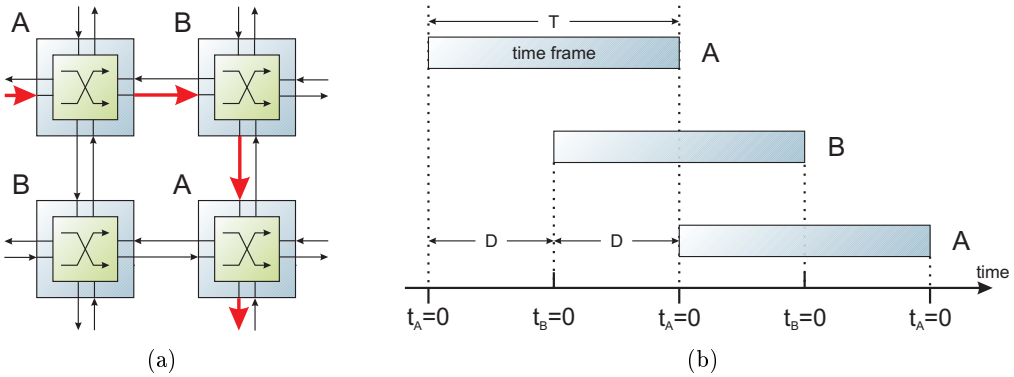



Figure 3.16: (a) Division of the regular network topology into 2 disjoint classes A and B of nodes. (b) Synchronization using the largest possible frame size $T = 2D$. The switch internal time counters of nodes A and B are shifted by $\delta = D$.

could therefore not be met on both switches with a simple forwarding of the received frames values for arbitrary values of $T$.

*moving the frame boundaries*

To overcome these constraints, shifted framing moves the frame boundaries of two succeeding frames such that the same number of time slots is grouped to a frame, but the slot positions shift. This can be done since the grouping of slots to frames is an administrative issue only required for the physical transmission, but not required for the application. In fact, the user application does not have to be informed about the certain slot positions in which the data is transported. Concerning the online switching process, the time slots of frames at different input ports are even merged into new frames at the the switches output ports (c.f, section 3.4.5). For that reason, the framing process at the output ports can also aggregate data slots of parts of succeeding frames at the same output port to a new frame. The shifting process keeps the order of the data slots as well as the globally constant frame size $T$. It is illustrated in figure 3.17.

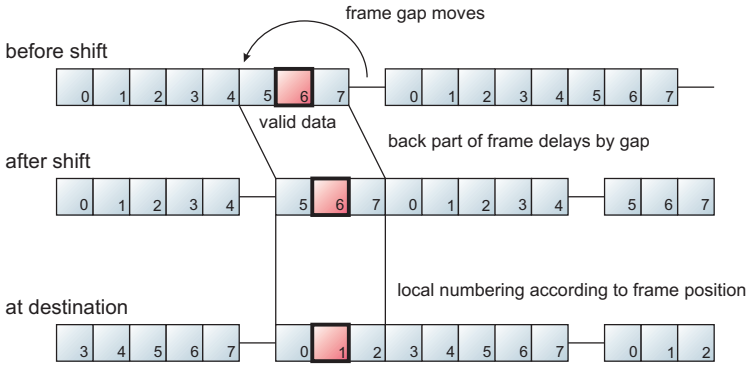



Figure 3.17: Schematic of the shifted framing technique. (top) To cope to the synchronization condition, the gap is moved three slot positions to front. The back part of the frame has to be delayed. (middle) Moving the gap has shifted the frame boundaries. (bottom) After having arrived at the destination, the frame slots are numbered by its position, thus she logical slot position of the transmitted data has been shifted by three slots modulo the frame size.

*formal description*

To give a formal description, let $s_{AB} \in \mathbb{N}, 0 \leq s_{AB} < f$ be the logical slot position shift for the frame transmission on the link from switch A to B. This means that data placed in data slots $i = (0, .., f-1)$ when entering switch A leaves the switch placed in the slots $(i + s_{AB}) \mod f = (s_{AB}, .., (f - 1 + s_{AB}) \mod f)$.

*effect on transmission time*

Note that moving the frame gap results in the fact that the absolute value for the transmission times $D_{AB}$ differs between the slot positions of the two frame parts. Data slots that enter the switch at slot positions $i \geq f - s_{AB}$ get an additional delay of the gap time $G$ in contrast to data slots entering the switch at slot positions $i < f - s_{AB}$. The two transmission times are denoted as $D_<$ for the first frame part and $D_>$ for the latter frame part with $D_> = D_< + G$. Both transmission times $D_<$ and $D_>$ are still constant over time. Figure 3.18 illustrates the shifted framing process for a slot shift $s_{AB}$ of 2.

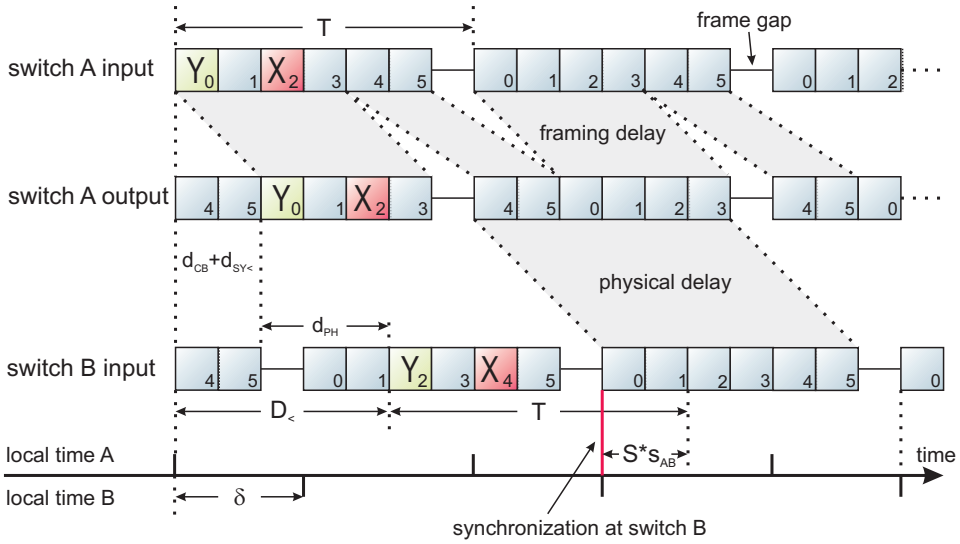



Figure 3.18: Example of shifted framing between two switches A and B with $f = 6$ data slots per frame and a logical slot shift of $s_{AB} = 2$. The output framing of switch A has adjusted such that the frames arrive at switch B correctly aligned according to the synchronization condition without any additional buffering. The numbers denote the logical slot number at each switch according to its local time. The slot number (frame position) of reserved data slots is shifted by $s_{AB}$ while traveling from A to B. The frames arrive correctly aligned at the switches although its local times are shifted by $\delta \neq 0$. The two data units $X$ and $Y$ experience the same transmission delay.

Comparable to equation 3.13, the following condition is required to hold for the timing of any two directly connected switches A and B:

$$\delta_{AB} = (D_{AB,<} - s_{AB} \cdot S) \mod T \tag{3.17}$$

*selection of $T$ and $\delta$*

Since the value of $s_{AB}$ can be chosen freely and independently for each switch output, shifting the frame boundaries effectively decouples the local switch timings from the global frame size. This results in the fact that the shift $\delta$ between the two local time counters is not longer dependent of the transmission time, but can be selected freely. As a result of this, shifting the frame boundaries results in two advantages:

1. The selection of an arbitrary frame size $T$ independent of the network topology.

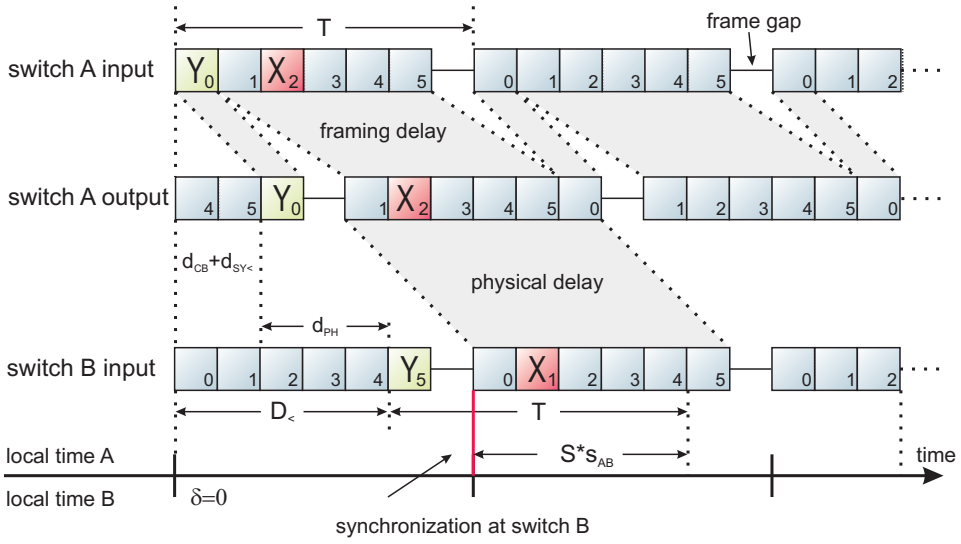



Figure 3.19: The same two switches of figure 3.18. Since the local time counters of both switches have been synchronized to each other ($\delta = 0$), the logical slot shift $s_{AB} = 5$ is required to keep to the synchronization condition of the framing scheme. The back part of the frame experiences an additional transmission time of the duration of the frame gap.

2. The switches can operate strictly synchronized, i.e. $\delta = 0$ between all switches.

The second point can be fulfilled by setting $s_{AB}$ according to:

$$s_{AB} = \frac{D_{AB,<} \mod T}{S} \tag{3.18}$$

In this case, the two switches are operating synchronized with $\delta = 0$ independent of the selection of $T$. The delay element at the affected output can be reduced to the minimum possible value such that $D_{AB}$ is an integer number of a time slot $S$. Figure 3.19 shows the selection of the slot shift $s_{AB}$ of 5 for the two switches A and B from figure 3.18 when using the same frame time $T$ but with synchronized switches $\delta = 0$.

*example using shifted framing*

**Example 3:** Consider Figure 3.20. The input data at node A is aligned at inputs according to the synchronization condition. Slots 0 and 3 are reserved for isochronous connections (port a). Both slots are forwarded on different paths. Slot 0 is forwarded via node B (ports b, d). On this path, $D = T$ and shifted framing is not necessary. Therefore, the data belonging to this connection occupies slot 0 also at node C (port e). The connection for which slot 3 at node A is reserved is routed directly to node C (port c). Since the physical delay between A and C is larger, shifted framing is necessary to keep the delay element small, which results in a shifted logical slot at the arrival at node C (port f). Both connections arrive properly aligned at node C according to definition 3.5.1. The relative slot numbers have been changed due to the different transmission times.

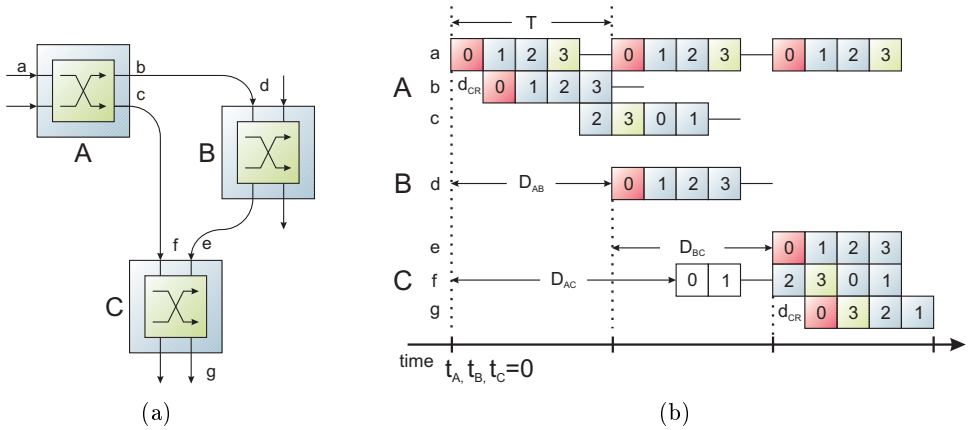



Figure 3.20: (a) Network topology example. The different transmission times require shifted framing on the path AC but not on AB and BC. Only links in single directions are shown. (b) All nodes are strictly synchronized. Although the slot numbers are shifted at output port c, no additional physical delay is introduced. Node C receives data from nodes A and B properly aligned. Frames are shown only partly for a better understanding.

### Framing Summary

The buffer-less forwarding of the switches requires the correct alignment of incoming frames at the input port of the switches according to definition 3.5.1. It has been shown that this alignment can be established independent of the network topology. The coupling of the timing between the input ports and the output ports of the switches can be solved by a shifted framing strategy at the switch outputs. The advantages are the following: *advantages*

1. The selection of arbitrary frame sizes $T$ and thus the number of time slots per frame $f$, which allows to select an optimized slot reservation granularity $S$.
2. A globally strict synchronized adjustment of all local time counters ($\delta = 0$) independent of the network topology and physical transmission times. This is important for the provision of a synchronization service to upper network layers (see below).
3. The value of the adjustable delay elements that are required in the data path to cope with different link delays can be reduced to the duration of less a time slot $S$ compared to values up to $T$ at the use of fixed framing. This ensures the minimum possible overall end-to-end delay of the connections.

The shifted framing process involves also two slight disadvantages: *disadvantages*

1. The process requires additional logic in the data path at each output port. This has to be considered if an implementation within programmable logic requires limited complexity.
2. The process results in different transmission times $D_<$ and $D_>$ for the two frame parts with the slot positions $i < f - s_{AB}$ and $i \geq f - s_{AB}$, respectively. This may result in additional connection jitter up to the size of the frame gap if the data frame is further divided into multiple reservation periods with a

periodic reservation pattern or in the case that multiple time slots are reserved for a connection (cf. section 3.7).

On networks with regular topology and comparable physical link delays, shifted framing is not necessarily needed since $T$ can be chosen to be $D = \max(D_{AB})$ according to example 1.

### 3.5.7 Synchronization Result

*slot shifts*

The global synchronization requires the selection of the slot shifts $s_e$ of each link $e$ that denote the shift in the logical slot number as described in the previous section. If fixed framing is used, the shifts equal $s_e = 0$ on all links. In this case, connection data placed into a specific data slot at the source node stays within this slot position on the whole route. If shifted framing is used, the slot shifts $s_e$ are fixed during the synchronization process to values different from zero depending on the network topology and the selected frame size $T$. The values $s_e$ are further required in the next stage within the network initialization phase, which is the global connection mapping process. The connection mapping process uses the shifts to keep track of the data slot positions that are occupied by the connections to compute a global contention-free slot assignment for all connections (cf. section 3.6).

*formal description*

To give a formal description, the vector representing each link $e \in E$ is extended by the slot shift $s_e$ required for synchronization: $e = (v_s, v_d, s_e)$. If shifted framing is used, the values $s_e$ are defined according to the following rule:

- If $v_s \in V_l \rightarrow s_e = 0$ (at the connection source node)
- If $v_s \in V_g \rightarrow s_e$ is defined as the slot shift between the input ports and the output ports of the switch at $v_s$ after synchronization (at intermediate nodes).

If shifted framing is not used (fixed framing), $s_e = 0$ for all edges $e$ as stated above.

### 3.5.8 Upper-Layer Synchronization Service

*synchronization sublayer*

The previous sections proposed methods to establish a global synchronization of the network nodes. Due to the fact that the local time counters are driven by the same global reference clock, the synchronization accuracy can be reached to the precision of a single cycle of the clock. Concerning the layered description of the network, the synchronization layer is implemented in the lower sublayer of the data link layer and executes framing and timing mainly for the switching process above and particularly transparent for upper-layer processes (cf. figure 3.1).

*motivation*

However, upper-layer processes may also require global synchronization information. The establishment of synchronization between upper-layer processes without an explicit synchronization layer below is difficult to implement and relies strongly on the physical environment. In embedded environments or SoCs, local electronic signals can be used, whereas in large scale networks, a common way to synchronize network peers is the usage of a high-level protocol like the network time protocol (NTP) [93], which provides only limited precision to about a few microseconds. For networks already synchronized, it is feasible to use the existing timing information for a general service to processes within upper network layers (compare e.g. the synchronous protocol stack (SPS) of [159]).

*generic synchronization service*

This section proposes a method to provide such a general synchronization service to upper-layer processes based on the MCGN network synchronization. It exploits the inherent timing of the network and releases the applications from implementing complex timing protocols for its own. The interface is kept simple, but allows to control globally timed signals up to the precision of the synchronization itself, which is the global clock cycle. The service can be used in multiple ways, e.g. to initialize local clocks or to synchronize upper-layer operations.

### Synchronous Signal Events

*types of signals*

The synchronization service is based on the transmission of timed signals that pass the network. Depending on the purpose, two different types of signals can be used: *connection synchronous signals (CSS)* and *global synchronous signals (GSS)*. CSS are used to synchronize the timing between the user processes at the two end-nodes of a connection. GSS are used for synchronization between all nodes of a network. Signals are raised locally and forwarded through the synchronized network. After the signal has been passed the network, a *signal event* is triggered locally at all affected destinations.

Each signal (either of CSS or GSS) is passed with a pre-defined signal identification number (ID) to mark its purpose. The IDs of all signals are application-specific and have to be defined prior to network operation. A possible usage may be the synchronous initialization of common (non-periodic) timers at different network nodes. Further operations can later be scheduled with respect to the timer values.

### Signal Transport

*deterministic timing*

The signal transport exploits the deterministic timing of the synchronized network. Although the local timers count with the global frame period $T$ and thus the local time is known only modulo $T$, section 3.5.6 showed that synchronization also fixes the transmission time $D_{AB}$ between two directly connected switches to the precision of the global clock cycle. Since a synchronized network follows definition 3.5.1, the forwarding of signals can also be based on the time slots of the external data frames. Independently whether shifted framing is used at the output ports or not, data placed in certain time slots $s$ at an output link $e$ arrives at the time slot $(s + s_e) \mod f$ at the destination switch input ($s_e = 0$ if frame boundaries are fixed). To not affect the guaranteed rate of the isochronous connections, the signal is encoded as side-band data within a small reserved part of the frame gap.

*signal transport*

The signal transport works as follows: After a signal of a specific ID is raised, a local down-timer is initialized with a time value larger than the transmission time of the farthest global path between any two nodes of the network. This value can be computed easily during the network initialization phase after all directly connected transmission times are known. The signal is distributed throughout the network by the following rules:

1. Each signal is sent together with its signal ID and the value of the time counter.
2. Each node receiving a CSS or GSS forwards the signal accordingly. A CSS is forwarded within the pre-reserved slots belonging to the connection. A GSS is forwarded to all outgoing ports except the receiving one.

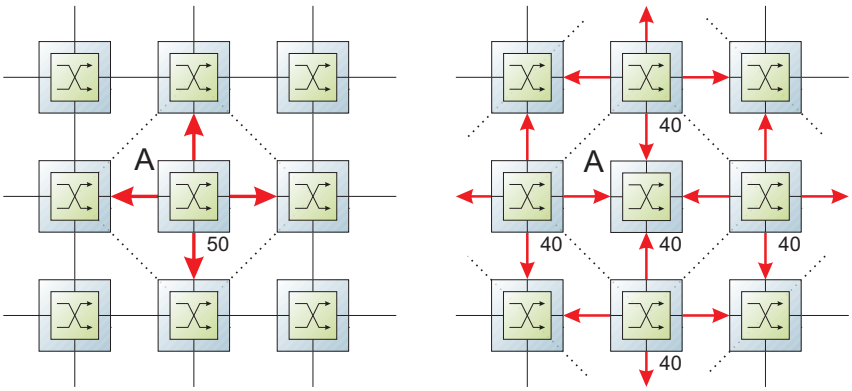



Figure 3.21: Example forwarding of a GSS along a 2-dimensional meshed topology. Node A raises the signal that is forwarded via multiple hops through the network while the timer decreases in time and space. After having arrived at all nodes, the signal event is triggered at all nodes at the same time.

3. The received time value is used to initialize a local down-counter corresponding to the received signal ID. For each output port, the counting value is reduced before the signal is forwarded by subtracting the transmission time $D_{AB}$ of the actual node A and the target node B corresponding to that output port.
4. Arriving signals for which the corresponding down-counter is already initialized have taken a roundabout way to the switch and are ignored.
5. After the local down-counter finishes, the signal event is triggered locally, i.e. the event and its ID are denoted to the upper-layer process.

The reduction of the count value effectively computes the signal event time in the time domain of the next node before forwarding the signal to that node. In other words, the signal timer decreases in time and space during the forwarding process. A node may receive the same signal from multiple adjacent nodes from which it arrived after traveling different paths. This is not a problem as both signals may indeed arrive at different times, but denote the same absolute event time, such that further signals can safely be ignored. Figure 3.21 illustrated the forwarding process for a GSS.

### Upper-Layer Interface

*generic interface*

The interface to upper-layer processes is kept simple for a generic usage with software or hardware implementations. A user process raises a signal of a certain ID and waits until the synchronization layer denotes its local trigger time. Local event triggering does not have to distinguish between signals initially raised at far nodes or at the local node itself.

An example implementation is described in the following: A GSS or CSS is raised via the *raise* signal together with its appropriate ID. The synchronization service answers the request with an *acknowledge* or *error* signal depending on the synchronization state and the signal ID. At the time, the GSS or CSS arrives, the local *event* signal is triggered together with the signal ID. The interface is illustrated in figure 3.22.

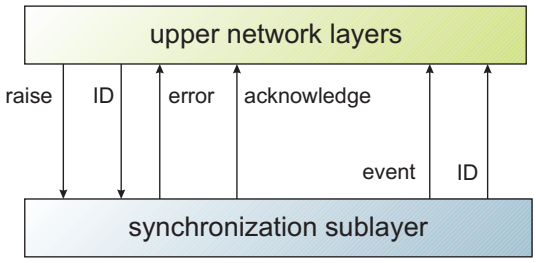



Figure 3.22: Illustration of the proposed GSS or CSS interface. The synchronization service is an orthogonal extension to the existing protocol stack that exploits the inherent synchronization of an MCGN network segment.

### Accuracy and Error Handling

*event time accuracy*

Since the signal forwarding relies on the global synchronization, the accuracy of the triggered events depends on the synchronization accuracy itself (cf. section 3.5.5). Depending on the network topology, a node can receive the same signal event multiple times, transported on different paths. The signal received earlier can be expected to have crossed a fewer number of intermediated network nodes and thus to be more reliable, such that signals arriving later are simply ignored. However, as the timing error of two adjacent nodes is in the range of the global clock cycle, the overall error will be less than a frame time $T$ or even less than a slot time $S$. Exact values depend on the physical accuracy of the timing of bidirectional links as well as the global clock cycle frequency of the network and the selected slot time.

*conflicting raises*

Multiple events with the same ID raised at different nodes at small different times result in misleading network-wide information about when the event has to be triggered. In that case, the synchronization logic denotes the requesting processes that the desired event ID has already been raised by another process and asserts the local error signal. However, race conditions, at which two nodes raise the same event ID at a time difference smaller than the link transmission time cannot be avoided by the current design. On the one hand, this can easily be avoided by a global administration and exclusive reservation mechanism for signal IDs, e.g. due to an individual assignment of IDs to each network node. On the other hand, this expands the simple and compact design with further complexity. The GSS or CSS signaling feature is primarily designed to add simple synchronization features to application processes without affecting other existing protocols. Synchronization mechanisms more sophisticated can be implemented on top of the proposed service if needed.

## 3.6 Connection Mapping

*reservation of network bandwidth*

The *connection mapping* process reserves network bandwidth resources by mapping connections to physical network links. For each connection, the process assigns a certain number of data slots at fixed positions on all intermediate links of the route between its end points exclusively for this connection according to its end-to-end QoS requirements.

*global problem*

Section 3.4.2 showed that the forwarding process requires a correct ordering of the reserved data slots to simplify the online switching and routing process. The

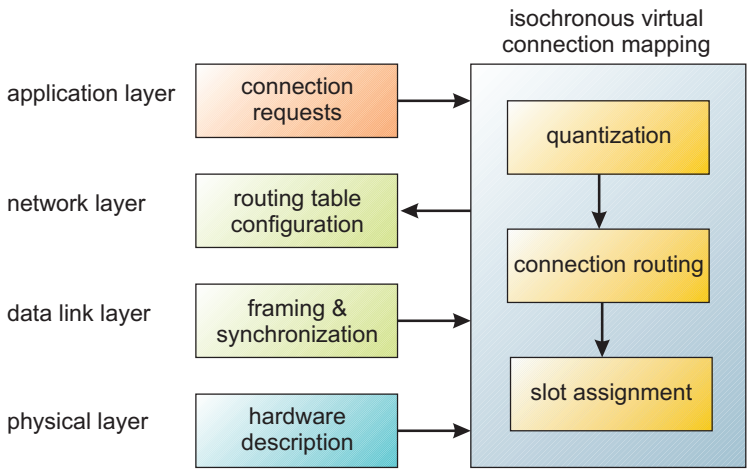



Figure 3.23: The connection mapping process consists of three steps executed offline during the network initialization phase. It computes the routing information needed at runtime. Information from different network layers are required.

reservation scheme ensures that data slots arriving at switch inputs can be forwarded immediately to the local outputs. Since no intermediate buffers are needed, this greatly reduces the transport time or *connection delay*. Furthermore, this reduces the *delay variation* of the connections for a better isochronicity. Section 3.5.6 also showed that the buffer-less forwarding results in a global coupling of the data slot timing of all network links, which leads to a complex *global* assignment problem to be solved. A weak mapping may leave many data slots unused or may fail to route some connections at all, whereas a good mapping results in low delay and jitter values. This section proposes an algorithm to solve the global mapping problem and to generate an appropriate reservation scheme.

### 3.6.1 Algorithm Overview

*offline computation*

Since the QoS requests are assumed to be known for all connections prior to network execution, the connection mapping process is executed during the network initialization phase.It computes the network routes of the connections, the reservation pattern of each link as well as the local routing tables of the switches. The online forwarding process is simplified to table look-ups of the pre-calculated routes. In fact, online routing and switching are merged to the same process of low complexity. Once assigned, data slot positions reserved for connections remain fixed during the operation of the network. Offline computation also allows to implement more complex algorithms in software independent of a software or hardware implementation of the runtime network part.

*input parameters*

Figure 3.23 illustrates the connection mapping process in the network layer view. Information from different layers are required to generate the online routing information:

1. A hardware description of the network topology. The graph $G = (V, E)$ represents the network with network nodes $v \in V$ and edges $e \in E$. For the mapping process, its is not necessary to distinguish between physical links or local links

to upper-layer processes. All links are bidirectional with the same bandwidth $w$ and fixed physical delay values (cf. section 3.2.1).

2. Information of the selected framing parameters, most important, the number $m$ of time slots per reservation period that equals the maximum number independent data slots that is available per link.
3. Data link layer synchronization information. The slot shift $s_e \in \mathbb{N}_0$ is assigned to each link $e \in E$ and denotes the shift of the data slots introduced by the switching process between the input port assignment and the output port assignment at the edges source node (cf. section 3.5.6). The number $f$ of time slots per frame that results out of the synchronization constraints the selection of $m$ according to equation 3.3.
4. A set $C$ of upper-layer connection requests $c \in C$. For each $c = (v_s, v_d, w_c)$, $v_s, v_d \in V$ denotes the source and destination nodes and $0 < w_c < w$ the bandwidth requirement. A change in the connection requests therefore enforces a new mapping.

*algorithmic steps*

The proposed mapping algorithm consists of three main stages executed one by one:

1. Quantization of the connection bandwidth requests to time slots.
2. Routing of the connections.
3. Calculation of a proper data slot assignment without connection conflicts at intermediate network links.

The result of the mapping process is a bandwidth assignment of all connections in terms of a slot reservation of all network links (positions of data slots belonging to physical links or local links to upper-layer interfaces). This information is used to configure the online routing tables of the switches.

*QoS results*

The QoS results for the connections directly depend on the mapping result, but are determined in different stages of the algorithm: The throughput is guaranteed simply by the reservation of an appropriate number of slots per connection according to the bandwidth demands. This is performed in step one of the algorithm. In contrast to the throughput, the transport delay and jitter bounds are a result of the connection mapping algorithm and cannot be requested in advance. The route selection in step two determines the absolute value of the connection delay. The isochronicity of the connections is determined by the jitter, which depends on the distribution of the reserved slots within the time frames, which are calculated in step three. For a detailed discussion of the QoS results, refer section 3.7.

*rejection of requests*

The connection mapping process can be compared to the *signaling phase* of a single connection in circuit-switching networks (cf. section 1.3 or also [143]). As a difference, the mapping process of MCGN handles the requests of all connections at a time. If the mapping process fails in finding a valid reservation pattern, it can simply be repeated with different framing parameters. The network user has to decide whether to leave connections out, to reduce the bandwidth requirements of connections or to modify the number $f$ of time slots per frame to allow for a larger number $m$ of time slots per reservation period. This would result in a finer

bandwidth granularity and simplify the mapping algorithm, but also may result in increased jitter results for connections already routed. The best QoS results are achieved for the smallest possible number $m$ that leads to a valid mapping.

### 3.6.2 Bandwidth Quantization

*global framing*

The bandwidth quantization is the first step of the mapping process. As discussed in section 3.2.3, the global framing strategy divides the time axis into consecutive time frames. Each reservation period consists of $m$ time slots of the same size. The bandwidth $w_s$ that is transported within a single time slot of a reservation period is denoted by equation 3.4.

*proposed algorithm*

During the quantization process, each connection is assigned a number of data slots according to its QoS throughput (i.e. bandwidth) request using the following algorithm:

**Algorithm 3.6.1** *Given the number $m$ of time slots per reservation period, the total bandwidth $w_f$ that is usable for data per link and the set $C$ of connection requests $c \in C, c = (v_s, v_d, w_c)$ with $v_s, v_d \in V$ and $w_c \leq w_f$ the amount of bandwidth requested by the connection.*

*For each connection request $c \in C$, calculate the number of data slots $m_c \in \mathbb{N}$ to be reserved as the smallest integer number that holds:*

$$\frac{w_c}{w_f} \leq \frac{m_c}{m} \leq 1 \tag{3.19}$$

Algorithm 3.6.1 rounds the requested bandwidth up to the next possible value according to the time slot granularity imposed by the global framing scheme. Note that no connection can request 100 % of the physical bandwidth $w$ since a small amount is wasted by the inter-frame gap. The algorithm does not calculate the exact positions of the reserved slots, but only the number of slots to be reserved within each reservation period exclusively for that connection on the whole route from the connections source node to its destination.

### 3.6.3 Connection Routing

*routing using shortest path*

The second step is the routing of the connections, i.e. the finding of valid paths between their end points through the network. The routing task can be solved with the following algorithm. It is based on the algorithm from Dijkstra [143] that calculates the shortest path between two network nodes. The Dijkstra algorithm itself requires a set of edge weights to calculate a pseudo-distance between two network nodes. It then finds the global route using the lowest total amount of weights. For that reason, the algorithm proposed below generates a set of edge weights that supports an even distribution of the routes among the global links. The edge weights are first initialized with the same number different from zero such that the Dijkstra algorithm returns the shortest path with reference to the number of intermediate hops.

**Algorithm 3.6.2** *Given the number $m$ of time slots per reservation period, the graph $G = (V, E)$ of the network topology, the set $C$ of connection requests and the numbers $m_c$ of reserved slots calculated by algorithm 3.6.1.*

1. *For each link $e \in E$, extend the description of $e$ by assigning a number $l_e$ denoting the load of the link such that $e = (v_s, v_d, l_e)$. Initialize all numbers $l_e$ with the value 1.*
2. *For each connection $c \in C$, do the following:*
   (a) *Calculate the sub-graph $G_c = (V, E_c)$ with the set of edges $E_c \subseteq E$ that contains all links $e \in E$ for which $l_e + m_c \leq m + 1$. The graph $E_c$ contains all links that have at least $m_c$ slots remaining.*
   (b) *Calculate the shortest path $r_c = (e_1, e_2, ..., e_n)$, $e_i \in E_c$, $e_1 = (v_s, v_i, l_i)$, $e_n = (v_j, v_d, l_j)$, between source node $v_s$ and destination node $v_d$ on the graph $G_c$ using the algorithm from Dijkstra. Use $l_e$ as the edge weights.*
   (c) *If the algorithm from Dijkstra fails, the request for connection c is rejected. Otherwise, for all edges $e_i \in r_c$, increase the corresponding edge weight $l_e$ by $m_c$*

*avoid overbooking*

During the processing of the algorithm, edge weights increase and paths for connections to be routed depend on the routing of previous connections. Succeeding connections are routed around heavily loaded edges and use free edges first. Step 2a ensures that only edges are considered for the routing process that offer sufficient remaining bandwidth to avoid an overbooking of links. Doing so, the algorithm ensures that each reservation period is booked with not more than $m$ time slots. Note that the same principle can be used to bound the number of slots reserved per link. This is useful to ensure a sufficient amount of unreserved slots to be occupied by packet-based traffic in the case of highly loaded connections.

*result*

The result of algorithm 3.6.2 is the set $R$ of all routes $r_c$ computed for each connection $c \in C$. The resulting end-to-end delay of a successfully routed particular connection is basically the sum of the transmission times $D_e$ of its links along the computed route (cf. section 3.7). The ability to request certain delay values by the user is not supported since the delay directly depends on the calculated route. If a particular connection cannot be routed, the connection request is rejected.

*optimizations*

Algorithm 3.6.2 proposes a basic algorithm. Conceivable optimizations can exploit symmetries, physical delays or pre-known optimal routings of certain connections. A crucial point of the algorithm is the calculation of the edge weights $l_e$, for which the proposed solution uses the current load, as well as the selection of the initial distance value, e.g. by using a value dependent of the physical delay. Other optimizations are to modify the processing order of the requests to start with delay-intensive long-range connections or to route direct neighbored connections first.

### 3.6.4 Slot Assignment

*global assignment problem*

The last step of the connection mapping algorithm assigns the dedicated slot positions within the mapping periods at each network link. It has been discussed in section 3.4.2 that the forwarding scheme of MCGN imposes a slot assignment scheme with the following constraints:

- Each data slot on a link can be used by a single connection only.
- No internal buffers are used. Data slots at input ports of switches are forwarded to their dedicated output ports with constant delay.

The avoidance of buffers couples the timing of data slots arriving at the input ports to the timing at the output ports. The assignment of data slots to connections is therefore a global problem.

*vertex color problem*

The assignment problem can be transformed to the *vertex color* [13] problem from graph theory, which is introduced first:

**Definition 3.6.1** *Vertex color problem. Given a graph $G = (V, E)$. Each node $v \in V$ is to be assigned a number (or color) $n_v \in \mathbb{N}, 0 \leq n_v < k$ such that no two adjacent vertices (that share the same edge) are assigned the same number.*

The vertex color problem is a common graph problem and many assignment or scheduling problems can be reduced to the problem of coloring a graph (e.g. scheduling problems like time tables as well as the famous puzzle *Sudoku*). The problem of finding the least number of colors $k$ required for a proper coloring according to definition 3.6.1 is found to be *NP*-hard [70]. However, there exist many algorithms to solve or approximate this problem [13, 88, 146]. To see the equivalence of the slot assignment to the vertex color problem, the following observations are stated:

*single slot*

- Without loss of generality, it can be assumed that each connection occupies a single time slot at each link of its route, i.e. $m_c = 1$. Connections that have been assigned multiple slots in algorithm 3.6.1 can be divided into $m_c$ independent connections that use the same route in parallel each with a single slot. At the end of the slot assignment algorithm, these connections can be re-merged back to the original one.

*constant delay*

- Since the forwarding at each switch is done with constant transmission time and all switches are synchronized to the same time period, the positions of the data slots at intermediate links along the route of a connection are determined just by the position of the data slot at the *first* link at the source node of the connection.

*inter-connection conflicts*

- The constraints of assigning data slots to a certain connection result out of the previous assignment of slots to other connections along the network route.

*assignment reduced to start slot*

According to the network topology used for discussion, all connections start at upper-layer processes, i.e. the first link of a route is the local link from the upper-layer process to the switch port at the source node (cf. section 3.2.1) The data slot at the first link of a connection $c$ is named its *start slot* $q_c \in \mathbb{N}, 0 \leq q_c < m$ in the following. The task of the slot assignment algorithm is therefore to assign an appropriate start slot $q_c$ for each connection $c$.

The algorithm that solves the assignment problem depends on the framing strategy at the output ports of the switch that is determined during the synchronization (cf. section 3.5.6). If the synchronization uses fixed framing, the slot position of the connection is the same number $q_c$ on all links of its route. If shifted framing is used, the slot positions are shifted by $s_e$ on each link $e$ along its route. The problem is first discussed in the more simple case of fixed framing.

### Slot Assignment with Fixed Framing

*assign different start slots*

The usage of fixed framing keeps the slot positions on the route of a connection. Since all slots of a connection are determined by its start slot, it can be stated that a connection contents with all other connections for *the same* start slot that have at least one link in common along their routes. The algorithmic problem therefore is to assign different start slots to connections that share a link. It is easy to see that this equals the vertex coloring problem with a graph that contains the connections as its vertices and has edges between connections that share a physical link.

As an example, figure 3.24 shows a network of four hosts. Each host contains a logical network node for its switching process within the data link layer as well as for its upper-layer process. The logical network topology used for discussion therefore consists of eight network nodes for global switches and local upper-layer processes. Figure 3.25 shows the list of the connection requests to the network as well as the corresponding collision graph to be colored.

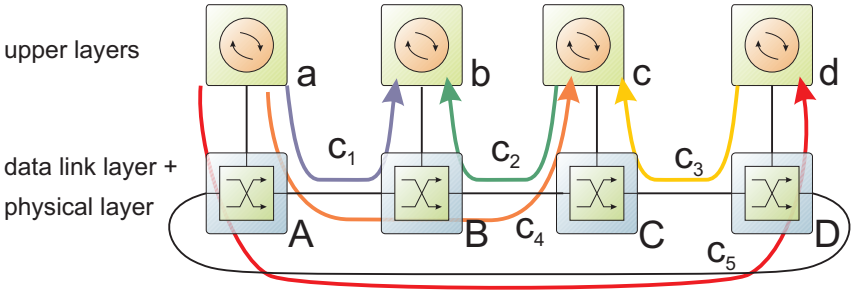



Figure 3.24: Example network consisting of four network hosts named $A$ to $D$ with the corresponding local upper-layer processes named $a$ to $d$. Five connection requests $c_1$ to $c_5$ are routed between the nodes. Physical links and local interfaces both have to be shared by the connections.

| Connection | Route | | | |
|---|---|---|---|---|
| $c_1$ | (a,A) | (A,B) | (B,b) | |
| $c_2$ | (c,C) | (C,B) | (B,b) | |
| $c_3$ | (d,D) | (D,C) | (C,c) | |
| $c_4$ | (a,A) | (A,B) | (B,C) | (C,c) |
| $c_5$ | (a,A) | (A,D) | (D,d) | |

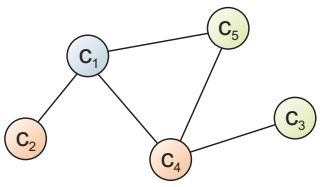



Figure 3.25: (left) The connection set of figure 3.24. Routes are represented as a set of successive network links. (right) The connection collision graph of the example network. Connections sharing a link are connected with graph edges. A vertex coloring of the graph assigns different colors to adjacent vertices and thus different time slots of network bandwidth to conflicting connections. Three different colors are required.

After these observations, the following algorithm can be defined to solve the assignment problem if synchronization is established with fixed framing:

**Algorithm 3.6.3** *Given the graph $G = (V, E)$ of the network topology, the number $m$ of time slots per reservation period, the set $C$ of connection requests $c \in C, c = (v_s, v_d), v_s, v_d \in V$ and the set $R$ of connection routes $r_c \in R$, $r_c = (e_1, e_2, ...), e_i \in E$.*

1. *Create a connection collision graph $G_{cc} = (V_{cc}, E_{cc})$ with vertices $V_{cc}, |V_{cc}| = |C|$ and edges $E_{cc}$ by using the following rules:*
   (a) *Create the graph vertices $v_c \in V_{cc}$ by assigning a vertex $v_c$ for each connection $c \in C$.*
   (b) *Create a graph edge $e \in E_{cc}, e = (v_1, v_2)$ between the vertices $v_1, v_2 \in V_{cc}$ of connections $c_1$ and $c_2$, if and only if it exists an edge $e_l \in r_1 \cap r_2$ in the routes of both connections $c_1$ and $c_2$.*
2. *Carry out a graph coloring algorithm for the graph $G_{cc}$. The graph coloring assigns each $v_c \in V_{cc}$ a color $q_c \in \mathbb{N}, 0 \leq q_c < k$. The algorithm tries to minimize $k$.*
3. *If the computed number $k$ holds $k \leq m$, the algorithm succeeds and the calculated numbers $q_c$ are the start slots of the connections. Otherwise, the algorithm fails without generating a proper assignment.*

*algorithm result*

The results of the proposed algorithm are the start slots $q_c$ for the connections. The algorithm fails in the case that the number of time slots needed for a proper coloring exceeds the number of available time slots $m$ per reservation period, especially for small values $m$. In this case, the slot assignment process can be repeated with modified connection requests with fewer bandwidth requests and thus a smaller number of required time slots per connection. Furthermore, the framing and synchronization can be changed to a larger frame size $T = mnS + G$ and thus a finer bandwidth division.

### Slot Assignment with Shifted Framing

*shifted slot positions*

Section 3.5.6 showed that using shifted framing at the switch outputs during the synchronization causes the frame slot positions to shift on successive links such that data placed into the start slot $q_c$ at the source node appears at different slot positions at the links along the route. The shift $s_e$ at each edge $e$ depends on the network topology, on the transmission time $D$ and on the selected frame size $T$.

*avoid certain start slot differences*

The constraint that two different connections cannot use the same time slot on a certain link results in a constraint for a forbidden *difference* in the start slots of both connections according to the sum of the slot shifts of both connections up to that link. This condition is illustrated for an example network of six hosts in figure 3.26.

*collision condition*

The further calculation of the appropriate collision condition requires the slot shifts $s_e$ for each link $e = (v_s, v_d, s_e), e \in E$ that result out of the synchronization process (cf. section 3.5.7). The *accumulated slot shift* $a_{c,e}$ of a connection $c$ from its start node up to the colliding link $e$ can then be computed as

$$a_{c,e} = \sum_{e' \in r_{c,e}} s_{e'} \tag{3.20}$$

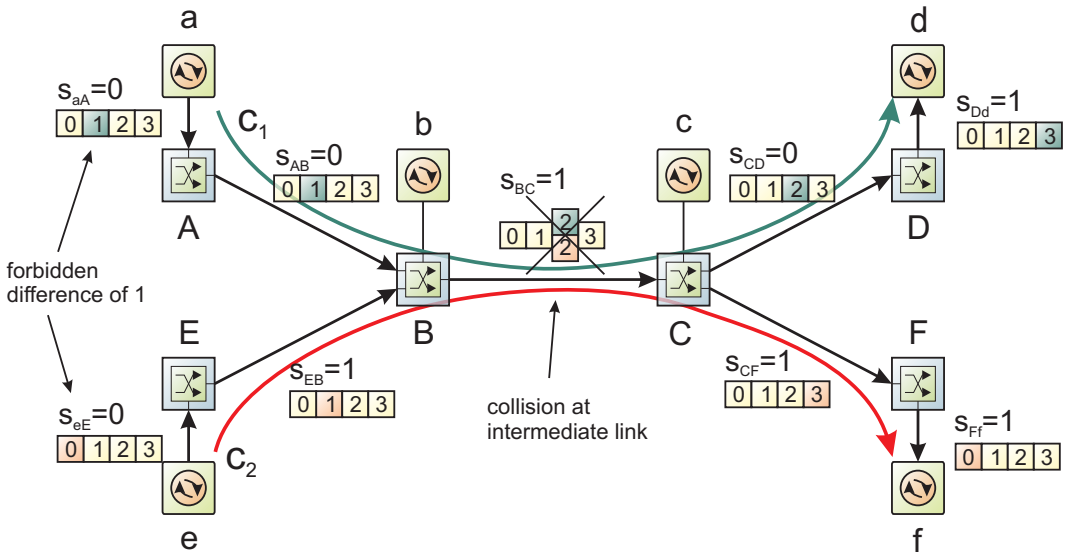



Figure 3.26: Example of colliding connections $c_1$ and $c_2$ sharing an intermediate link. Synchronization uses shifted framing that causes slot positions to shift differently along the routes. The example collision requires to avoid a start slot difference of one during the slot assignment process.

along the intermediate links $e'$ of the partial route $r_{c,e}$ from the source node up to $e$. The appropriate slot assignment algorithm resembles algorithm 3.6.3, but uses a modified version of the vertex color algorithm:

**Algorithm 3.6.4** *Given the graph $G = (V, E)$ of the network topology, the number $m$ of time slots per reservation period, the set $C$ of connection requests $c \in C, c = (v_s, v_d), v_s, v_d \in V$ and the set $R$ of connection routes $r_c \in R$, $r_c = (e_1, e_2, ...)$, with the edges $e_i \in E$ extended by the relative local slot shifts $s_e$ as described above. The algorithm works similar as algorithm 3.6.3 with two modifications:*

1. *(b) Create a directed graph edge $e \in E_{cc}, e = (v_1, v_2, d)$ between the vertices $v_1, v_2 \in V_{cc}$ of connections $c_1$ and $c_2$, if and only if it exists an edge $e_l \in r_1 \cap r_2$ in the routes of both connections $c_1$ and $c_2$. Compute the edge weight $d \in \mathbb{N}_0$ as the difference of the accumulated slot shifts $d = (a_{c_1,e_l} - a_{c_2,e_l})$ mod $m$ of both connections.*

2. *Carry out a modified graph coloring algorithm for the graph $G_{cc}$. The graph coloring assigns each $v_c \in V_{cc}$ a color $q_c \in \mathbb{N}, 0 \leq q_c < k$ such that the colors of any two adjacent vertices $c_1$ and $c_2$ do not differ by the value of its directed edge modulo $m$.*

Algorithm 3.6.4 computes the start time slots $q_c$ of all connections such that no collisions at intermediate links occur. Figure 3.27 shows the corresponding connection collision graph of the network of figure 3.26. Since only two connections have to be routed, only two vertices exist in the graph connected with a single directed edge.

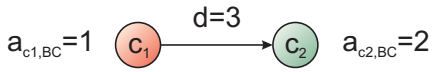



Figure 3.27: The connection collision graph that corresponds to figure 3.26. The network uses $m = 4$ slots per reservation period. The forbidden distance between the start slots $q_1$ and $q_2$ of both connections calculates to $d = (1 - 2) \mod m = 3$.

### 3.6.5 Overall Algorithm Result

*find best mapping*

The two algorithms 3.6.3 and 3.6.4 result either in the calculation of the appropriate start slots $q_c$ for all connections or fail to map the connection requests. Since the QoS jitter results depend on the mapping process, it can be repeated for different reservation periods ($m$, $n$) or even a different synchronization (parameters $f$, $T$, $S$) for the same set of connection requests. The best jitter values are achieved for the smallest possible value of $m$ that results in a successful mapping (cf. section 3.7).

*generation of routing tables*

After the optimal mapping has been found, the calculated start slots are taken to generate the routing tables of the switches:

- In the case that the frame is divided into multiple reservation periods ($n > 1$), the calculated slot assignment of the start slots $0 \leq q_c < m$ is used for all remaining periods with the slot numbers $m \leq i < f$. The connection $c$ may therefore insert its data at the slot times $\{q_c, q_c+m, q_c+2m \ldots\}$ at the interface at its source node.
- The positions of data slots reserved at intermediate links are computed by taking the accumulated slot shifts into account: The reserved slots of connection $c$ at a link $e$ equal $\{q_c + a_{c,e}, q_c + a_{c,e} + m, q_c + a_{c,e} + 2m, \ldots\}$.
- Connections that have been assigned multiple slots in algorithm 3.6.1 have been split into $m_c$ single-slot connections for the slot assignment. After its execution, the connection features the appropriate number of different start slots $0 \leq q_{c1}, q_{c2}, \ldots < m$ and also the corresponding slots in further reservation periods as well as on all links of its route.
- The local routing tables are calculated for each output and each frame time by denoting the input ports that belong to the connections of the appropriate reserved slots.

The routing tables calculated in the last step are used to configure the switches. After that, the network initialization phase is completed and the network is ready to operate.

### 3.6.6 Further Remarks

*optimizations*

It has been shown that the proposed algorithms 3.6.1 to 3.6.4 are suitable to solve the connection mapping problem. Since the steps two and three of the algorithm solve problems of network-wide complexity, it strongly depends on the connection requests of the user whether a successful mapping can be found. Again, since the number of requests or the requested bandwidth are not limited in advance, a perfect algorithm solving arbitrary requests cannot exist. Conceivable improvements of the presented algorithms are:

- The grouping of connections in the case of multiple connections share the same source and destination endpoints. This requires additional effort by the upper-layer processes to multiplex and demultiplex the individual data streams, but may lead to a more efficient usage of bandwidth. *grouping*
- The sorting of connections by bandwidth requirements or by hop distance to solve the algorithmic problems first for the more simple sub-cases and thus to reduce the complexity for the remaining problems. *sorting*
- The introduction of randomness to select the best out of multiple tries. *randomness*
- An iterative solution alternating through steps two and three to re-route problematic connections for a minimization of vertex color conflicts. *iterating*

The algorithms can further be optimized for specific applications according to the network topologies, symmetries or pre-known connection requests. As an example, the reference implementation presented in the succeeding chapter has been optimized for a large number of connections for the research with VLSI artificial neural networks within a regular topology. *reference implementation*

## 3.7 QoS Results for Isochronous Connections

This section summarizes the QoS guarantees made by the MCGN architecture for the transfer of data within isochronous connections. As a general remark, it can be stated that the exclusive reservation of time slots to connections avoids traffic interactions in the global path. The discussion of QoS guarantees can therefore be reduced to the traffic characteristics of a single isolated connection only. The following calculations are given for a framing according to section 3.2.3: The time frame and the frame gap have durations $T$ and $G$. The frame consists of $f$ data slots, divided into $n$ equal reservation periods of $m$ slots each. The bandwidth of $m_c$ time slots is reserved for the connection and the connections route $r_c$ consists of $h_c$ physical links to the destination.

### 3.7.1 Throughput and Drop Rate

The throughput is guaranteed by the reservation of $m_c$ time slots per reservation period exclusively to the connection $c$. This results in the reserved bandwidth of $w_c = w_s \cdot m_c$ (cf. equation 3.4). The slot admission policy at the transmit interface at the source node of a connection ensures that this bandwidth is not exceeded as discussed in section 3.4.6. *reserved bandwidth*

Since connection-oriented traffic is transported as priority traffic, the reserved bandwidth equals the guaranteed throughput. For the same reason, no data slots at intermediate switches are dropped. Therefore, the total drop rate is zero after data has been sent at the source interface by not exceeding the reserved bandwidth. *no drops*

### 3.7.2 Reliability

The reservation guarantees that no data slots will be lost due to traffic interactions. However, external errors in the physical layers of the network (e.g. due to electrical *error forwarding*

distortions) can indeed lead to slot losses. Physical layers that implement sophisticated encoding mechanisms (e.g. the usage of symbols or 8b/10b-encoding) are able to detect such errors directly [150]. In most cases, the data link layer also stores checksums like CRC around the frame data to ensure the data integrity. This is modeled by the inter-frame-gap, which is not available for user data.

*error forwarding*

Within MCGN, the forwarding of this error information to the upper network layers at the destination node is hindered by the following reasons:

- The switches forward incoming data slots without buffering to the output ports.
- The switching process merges data slots from different frames of different input ports to new frames at the output ports.
- Error information via checksums are available not earlier as the incoming frame finishes.
- A detected checksum error marks the whole frame as erroneous.
- The error information cannot be forwarded until all data slots of the received frame have already been forwarded

To handle frame errors, an error property has to be stored at the frame end after the valid data (i.e. within the frame gap). Two alternative strategies are conceivable:

- The storage of an error-property for the whole frame.
- The storage of an error-property for each individual time slot.

*implementation*

The first alternative would require to mark *all* frames sent to output ports as faulty, as long as at least a single data slot of the erroneous frame is sent to that output. This would be repeated by each downside switch at any output port of the switch that initially detects the checksum error. Consequently, this would cause to distribute the failure property of the frames over the network and would therefore affect multiples connections, although its data has been transmitted correctly. The second alternative would set all bits in case of an error and forward the bit according to the slot assignments. This storage of the bits would cause a reduction in the available bandwidth and would increase the administrative overhead. Furthermore, the calculation of the forwarded error-bits would be a time-critical task.

*unreliable connections*

For that reason, transmitted data within isochronous connections is not specified to be reliable for the sake of optimized connection delay and jitter. It is proposed to implement additional error detection and error handling algorithms within upper-layer processes. Note that this is no strong limitation, since most existing upper-layer protocols implement further error detection algorithms for its own. Error detection information from the physical layer or from the data link layer can still be evaluated to monitor the reliability of the physical links.

### 3.7.3 Delay

*deterministic delay*

The connection delay equals the overall end-to-end transmission time of a data slot. As discussed in detail in section 3.5, the synchronization avoids the buffering within the switches of the network. As a result of this, the forwarding of data slots through

the network is performed with a constant transmission time between the switches, which results in a deterministic overall connection delay.

*constant intermediate delays*

Connection data to be sent is placed into available time slots according to the admission policy at the transmit interface described in section 3.4.6. The data path of the interface is connected directly to an input port of the local switch (cf. figure 3.6). Starting at the first switch, the global route has previously been calculated during the mapping process and is fixed for each connection. Section 3.5.4 showed that the delay between the input ports of two adjacent switches connected via the edge $e$ is given by the transmission time $D_e$. The value of $D_e$ can be split into separate parts for the time needed for switching, synchronization and the physical layer transmission:

$$D_e = d_{CB} + d_{SY} + d_{PH}. \tag{3.21}$$

The value of $D_e$ is constant only for each individual data slot of a frame, but may differ between the slot positions by the duration of the frame gap, depending on the framing strategy at the output ports (cf. section 3.5.6). The end-to-end delay $D_c$ of a connection $c$ equals a summation of the individual transmission times $D_e$ of the links $e$ along its route $r_c$:

$$D_c = (\sum_{e \in r_c} D_e) + d_{CB} = const \tag{3.22}$$

The last term corresponds to the time required to traverse the central crossbar at the last switch to the local receive interface. According to the particular values of $D_e$, $D_c$ is also constant for each individual time slot.

*regular topology*

The distribution of the transmission times $D_e$ depends on the network topology. If the network has a regular topology with similar transmission times, all synchronization delay elements can be adjusted such that $D_e$ equals the same value $D$ on all links $e$ with $D = \max D_e$. In this case, equation 3.22 can be simplified to

$$D_c = h_c \cdot D + d_{CB} \tag{3.23}$$

with $h_c = \|r_c\|$ as the number of network hops (in terms of physical links) between the source node and the destination node.

*delay bounds*

Concerning the absolute value of the delay, the switching component $d_{CB}$ can supposed to be small in the range of a clock cycle, since this includes mainly the traversal of the crossbar. The synchronization delay depends on the framing strategy at the output ports of the switch. In the general case, synchronization uses the shifted framing method to best adopt to the network topology. For this case, section 3.5.6 showed that the delay $d_{SY}$ is bounded by the duration of a time slot $S$. The proportions between the duration of a time slot $S$ and the delay of the physical layer $d_{PH}$ depend on the network structure as well as on the selected framing parameters. To reach a fine bandwidth division, it can be assumed that most networks use a comparably small slot size $S$ such that $S \ll d_{PH}$.

*conclusion*

This leads to the conclusion that the total overall delay value is mainly determined by the physical layer of the network for encoding, decoding and physical transmission. In other words, the forwarding technique of MCGN, which avoids buffering by reservation and synchronization, reduces the internal processing time of the switches to the very minimum such that the resulting overall connection delay mainly consists of the time for the physical transmission between the network hops.

### 3.7.4 Jitter

*jitter sources*

The above calculation of the connection delay considers the transmission time between the transmit interface at the source node of the connection and the receive interface at the destination. Data that has been placed into a specific time slot indeed arrives after a fixed number of global slot cycles. However, this holds true only for each individual time slot of the whole time frame.

Variations in the end-to-end connection delay, or *connection jitter*, are caused by the following sources:

1. The jitter of the local clock with respect to the global clock reference.
2. The time to wait for an appropriate time slot at the transmit interface.
3. The framing structure that requires an inter-frame gap every $f$ data slots.
4. The shifted framing technique required for synchronization.

Clearly, a connection jitter as small as possible is generally desired as it corresponds to good isochronicity. If a very small jitter is required by the application, the jitter can be reduced further by using a jitter buffer at the destination node, but this also increases the total transmission delay. The effects (2-4) rely on the distribution of the time slots, which is calculated by the connection mapping during the network initialization phase. Since all effects are deterministic, the jitter for each connection can be calculated exactly prior to the network operation. Due to the exclusive reservation of data slots, this can be done separately for each connection.

*definition*

To calculate the jitter, a mathematical definition is required. Unfortunately, the term jitter is defined differently by the community. Some publications avoid the term completely and stay to the expression *delay variation* [31]. This thesis uses the common definition of jitter $J_c$ as the difference of *worst-case* delay and *best-case* delay:

$$J_c = \max(D_c) - \min(D_c) \tag{3.24}$$

*calcus*

The jitter sources mentioned above affect different parts of the transmission process. The clock jitter (1) is independent of the other three sources (2-4), which depend on each other since all are correlated with the selected frame structure. Therefore, the jitter is divided into the two independent parts:

$$J_c = J_{clock} + J_{con} = J_{clock} + \max(D_{con}) - \min(D_{con}) \tag{3.25}$$

The first part denotes the clock variation whereas the latter part denotes the connection jitter between the upper-layer user processes measured in multiples of clock cycles.

*connection jitter*

To calculate the connection jitter, the transmission process of connection data is split into the process of waiting for an appropriate time slot and the process of transmitting the slot data to the destination.

$$D_{con} = D_{wait} + D_{trans} \tag{3.26}$$

In a next step, the connection delay is calculated for the transport of data within the time slot $i$ by considering the above jitter sources (2-4). The frame structure

requires to consider *all* time slots $0 \leq i < f$ of a frame and cannot be reduced to a single reservation period.

*accuracy*

An exact calculation of the wait delay requires to consider each clock cycle within a time frame at which data may become valid at the local interface. This is necessary to include the inter-frame gap in the calculation. Depending on the framing parameters, the size of the frame gap may be ignored for $G \ll S$, but since MCGN supports very tiny slot sizes $S$, the general discussion has to include it.

The following paragraphs describe each jitter source and give upper boundaries for the several effects. The description closes with the discussion of an example calculation.

### Clock Jitter

*clock distribution*

The clock jitter is caused by the physical distribution of the reference clock to each network node, which is required for the synchronization as described in section 3.5.2. The time period of the reference clock is denoted by $\gamma$. For a proper operation, the jitter of the clock can be assumed to be:

$$J_{clock} \ll \gamma. \tag{3.27}$$

Since the size of a time slot is at least a clock cycle and the network architecture can assumed to be a digital system, the effect of the clock jitter is ignored in the following ($J_{clock} = 0$).

### Interface Wait Time

*wait time*

Interface delay is caused at the source interface, at the time an upper-layer process waits until a free time slot becomes available for its connection (cf. section 3.4.6). The value of the jitter depends on the distribution of reserved data slots for the connection within the reservation period of the frame. An even slot distribution allows a continuous injection of data and thus results in low connection jitter, whereas a rare or cumulated placement of the slots forces data to be held causing unwanted delays. The distribution within the reservation period depends on the mapping result. The number of slots reserved depends on the bandwidth requirement for a dedicated connection. The frame gap has to be considered for the case that data becomes valid at the interface at the time the last reserved slot just passed by.

*worst case example*

Figure 3.28 shows an example reservation pattern with a single reservation period per frame and a single slot reserved for the connection. The minimum wait occurs, when the data becomes valid at the time of the reserved time slot (indicated by an arrow in the figure):

$$\min(D_{wait}) = 0 \tag{3.28}$$

The maximum number of cycles to wait occurs if the data becomes valid a cycle later:

$$\max(D_{wait}) = T - \gamma \tag{3.29}$$

This configuration also results in the maximum jitter possible. The jitter value does not depend on the slot assignment algorithm, but only on the selected frame size $T$.

*multi-slot connection*

If multiple slots are reserved for the connection, this reduces the *worst-case delay*

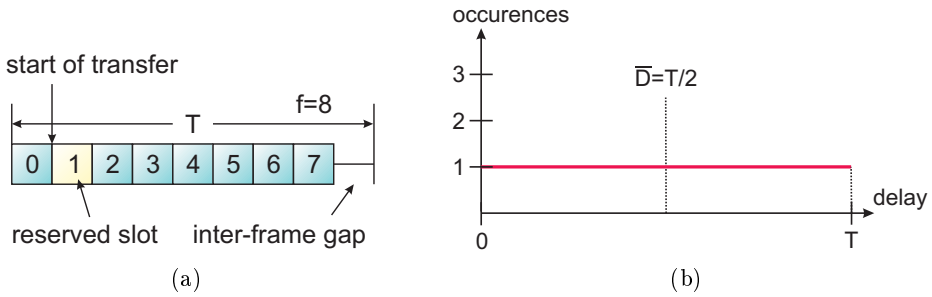



Figure 3.28: Calculation of connection jitter caused by the waiting process for an appropriate time slot. (a) Example reservation pattern for a single-slot occupancy. (b) Distribution of the wait times depending on the relative cycle, the data becomes valid at the transmission interface.

by the duration $S$ of a time slot for each additional reserved slot. The worst-case wait-time occurs in case the slots are reserved at successive slot positions and data becomes valid just after the last reserved slot passed by:

$$\max(D_{wait}) = T - (m_c - 1) \cdot S - \gamma \tag{3.30}$$

Clearly, the individual maximum wait-time depends on the particular distribution and may be smaller.

### Effect of Reservation Periods

*framing effect*

The reservation scheme allows to split the data frame of $f$ time slots into $n$ equal reservation periods with $m$ time slots each. Each reservation period gets the same reservation pattern calculated by the slot assignment algorithm of the connection mapping process. The usage of multiple reservation periods supports large frame sizes to waste only a small amount of bandwidth for the inter-frame gap, but also achieves a small connection jitter.

Since each reservation period of a particular link gets the same reservation pattern, at least $n$ data slots are reserved at each link along the route of each connection. The interface wait time is therefore bounded by the duration of a single reservation period $(T - G)/n = mS$ extended by the gap time $G$:

$$\max(D_{wait}) = m \cdot S + G - \gamma \tag{3.31}$$

The usage of $n$ reservation periods also results in the fact that the frame gap has to be considerered only in every $n$-th reservation period during the calculation of the wait time at the transmit interface. As an example, consider figure 3.29. The mapping process reserved a single slot within each reservation period. Although the slots are distributed evenly among the frame, the inter-frame gap causes the distance between the slots to vary between 4 and 5 time slots.

### Effect of Synchronization

*fixed framing*

Section 3.5.6 showed that the synchronization can be performed using two different

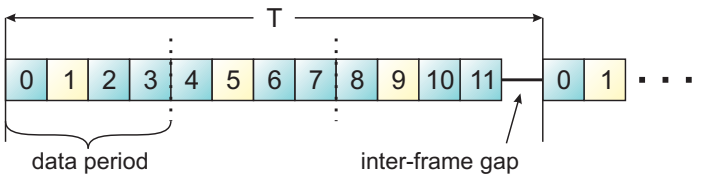



Figure 3.29: Multiple reservation periods cause different wait times even on periodic reservation pattterns

framing techniques. With fixed framing, $d_{SY}$ is constant in time and each data slot has the same transmission time, which can be calculated according to equation 3.22:

$$\min(D_{trans}) = \max(D_{trans}) = D_c \tag{3.32}$$

*fixed framing*

Shifted framing can be used to synchronize a network of arbitrary topology with arbitrary frame sizes $T$ by introducing a logical slot shift $s_e$. The drawback of this technique is that this results in two different transmission delays. The slot shift causes the frame to be split into two parts at the position $f - s_e$. The transmission times of the slot numbers $i$ with $0 \leq i < f - s_e$ and of the slot numbers $f - s_e \leq i < f$ differ by the duration of the frame gap $G$:

$$\min(D_{trans}) = D_{c,<} \tag{3.33}$$
$$\max(D_{trans}) = D_{c,>} = D_{c,<} + G \tag{3.34}$$

*multi-hop connections*

If shifted framing is used on multiple links along the route of a connection, the slot position changes multiple times and the particular transmission times can alternate between $D_<$ and $D_>$. In case of a regular topology that requires the same shift on all links, the slot numbers indeed shift on the whole route such that both transmission times occur, but the constant shift causes the differences in the total delay to be the same as on single-hop connections: The frame is split into two parts, the latter part experiences a delay that is increased by the duration of the frame gap.[4] In contrast to that, network topologies that require multiple different slot shifts result in a difference of the transmission times of the data slots of up to a gap for each network hop:

$$\max(D_{trans}) \leq D_{c,<} + h \cdot G \tag{3.35}$$

A transmission using a dedicated (start) slot still has a constant transmission delay, but only the transmission delays of different slot positions differ. In case the framing is implemented with a single reservation period ($m = 1$) and only a single slot is reserved for a connection ($m_c = 1$), only a single slot is available per frame, such that the transmission time is constant ($D_{trans} = \text{const}$) and synchronization has *no* effect on the connection jitter.

*example setup*

**Example Calculation** Consider figure 3.30(a). The framing uses a single reservation period with $f = m = 8$ data slots. The data slot $S$ as well as the frame gap $G$ are of the same size of $S = G = 2$ cycles. The total time frame $T$ is of the size

[4] The proof is left out here for readability.

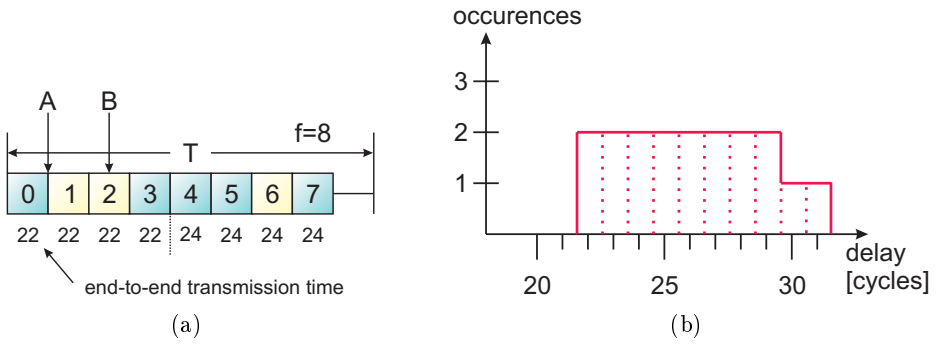



Figure 3.30: Jitter calculation example for a multi-slot reservation pattern (a) The slot reservation pattern. Transmission delays for each slot are noted below the frame. Each slot as well as the frame gap comprises two clock cycles. (b) Distribution of the total connection delays including the wait time and the transmission delay.

$T = 18$ clock cycles. The connection for which the jitter is to be calculated occupies three slots ($m_c = 3$). The transmission times for each slot position are written below the slots in multiples of clock cycles. The connection has a single hop. The synchronization has been made using a slot shift of $s_e = 4$, which causes the slots of the back part of the frame to be transmitted with an increased delay of the duration of the gap.

*calculation*

To calculate the connection jitter, it is required to compute the total connection delay for data becoming valid at each cycle $t = (0, .., 17)$ within the time frame $T$. Data can be sent to the network only at the cycles (0, S, 2S, ... = 0, $2\gamma$, $4\gamma$, ...). As an example, data that becomes valid at the first two cycles undergoes a total connection delay of 22 or 23 cycles, respectively. Figure 3.30(b) shows the resulting delay distribution. The minimum delay of 22 cycles results for data that becomes valid within the cycles 1 and 3 (position 1 has been marked with an $A$ in the figure). The maximum delay of 31 cycles (7 wait + 24 transmission) results for data that becomes valid at cycle 4 (marked with a $B$). The total jitter calculates to:

$$J_c = \max(D_c) - \min(D_c) = 31\gamma - 22\gamma = 9\gamma = 4.5 \cdot S \tag{3.36}$$

Note that the calculated connection jitter is independent of the number of network hops. It can further be reduced by a more even distribution of the reserved slots within the frame.

### Overall Jitter Results

By using equations 3.24, 3.25 and 3.26 and by considering the four jitter sources discussed above, the connection jitter is bounded by the following equations:

*single-slot connections*

The maximum jitter computes for a single-slot connection ($m_c = 1$) transported within a single reservation period ($n = 1$):

$$J_c = T - \gamma \tag{3.37}$$

The jitter of a single-slot connection that is transported using a framing with multiple reservation periods ($n > 1$) equals:

$$J_c \leq m \cdot S + (1 + h) \cdot G - \gamma \tag{3.38}$$

The term $h \cdot G$ of the above equation depends on the synchronization, i.e. the network topology. The equation can be formulated stronger in the case that shifted framing is used with the same shift on each link: The longest wait-time (via the frame gap $G$) then corresponds to the slot in the first period. This is either in the back part of the frame (resulting in $D_{c,>}$ for all slots of $c$) or corresponds to the smaller transmission time $D_{c,<}$ (which cancels the additional gap wait). In this case, $D_{trans}$=const. The usage of fixed framing or shifted framing with the same shift at each link therefore results in:

$$J_c = m \cdot S + G - \gamma \tag{3.39}$$

for *all* single-slot connections. Note in particular that the jitter is independent of the number of network hops!

*general case*

For the most general case of a multi-slot connection ($m_c > 1$) transported within multiple reservation periods ($n > 1$), the jitter is bounded by:

$$\begin{aligned} J_c &\leq m \cdot S + G - (m_c - 1) \cdot S + h \cdot G - \gamma \\ &= (m - m_c - 1) \cdot S + (h + 1) \cdot G - \gamma \end{aligned} \tag{3.40}$$

Again, the $h$-term depends on the global synchronization and may be zero. The first term results out of the distribution of the $m_c$ slots within the reservation period. Since the gap can assumed to be small compared to the whole frame, it can be seen that even for this general case, the dependence on the number of network hops is only very small.

### 3.7.5 Summary

It has been shown that the connection jitter is determined completely by the global synchronization and the connection mapping process. Its values can be calculated for each connection prior to the network operation. The above equations are worst-case values. The worst-case values imagine the worst-case mapping result, i.e. a slot reservation pattern with large slot distances. The actual values will be smaller in most cases.

*recommended settings*

To conclude the discussion, it can be stated that the connection jitter (i.e. the isochronicity of the connections) for a given set of connection requests is determined by the following factors:

- The physical layer of the network (the required synchronization settings).
- The selected frame structure (parameters $f$, $S$, $m$).
- The slot assignment result (the performance of the mapping process)

*physical properties*

The physical layer of the network cannot be assumed to change. It determines the size of the frame gap $G$, which is required for synchronization and error detection. Furthermore, it determines the network topology and thus the global synchronization settings in terms of the slot shifts $s_e$ on each physical link. Regular network topologies can be configured by using the same slot shift on all links. This reduces the influence of the frame gap to the connection jitter according to equation 3.39.

*framing*

The framing parameter should be selected to large values $f$ to support multiple reservation periods for an even distribution of data slots. This ensures good jitter

results even on difficult connection requests that require a large value for the size of the reservation period $m$.

*slot size*

Most important, the network should be configured for the use of small data slots $S$. Small data slots allow to reserve a larger number for each connection, which can more evenly be distributed among the reservation period. MCGN explicitly supports the usage of tiny slot sizes due to its protocol-invariant and low-complex forwarding process. The data slot size can be tiny down to a single clock cycle.

## 3.8 Service for Packet-Based Transports

*unreserved or unused slots*

This section describes the transport of packet-based traffic by the MCGN architecture. Packet-based traffic is transported as best-effort traffic, i.e. it uses the remaining bandwidth not used by priority traffic. The usage of unused priority slots for packet data allows the reservation of links completely for priority traffic while still having the opportunity to transmit packet data over this link depending on the actual bandwidth requirements of connection-oriented traffic.

*traffic matrix not known in advance*

In contrast to priority-traffic, no assumptions about the traffic matrix are made prior to the network operation. The user does not have to provide information about the amount and the rate of packets to be sent between the network nodes. Consequently, no bandwidth resources are reserved prior to network operation. The network initialization phase does not concern packet-based transfers. Packet routing is performed at runtime. MCGN uses buffers (or queues) within the bypass-switch to handle bursts and collisions in the packet flow.

### QoS for Packet-Based Transports

*implementation-specific*

The architecture of MCGN defines how isochronous connections and packets can be merged to use both traffic types in a hybrid network. The bypass-switch presented in this section is the central component that achieves this task. The design of the switch is rather a universal architecture concerning the placement and arrangement of buffers and schedulers than a detailed specification.

Because of that, no strict QoS guarantees can be made in general concerning the transport of packet-based traffic. However, the packet-based part of the bypass-switch equals the design of an input-queued crossbar switch. It is possible to choose a particular packet scheduler or buffering technique that offers QoS for this type of switch. Concerning the packet-based nature of the traffic, the QoS services provided will be more likely statistical than guaranteed.

### 3.8.1 Packet Embedding

*slot occupancy*

Each data slot is used either by connection data or packet data. Unreserved slots can be directly equipped with packet-based data. in contrast to that, the usage of data slots that are reserved but not used by priority traffic requires an additional single bit of each reserved slot for this purpose, which slightly reduces the available rate of the connection. The bit denotes whether the slot is occupied by connection-data or not.

The slot organization is illustrated in figure 3.31. The start of the packet is marked by a start-of-packet (SOP) character. The packet data is then placed consecutively within all available slots. Note that packet data is not merged between different packets as it is the case for connection-based traffic. Note further, that packets can be extended across a large number of slots and even across several time frames. This depends on the packet size, the selected framing parameters for the transport of connection-based traffic as well as on the actual amount of bandwidth required by the connections.

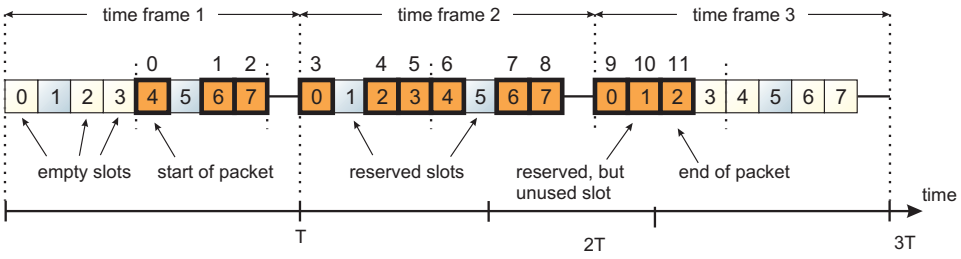



Figure 3.31: Embedding a best-effort packet into the slotted framing scheme. Slots one and five are reserved for priority connections. The start-slot and end-slot of the packet are marked by special characters. The relative slot numbers of the packet are written above the framing scheme. The packet occupies time slots not used by priority traffic to improve the overall bandwidth efficiency.

*physical layer requirement*

The purpose of the SOP character is to explicitly mark the packet start within the unreserved slotted data stream. The usage of a dedicated framing character ensures a robust detection of a packet and the transmission of packets of different sizes within the same network. The character does not have to occupy the full data slot, but must be unambiguously detectable. Since the network architecture requires the physical layer to provide special characters distinguishable from data for synchronization purposes (cf. section 3.5.6), the same technique is feasible for this case.

*interaction between classes*

The hierarchic priority of both classes results in an influence from the actual bandwidth used by connections to the bandwidth available for packet-based transfers. Clearly, this depends on the amount of data slots reserved per frame compared to the amount of unreserved slots. In principle, it is possible to reserve all data slots of a time frame solely for priority traffic, such that best-effort data can only be transported within unused slots. However, this hinders the transport of packets and starves out the whole traffic class completely in the case of heavily loaded connections. The proposed reservation process for connections-based data therefore can be configured to keep a certain amount of slots free for packet-based data (cf. section 3.6.3).

### 3.8.2 Packet Format

A best-effort packet of MCGN consists of a routing header, the user packet as payload data as well as control characters required for the online transportation process. Since the size of a time frame as well as the size of a data slot are selected by the user for an optimal configuration of connection-based traffic, it is not defined how much

fraction of a time slot a data field occupies. The SOP character and the header may be completely located in the first data slot or distributed across a couple of slots.

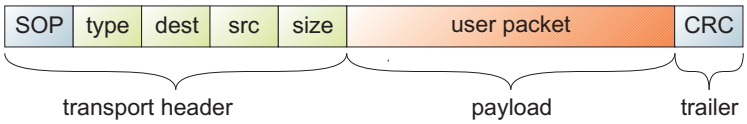



Figure 3.32: Data format of a best-effort packet. The user packet is transported as payload.

*packet header*

Figure 3.32 shows the resulting format of a best-effort packet. The SOP character is part of the transport header. The header further contains the type of service, the destination address and the source address of the packet, as well as the payload size stored in separate fields. This information is required to route the packet to the correct network node and to deliver it to the correct protocol interface (see below). The protocol type and destination address are placed first to support a fast routing decision at the arrival of the packet at an intermediate switch.

*user data*

The header is followed by the payload data, i.e. the user-packet to be transported. Its data content depends on the protocols implemented within the upper network layers. The transportation process of MCGN is independent on the packet payload. It is neither defined nor investigated during network operation, which is part of the multi-protocol support of MCGN. Existing packet-based protocols like IP or ATM can be transported via MCGN without further adaptation. As the only constraint, the user has to define its maximum size. This is required to facilitate the packet buffering process within the switches as well as the detection of erroneous data. The packet trailer consists of the CRC checksum, which ensures the data integrity and is calculated over the header and the payload.

#### Implementation

*application-specific*

The specification of a MCGN packet is rather qualitative than quantitative. MCGN defines the data fields and its meanings, but not the exact size and bit-representations (as e.g. the endianess) of each field. This is since the required size of the data fields depends on the actual application as well as on the physical layer of the network. In small NoCs, it could be sufficient to use only a few bits for the whole transport header to avoid bandwidth waste and to reduce the packet latency. In contrast to this, in larger networks, every address field could required to be a few bytes in size to serve a large number of hosts or to ensure the uniqueness of the addresses.

*example*

As an example, the common IEEE 802.3 Ethernet [92] protocol is a data link layer protocol for LANs and WANs. A common variant uses 14 bytes for a comparable header. The reference implementation of MCGN described in the succeeding chapter uses only 4 bytes for the complete packet header.

### 3.8.3 The Bypass-Switch

The bypass-switch is the core component of the MCGN architecture. Most of the properties of MCGN result directly out of the properties of the switch. The switch performs the integration of the two traffic classes on the physical links. Its main features are :

*main features*

- A hierarchical design. The switch can be split into separate parts responsible for the transfer of connections and for packets, respectively. The connection-based part is executed with priority and equals the isochronous switch discussed in section 3.4.1. The packet-based part equals an input-queued crossbar switch with VOQs.
- The avoidance of internal buffers for connection-based traffic. This is the key aspect to ensure good QoS delay and jitter results as described in section 3.4.
- Separate crossbar inputs for packet-based traffic from the queuing stages. This is a significant improvement in the performance of best-effort data at the presence of priority data without reducing the service guarantees for the latter (cf. section 5.7).
- A generic and modular design concerning the switching of packets. The implementation of the queuing stages and the central scheduler can be optimized for the application.
- A scalable design in terms of port numbers and line speed. The switch does not require internal speedup. The complexity is bounded by the implemented scheduler and the duration of a time slot.
- A compact design feasible for the implementation in programmable logic.

The bypass-switch provides separate interfaces for connection-based traffic and for packet-based traffic to the local upper network layers for a feasible usage.

### Switch Overview

*general remarks*

The design of the switch is based on an input-queued crossbar switch with VOQs and a centralized scheduler, but with additional queuing-bypasses for priority data. An $N$-port switch has $N$ bidirectional ports of the same line rate. Ports can be either global ports connected to other switches via the physical layer or local ports connected to upper-layer processes via local interfaces. Local ports implement only either of the two traffic classes. The switch operates according to the framing scheme of external data discussed in section 3.2.3 that divides the time and thus the bandwidth into frames and slots. It is illustrated in figure 3.33.

*demultiplexing of traffic classes*

Data slots arriving at a global switch input port are first separated according to its traffic class. The membership to a class is defined by the reservation pattern stored within the priority scheduler as well as by the single bit of reserved slots denoting the actual occupancy. The hierarchical design of the switch allows to discuss the switching process for both data types separately.

*priority traffic*

Data slots belonging to priority connections are handled the same way as within the isochronous switch described in section 3.4.1. The slots are forwarded immediately to the crossbar and are switched to its corresponding output port without any buffering. The reservation pattern and the local time counter are contained in the priority scheduler in figure 3.33.

*best-effort packets*

Best-effort packets are stored within the input queues and are pre-sorted to its destination output ports (VOQ). The switch implements conventional store-and-forward packet switching [71], i.e. each packet is stored completely before the crossbar is requested. This is required due to the fact that the arrival time of the last

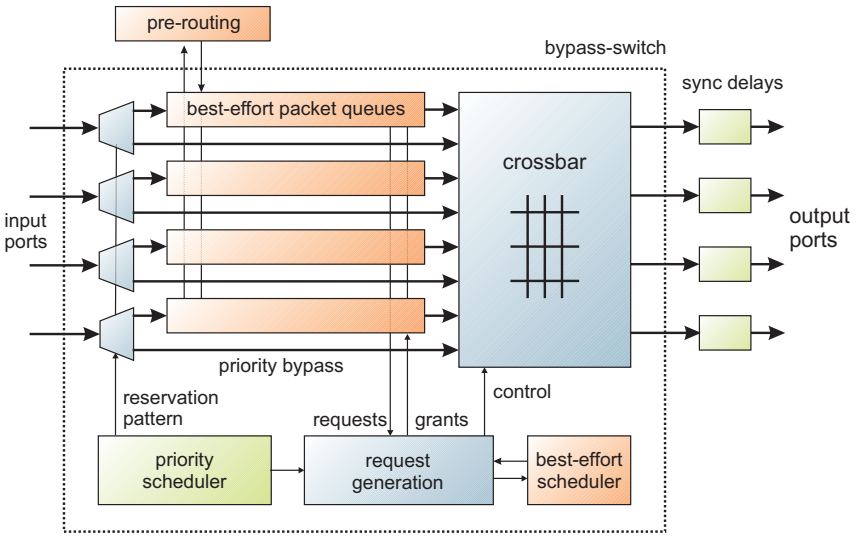



Figure 3.33: Schematic of a four-port bypass switch with only global ports for both classes. The hierarchic design provides buffer-less forwarding of priority data by also serving best-effort packet transports comparably to an input-queued switch with VOQs. The doubling of the crossbar inputs avoids contention between both traffic classes at the inputs completely.

data slot cannot be predicted as it depends on the current priority traffic. The early forwarding of data slots (wormhole switching, cut-through switching) could block the corresponding links for an unreasonably long time in the case the last slot is delayed. The request generation logic evaluates the $N^2$ requests from the queuing stages and extracts valid request that do not interfere with pre-reserved port-matchings from the priority scheduler. The central best-effort scheduler selects the remaining queues and the crossbar is configured accordingly.

### Virtual Output Queuing

Best-effort packets are stored in VOQs that is, the switch uses different queues for each output at each input. Although each queue stores full best-effort packets only, the writing and reading of the queues is performed according to the slotted timing of the switch. Figure 3.34 shows a schematic of a single VOQ located at a switch input.

*queue input*

Data slots belonging to best-effort packets are demultiplexed out of the packet stream and re-assembled into packets. At the time a packet arrives, the destination address of the header is investigated to calculate the local output port the packet has to be forwarded to. This requires the routing decision to be made at the time the packet arrives at the switch. The packet is then stored in the appropriate queue. The CRC checksum is verified to ensure the data integrity of the queue content.

*queue output*

Each VOQ generates $N$ *request* signals to denote the existence of packets to be forwarded to the outputs. The central scheduler returns *select* signals to denote the queue to be served next as well as the appearance of free data slots at the actual output. The selected packet is then read from its queue slot by slot and multiplexed into the slotted data stream according to the external reservation pattern. Packets

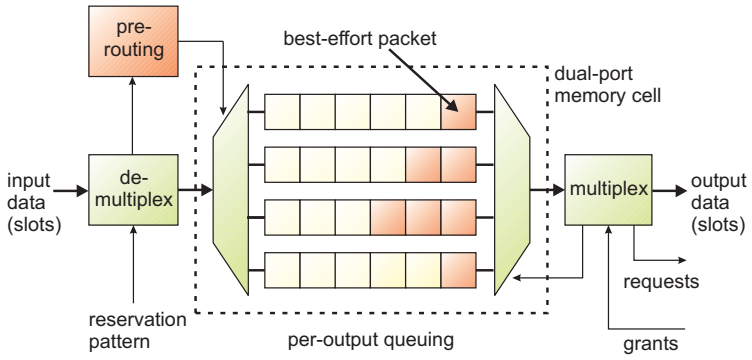



Figure 3.34: A virtual output queue

are not merged, i.e. the multiplexer serves no packet until the previous packet has been transmitted completely.

The timing scheme implies that only one queue is written and read at the same time. Therefore, all $N^2$ queues required for an $N$-port switch can be implemented with $N$ dual-port memory elements and $N^2$ FIFO controllers.

### Best-Effort Packet Scheduler

*slot-based scheduling*

The timing of reserved traffic is based on the duration of a time slot, which causes the scheduling process and the crossbar arbitration to operate accordingly. The operation of the best-effort scheduler is controlled by the request generation logic: In each time slot, the scheduler gets up to $N \times N$ *request* signals from all currently unmatched VOQs for which inputs and outputs are not used by priority traffic in the *following* time slot. The scheduler then calculates a bipartite matching of inputs and outputs within the duration of a time slot. At the time the scheduler completes, the output multiplexer of each VOQ is informed of the scheduling result by asserting the *select* signals. To ensure that the data slots of different packets are not merged to the same output port, the scheduling result for a dedicated output is hold until a packet has been transmitted completely.

*scheduler type*

Section 1.5 discussed the guarantee of QoS for packet-based traffic, which is difficult to achieve in input-queued switches due to the HOL blocking problem and the preliminary unknown traffic pattern. The HOL problematic has been solved by implementing VOQs. The modular and hierarchic design of the bypass-switch allows to select a couple of existing algorithms for the online scheduling. The implemented algorithm can in principle be any scheduler that is usable with VOQs, i.e. that calculates a bipartite matching out of $N \times N$ requests. This concerns all crossbar schedulers presented in section 1.6, e.g. iSlip [84, 86], RPA [59] or DPA [59] etc.

*generic design*

Since the scheduler for best-effort packets is interfaced to the request generation logic, the respect of pre-reserved slots is transparent to the scheduler. As an advantage, the implemented scheduler can be any packet scheduler that handles $N^2$ requests and asserts $N$ select signals as used in input-buffered switches. No special adaption of existing scheduling algorithms of that type to the bypass-switch is required. This allows to select the algorithm best adopted to the application in terms of complexity and QoS results.

### Central Crossbar

*multiplexing of data slots*

Data slots belonging to connection-based traffic as well as packet-based traffic is fed to the crossbar via separate inputs. The crossbar has $2N \times N$ input ports and multiplexes the data slots independent of its traffic class. There is no internal speedup. Each time slot, the crossbar transfers at most one data slot from each input port to the output ports. Comparably, each output ports receives data from at most one input port. The crossbar is controlled by the the request generation logic by denoting the actual input port to each output port according to the reservation pattern and the scheduling result of the best-effort scheduler. No further control logic is required.

## 3.8.4 Interface to Upper Layers

*local ports*

It has been discussed in section 3.4 that the framing scheme and the synchronization of a MCGN network requires all nodes to cope to the special switching technology of the network. This requires to implement at least parts of the bypass-switch even at network nodes used only as end-nodes for best-effort packets, depending on the traffic classes used by the upper-layer processes. Clearly, local ports for connection-based traffic and best-effort packets do implement its single traffic class only. Furthermore, each port requires flow additional control signals that are not shown in figure 3.33.

### Local Interface for Connection-Based Traffic

*corresponds to isochronous switch interface*

The isochronous switch has already been described in section 3.4.1 as the connection-based part of the bypass-switch. Consequently, the interface to connection-based traffic of the bypass-switch equals the interface described in section 3.4.6. The interface is again illustrated in figure 3.35 with respect to the bypass switch. The figure shows the transmit and the receive parts together with the relevant parts of the switch. The local port is implemented with a direct connection to the crossbar. The data flow is controlled by the priority scheduler, which corresponds to the forwarding algorithm of figure 3.6. The control signals equal the signals described in section 3.4.6.

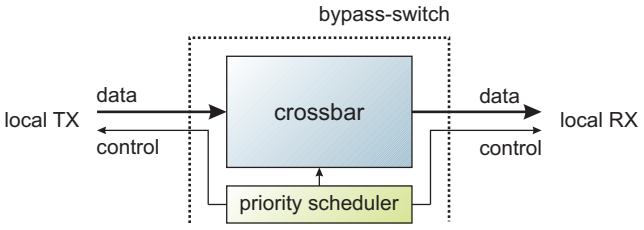



Figure 3.35: The local port interface for connection-based traffic equals the upper-layer interface of the isochronous switch. The interface consists of a direct connection to the crossbar. The data flow is controlled by the priority scheduler that contains the routing tables.

### Packet Adaptation Layer

*fragmentation to slots*

Figure 3.36 shows the packet interface of the bypass-switch to upper-layer processes. Since the timing of the switch is based on time slots, the transmission requires the packets to be split into single slots as well as the reception requires the reassembly of the received slots back to packets. This tasks are performed by an additional layer located between the switch and the upper-layer processes. The layer is denoted as the *packet adaptation layer (PAL)* in the following.

*multi-protocol support*

The PAL supports multiple different upper-layer protocols. Each packet protocol is assigned a unique type-identifier for this purpose. Data exchange between the several layers is controlled by *valid* and *accept* signals. The multiplexing of the protocols requires a separate set of the control signals for each interface. It is also possible to support only a general packet protocol like IP [114] and carry further transport protocols within to reduce the complexity of the interface.

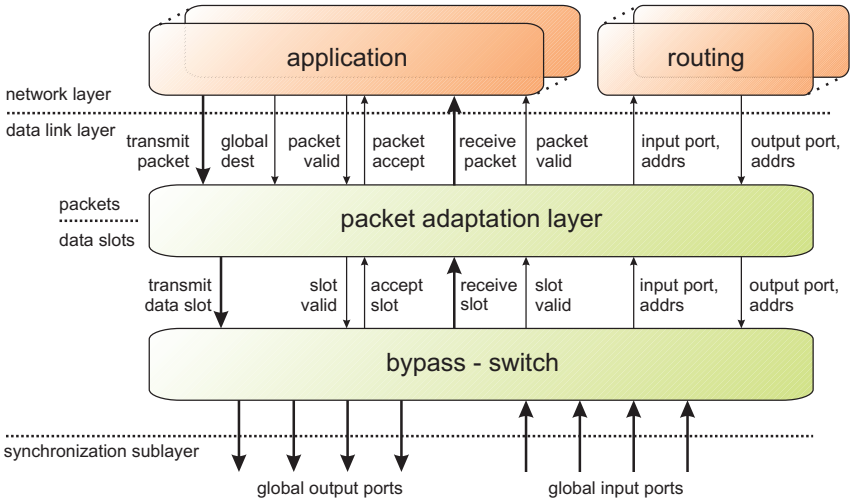



Figure 3.36: The upper-layer interface for the transmission of best-effort packets. The PAL converts user packets into data slots and vice-versa. The routing process calculates the output port number for the packets. Multiple protocols are supported.

### Packet Transmission

In contrast to the transmission of connection-based data, the transmission of packets is not limited by an admission policy, but by the actual load of the network. Furthermore, the maximum packet size has to be obeyed. The adaptation layer performs the following tasks to transmit a packet:

1. The conversion of the global destination address and source address from the user format to the MCGN format of the network stored in the packet header. This is required since the network layer may use its own addressing scheme, which is different from the scheme used in the actual MCGN network segment.[5]

[5]This can be compared to the conversion of IP [114] addresses to Ethernet MAC addresses [92] by the ARP [112] protocol

2. The compilation of the best-effort packet by adding the required header and trailer to the user data. The type of the packet depends on the upper-layer protocol transmitting the packet.
3. The split of the best-effort packets into single data slots
4. The transmission of the data slots to the switch according to its slotted timing.

Since the switch first demultiplexes the data slots back to packets for the storage in the VOQs, the last two tasks can be left out if the implementation allows to merge the switch and the adaptation layer in a single process.

#### Packet Reception

The reception of best-effort packets is performed similar:

1. The data slots are received by the PAL according to the actual network traffic.
2. The PAL reassembles the slots back to the best-effort packet.
3. The packet type stored in the packet header is investigated.
4. The packet is forwarding to the protocol that corresponds to the type.

*routing process*

As an additional requirement, each network layer protocol must provide a routing process for its own address format. The routing process receives global destination addresses in the format of the protocol and calculates the route to the destination. It then returns the corresponding number of the output port. The routing process is also required to calculate the local output port for best-effort packets arriving at the switch to place the packet into the correct VOQ.

### 3.8.5 Packet Routing

*universal routing support*

Routing is commonly an issue of the network layer. Although MCGN is mainly a switching architecture, it provides a general and flexible routing support to facilitate the usage of multiple different network protocols. The advantage of the MCGN architecture lies in the facts that first, the packets do not necessarily have to be forwarded to the local upper layers for a routing decision and second, that different routing strategies can be implemented for individual protocols within the same MCGN network segment.

*formal description*

The forwarding decision is required for each individual packet that enters the switch to calculate the VOQ that corresponds to the correct output port (cf. again, figure 3.34 and figure 3.36). The bypass-switch therefore investigates the header of incoming packets and reads the protocol type $\phi$ as well as the destination address $DA$ and the source address $SA$. The forwarding decision to be made for a packet entering via an input port $I$ can be formally defined as a routing function

$$\Omega_{\Phi}(DA) = O \tag{3.41}$$

that calculates the output port number. A more general definition is given by

$$\Omega_{\Phi}(I, SA, DA) = (O, SA', DA') \tag{3.42}$$

that additionally calculates the new addresses $SA'$ and $DA'$ (which will be kept to $SA' = SA$ and $DA' = DA$ for most protocols). The output port can either be a local port (to the receive interface of the corresponding protocol) or a global port.

*multi-protocol support*

To support multiple protocols, the function $\Omega$ can be implemented differently for each protocol type $\Phi$. Since each network protocol itself controls the content of the packet header, the implemented routing protocols are completely transparent to the operation of the switch. This allows to operate conventional routing protocols like open shortest path first (OSPF) [96] for IP [114] packets as well as label-switched protocols like MPLS [118] or connection-based packet forwarding like used in ATM [143] networks in parallel within *the same* physical MCGN segment. To give some examples, the following routing strategies are feasible to be used within MCGN:

1. *protocol-independent packet switch*

   The switch operates protocol-independent. No knowledge about upper-layer protocols is required. Packets are switched only between global ports. The address fields are of the same format for all protocols. The output port is assigned according to the previous arrivals of packets at the input ports. For that reason, the switch administrates a list of known source addresses at each port. The packets are forwarded according to its destination address either to the corresponding port at which that address previously appeared or to all ports in the case the address has never been seen. This behavior resembles the operation of an Ethernet switch.

2. *conventional router*

   All incoming packets are forwarded to the upper-layer processes according to its protocol type. Routing decisions are made in the following. Packets are then re-transmitted to the calculated destination. As an advantage, the complexity of the routing algorithm is not bounded by the switch timing but however, the packet delay is increased. In this case, the switch operates like a conventional router.

3. *early routing*

   MCGN supports a fast variant that performs the routing decisions early at the time the packet enters the switch. The packet data is not investigated, but only the destination address of the packet is evaluated. The packet is stored into the VOQ of the calculated global output port and does not have to be forwarded to upper layers for the routing decision, which drastically reduces the forwarding time. This requires a fast routing decision, since the complexity of the routing algorithm is bounded by the packet arrival rate.

4. *label-switching router*

   The MCGN switch can also be used to implement label-switched routing for certain protocols as it is used in e.g. MPLS [118] networks. The addresses equal local labels that are valid only on the local links between the switches. The routing algorithm changes the two addresses (labels) $SA$ and $DA$ according to the label-switched path of the network. A detailed discussion of this technique is beyond the scope of this thesis.

The routing functionality finally selected depends on the application but is indeed limited by the complexity of the routing decision. Note that sophisticated routing algorithms commonly require the knowledge of the network topology and thus introduce further administrative overhead. As a final remark, it is stated that the second

variant allows to transfer the packets to other networks using a different switching technique than MCGN. Such an edge router implements both technologies and is part of both network segments. This makes it possible to interconnect MCGN networks to other existing transport technologies.

## 3.9 Scalability and Complexity

This section summarizes the space and time complexity of the MCGN architecture with respect to both traffic classes.

### 3.9.1 Space Complexity

Concerning the space complexity, the MCGN architecture has been developed for a compact implementation within programmable logic. The synchronization sublayer mainly consists of the variable delay elements and the framing logic at each output port. Its logic consumption is minimal. The local interfaces for the connection-based traffic consists only of few handshaking logic as well as the packet interface, which mainly converts the data slots to packets and vice versa.

*complexity of the bypass-switch*

The space complexity is therefore determined by the central bypass-switch. Since the VOQs can be implemented within a single memory element, the central crossbar determines its space complexity. By doubling the amount of input ports, the switch removes conflicts between priority and best-effort traffic at the input ports with moderate additional costs. Although the size of the crossbar is doubled to $2N \times N$, its space requirement is still of

$$O(N^2) \quad (\textit{space complexity}) \tag{3.43}$$

complexity. A compact implementation of the switch therefore has to ensure an efficient implementation of the central crossbar.

### 3.9.2 Time Complexity

*connection-based traffic*

Concerning the time complexity, it has to be distinguished between connection-based traffic and packet-based traffic. It has been described in detail in section 3.4 how MCGN establishes isochronous connections by moving the forwarding complexity from online to offline. The remaining online process is reduced to the look-up of pre-calculated routing tables. The online complexity is therefore of

$$O(1) \quad (\textit{online time complexity}) \tag{3.44}$$

complexity. Furthermore, the look-up can be done in advance for each slot such that incoming data slots are forwarded immediately. Since also the data of isochronous connections is transported header-less and its content does not have to be investigated, this supports very tiny data slot sizes down to single clock cycles in conjunction with high line rates. The time complexity of the online forwarding process is in particular independent of the number of switch ports $N$ as well as on the number of network hops.

Concerning packet-based transports, the modular design of MCGN requires to select an online scheduling algorithm as well as a routing algorithm. It is therefore not possible to present exact complexity results, but the following paragraphs point out the possible complexity issues that have to be taken into account for the selection process.

*packet scheduler*

The scheduling algorithm can be any algorithm that calculates a bipartite matching out of $N \times N$ requests. Its time complexity is therefore bounded by the number of switch ports $N$. A new scheduling is required each time slot, which directly bounds the time complexity of the scheduler by the duration of a time slot $S$. Other setups are conceivable that calculate a scheduling only each $n \in \mathbb{N}$ time slots, but is not further discussed here. Since the transport of connection-based traffic is best for small time slots, the size of the time slot can either be selected as the smallest duration usable with the selected best-effort scheduler - or to select the best-effort scheduler according to the time slot duration.

*packet routing*

The second time-critical task concerning packet-based transfers is the execution of the routing decision. Depending on the implemented routing model, an incoming packet may have to be delayed until the routing algorithm completes before to be stored into the VOQs. In the case of the fast routing model of MCGN, a routing algorithm with low complexity should be selected, e.g. that uses pre-calculated tables or a simple routing concept. In the case that all incoming packets are first routed to upper layers, the time complexity of the routing decision is bounded by the packet arrival rate. The selection of the routing algorithm further has to consider a possible dependence on the number of network hops.

## 3.10 Summary

*novel switching architecture*

This chapter described the novel MCGN switching architecture, which combines transport techniques from circuit switching and packet switching. The architecture provides an end-to-end transport of multi-protocol data within both, isochronous connections as well as packet-based transports. It has been shown how MCGN uses a framing and synchronization technique prior to the network operation to exploit the deterministic behavior by a low-complex forwarding algorithm at runtime. Connection-based traffic is transported within reserved time slots, whereas packets are transported within unreserved or unused slots and are scheduled purely at runtime. MCGN provides a network-wide synchronization service to upper-layer protocols. Initial algorithms for the synchronization and the reservation processes have been presented. The resulting QoS bounds in terms of delay and jitter are given according to the selected framing parameters. MCGN guarantees 100 % throughput for connectio-based traffic.

*novel switch type*

The merging of the traffic classes is performed by a novel switch-type, the bypass-switch. The switch features a hierarchic design with respect to the traffic classes. Its modular approach allows to select the routing algorithm and the packet scheduler best adopted to the application. Its upper-layer interface provides protocol-independent transport services separately for each traffic class. The multi-protocol property of MCGN allows to transport existing protocols like IP, ATM etc. via MCGN without further adaptation.

*main features*

The main features of MCGN are: (1) guaranteed QoS for connection-based traffic with very low delay and jitter due to the avoidance of buffering within the data path, (2) a low online complexity for isochronous data, (3) a hierarchic and modular design, (4) scalability in terms of port numbers, line speed and the number of network hops (depending on the selected routing and scheduling algorithm), (5) multi-protocol support and (6) a compact design to be implemented within programmable logic.

### 3.10.1 Future Work

The modular design allows to add a large amount of additional features to the architecture. The following extensions are conceivable:

*support of multicasts*

- The support of multicast connections and packets. Concerning connections, this does not require a change of the isochronous switch specification, since the routing tables have simply to be configured to denote the same input for multiple outputs at a certain time. The connection mapping algorithms for the bandwidth reservation and the generation of the appropriate connection-collision graph have to be modified. The vertex-coloring of the graph can be preserved (cf. section 3.6). For packet-based transfers, multicast requires the duplication of incoming packets during the writing or the reading of the VOQ buffers.

*enhanced frame division*

- The dividing of the time frame into reservation periods could be enhanced to a more sophisticated technique. A hierarchic division of the frame is possible with a different amount of bandwidth to be reserved per data slot according to the level of hierarchy. Furthermore, the reservation pattern itself can be time-multiplexed such that the assignment of a particular slot changes from frame to frame periodically. The isochronous switch would use multiple hierarchic routing tables that are periodically looked up. This would not only allow a fine-grained reservation of bandwidth, but also to provide different kinds of QoS jitter guarantees. Nevertheless, this also allows larger data slots to support more complex best-effort scheduling algorithms.

*slight buffering and re-ordering*

- The introduction of a small re-ordering buffer at each switch within the data path of the isochronous connections. Although this would increase the delay slightly, this also would reduce the complexity of the connection mapping significantly. The resulting forwarding technique would be a mixture of the buffer-less forwarding of MCGN and the stop-and-go queuing presented in [44] or comparably in [58, 130].

*improve connection handling*

- The grouping of connections with the same end-points to a single connection. This would require further administration of the different data flows within upper network layers, but would reduce the mapping complexity. This technique is used in the reference implementation of MCGN presented in the next chapter.

*online re-configuration*

- It is further possible to add or remove connections within a MCGN network by calculating a new slot assignment and by re-configuring the routing tables at runtime without denoting the non-affected upper-layer processes. Since this will also affect connections that persist, is requires a detachment of the upper layers by preventing the injection of data during the re-configuration process.

*application-specific implementation*

Since the presented specification of MCGN discussed the design concepts and arrangements of processing instances instead to define exact signals and bit-meanings, a final implementation requires the fix of the several parameters to application-specific values. The succeeding chapter presents the reference implementation of an MCGN network developed for the research of hardware neural networks. The implementation uses nearly all of the specified features and has been implemented in programmable logic.

# Chapter 4

# Implementation of the Transport Network

---

*In the previous chapter, the specification of the MCGN architecture has been introduced. This chapter describes the reference implementation of the MCGN architecture within programmable logic and software. The chapter shows how the design principles of MCGN in terms of framing, synchronization, mapping and packet transports are put into practice. The reference implementation of the bypass-switch is presented in detail. MCGN is used for the lower network layers of the transport network for neural network experiments within the hardware framework of chapter 2. The chapter also presents the implementation of the higher layers of the transport network, namely a demonstrator application for isochronous connections as well as a distributed shared memory subsystem. The implementation within commercial programmable logic demonstrates the compact and scalable design of the network architecture. The evaluation of the implemented network is presented in the subsequent chapter.*

---

*MCGN for hardware neural network experiments*

The transport network consists of two parts: its lower layers are a reference implementation of the MCGN switching technology, whereas its higher layers contain the application-specific functionalities for neural network experiments within the Stage 1 framework. As the most important point, the reference implementation of MCGN fulfills the QoS requirements of the two ANN chips `HAGEN` and `Spikey` for the transport of the neural data as discussed in section 2.4. To be more precise, the transport requirements of the neural network experiments are satisfied by the two different traffic classes of MCGN:

- Isochronous connections are used for the transport of neural network data of the ANN chips. The synchronization and the reservation of time slots guarantee the throughput as well as bounded delay and jitter.

- Packet-based transports are used for an on-demand transport of the memory content of the different SDRAM chips. Packets are placed in unreserved or unused slots.

*design focus*

The implementation and operation of the transport network requires to fix certain configurable parameters as e.g. the slot size $S$ or the selected type of best-effort scheduler. Since the focus of the framework is the transport of neural data, the priority traffic class has been given advantage in case of interferences between both classes. An example is the size of the time slot $S$, which is selected to be two clock cycles, which limits the complexity of the best-effort scheduler to this duration. The digital design to be executed within the FPGA has been further optimized for a low consumption of programmable logic due to the limited size of the `xc2vp7` FPGA used. Appendix A lists resource consumptions for main design entities implemented with different parameters.

*configurability*

Due to the modular design of the network architecture, the reference implementation can be used for different applications with only minor modifications. This does not only include applications within the Stage 1 framework, but also other network environments that require multiple traffic classes to be served. As an example, the network can be used within the follow-up Stage 2 of the research project FACETS [89, 121] for wafers-scale integration of neural networks.

## 4.1 Overview

A block-level schematic of the transport network is shown in figure 4.1. It consists of two different parts:

*programmable logic*

- A digital design to be synthesized within programmable logic. The design consists of multiple modules, which are organized in a hierarchical way. The two main modules are `phys_sync`, which provides the physical transmissions via delay-reduced synchronized links, and the module `switch`, which contains the bypass-switch. Both modules can be instantiated and parameterized similar to intellectual property (IP) cores. The digital design has been developed using the hardware description language VHDL [32].

*software*

- Software to be executed on the control PC. The programs `mgtsync`, `mapping` and `mgtroute` perform the three steps of the network initialization phase of MCGN, namely the synchronization of the network, the calculation of the slot assignment and the configuration of the routing tables. The executable `switchtest` is a software simulator for a switch and generates testbench traffic pattern to verify the implemented design. The software has been developed using the C/C++ programming language [139]. The python [116] script `generate_network.py` creates test neural network topologies for evaluation.

*operating environment*

The digital design is embedded within the top-level design of the FPGA of the `Nathan` network modules. Only this part is executed during the operation of the network. The software is executed during the network initialization phase. The hardware/software interface is accomplished by the SlowControl of the framework (cf. section 2.2.4). The top-level design of the FPGA contains additional parts like

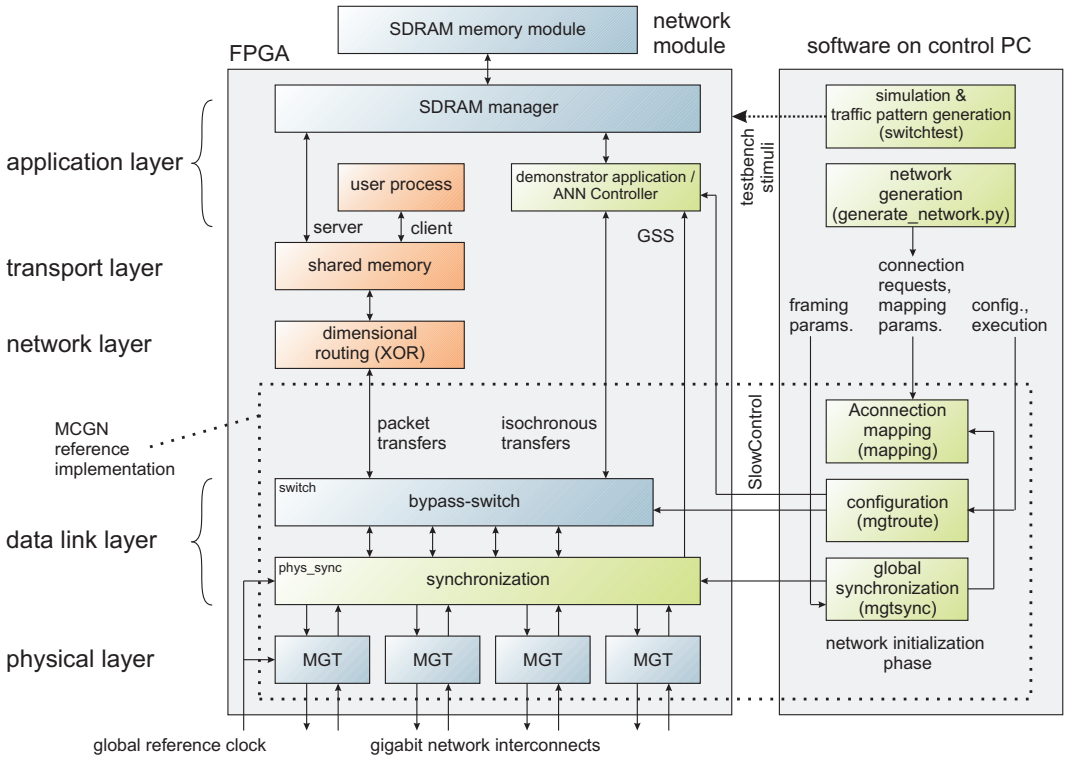



Figure 4.1: Block-level overview of the transport network. The implementation comprises a digital design within the programmable logic as well as three software programs, which are executed on the control PC. The reference implementation of MCGN is denoted by a dotted line. The software/hardware interface is accomplished by the SlowControl of the framework.

the ANN controller, the synchronous dynamic RAM (SDRAM) memory manager or the global clocking, which are not further described here, the reader may refer to [125, 47].

### Upper Network Layers of the Transport Network

*upper-layer functionalities*

Since MCGN is merely a switching technology, its specification comprises only the lower network layers. The transport network for hardware neural network experiments requires further functionalities within upper network layers. For this reason, two upper-layer processes have been implemented:

- The transport of neural network data within isochronous connections is demonstrated by an application that transmits its data within single time slots. The probability to transmit data within a slot can be tuned for different data rates to model the Bernoulli statistics of events from spiking neurons.
- The transmission of packets is demonstrated by a distributed shared memory (DSM) system, which comprises the SDRAM chips of all `Nathan` network modules. The access to memory on a distinct network module is provided by read and write operations to global DSM addresses.

*limitations*

This thesis is part of a larger scientific project. At its present stage, the developed

transport network is not finally connected to the controller of either of the two ANN chips [47, 125]. This is due to the fact that such a connection requires major modifications of the present implementation of the controllers. The development of the ANN chips and also the controllers is not part of this thesis. This work will be done in near future. The mentioned demonstrator application for isochronous transfers can be used as a starting point. In contrast to that, the shared memory subsystem is operating successfully.

*organization of the chapter*

The following sections describe each part of the implementation of the transport network. First, the framing of bandwidth to time frames and the embedding of best-effort packets is described. The succeeding sections describe the implementation of the digital part of MCGN, namely the physical layer, the synchronization sublayer, the bypass-switch as well as the corresponding software part to be executed during the network initialization phase. The last two sections concern the transmission of connection-based data and packets over MCGN, which is required for the neural network application.

*level of detail*

The level of detail is limited to the basic ideas and solutions. Due to the complexity of the implementation, a listing of the source code of either hardware or software is not presented. The description refers to the source code stored within the SVN [140] repository `fpgasystem` of the Electronic Vision(s) group.

## 4.2 Framing and Packet Encoding

This section describes the framing of the physical bandwidth and the embedding of packets. It is repeated here for comprehensibility that the data of isochronous connections is transported within periodically reserved slots at fixed frame positions, whereas the data of a best-effort packet is placed in free data slots and can be distributed over multiple frames.

### 4.2.1 Format of a Data Frame

*general remarks*

The frame format equals the MCGN scheme, which is illustrated in figure 3.4: The bandwidth is divided into periodic data frames of the same (configurable) frame size $T$ at all links. Each frame consists of a frame gap of duration $G$ and $f$ consecutive data slots of the same size $S$. The selection of $T$ depends on the required reservation pattern of the application and results in achievable QoS delay and jitter results for connection-oriented traffic as shown in section 3.7. All data within the transport network is transmitted within 16 bit-wide data paths at a clock frequency of 156.25 MHz. The implemented frame format is shown in figure 4.2.

*synchronization header*

The frame starts with the synchronization header of 16 bit. The header contains the start-of-frame character `SOF` as well as the 8-bit field `GSS` for the implementation of the global synchronous signals. The SOF is encoded as the 8b/10b character K28.1[1] of the MGT to unambiguously detect the frame start.

*data slots*

The header is followed by the data part of the frame consisting of the data slots. All frames are of the same size and contain $f$ data slots. Section 3.7 discussed that

[1] Synchronization characters can be transmitted due to the 8b/10b encoding of the used MGT, which provides 12 extra *K-characters* distinguishable from data bytes. [157]

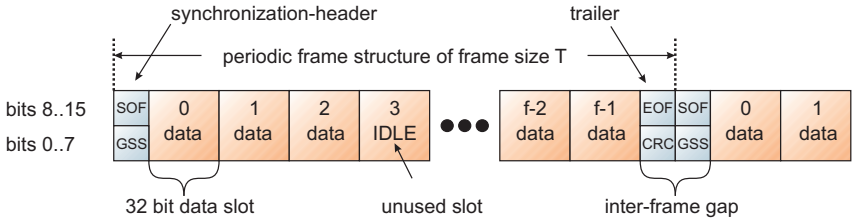



Figure 4.2: Format of a data frame. The granulation of the data slots has been reduced to 32 bit to achieve optimal QoS results. Unused slots are marked with `IDLE` symbols ($K$-characters of the MGTs) without the need of extra bits. The reservation pattern that assigns the traffic class to each slot is stored within the forwarding table of the switch.

the QoS results for delay and jitter for connection-based traffic are best for small data slot values $S$. Consequently, the size of a data slot has been selected to be 32 bit ($S = 12.8\ ns$) to carry a single neural network-event of 21 bits (cf. section 2.1.2). In other words, a single data slot is reserved for each neural network spike event.

*reservation periods*

The design of MCGN proposes the definition of $n$ reservation periods of $m$ time slots each that contain the same periodic reservation pattern such that $f = n \cdot m$. To save programmable logic, these reservation periods have not been implemented in hardware. However, the same effect can be reached by simply assigning periodic reservation patterns to all $f$ time slots resulting in the same QoS results. The optimum number of time slots $f$ per frame depends on the specific requests for isochronous connections and thus on the neural netlist and the configuration of the experiment to be carried out. For this reason, the value of $f$ has been implemented to be configurable at runtime for a generic hardware design. The present configuration allows a maximum selectable $f$ of $f = 128$ data slots.

*trailer and gap*

The frame ends with the trailer, which contains the CRC checksum computed over the data part of the frame. The CRC replaces the internal CRC of the MGT that has been deactivated to improve the data path delay. Since the checksum has to be calculated independently for all MGTs for transmit and receive, the size can be configured between 8 bit and 4 bit to trade the logic consumption against the data integrity. The frame-gap is defined as the non-data part of the frame. It therefore contains the trailer of the frame plus the synchronization header of the succeeding frame and is 32 bit large (12.8 ns).

### 4.2.2 Format of a Best-Effort Packet

*shared memory transports*

The format of an embedded best-effort packet has been defined according to section 3.8.2. Although the specification supports variable-sized packets, the present implementation uses a fixed size of 44 byte (11 data slots) for all packets. This simplifies the packet buffering within all layers, but can be changed easily if required later. The packet header is defined within the VHDL module `packet_pkg`. It is shown in figure 4.3.

*packet header*

The packet header consists of the SOP-character and the routing information, the protocol qualifier and the packet size (currently fixed). All information required for the local routing and forwarding processes are stored within the first 16 bit of the packet. That is, the destination node in terms of its backplane number and

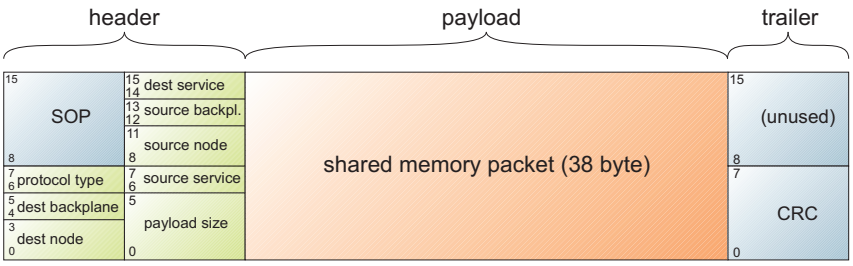



Figure 4.3: Data format of a best-effort packet.

its `Nathan` module number and the local protocol type identifier. This allows to calculate the local output port for an incoming packet within a single clock cycle to reduce the packet delay. The current implementation allows to address up to 4 backplanes, which results in the total of 64 usable `Nathan` network modules.

*payload*

The header is followed by the payload starting at the second slot with the header of the transported protocol. The current implementation only defines the *shared memory* protocol as payload using the protocol type identifier of 0. The best-effort packet size of 44 byte has been chosen to support up to two shared memory data units of 16 bytes each (cf. section 4.9.2).

*packet trailer*

The packet ends with the CRC checksum. Comparably to the checksum of the data frame, the size of the packet checksum is also configurable to up to 8 bit to trade the logic consumption against the data reliability.

### 4.2.3 Notification of the Slot Usage

The data of best-effort packets is embedded within data slots that are unreserved or unused by isochronous connections. The actual traffic class of the data within reserved slots has therefore to be detected by the switch at runtime. The notification whether a slot is reserved or used for data is stored differently:

*reserved slots*

The reservation pattern of the data slots is stored within the routing table of the switch and *not* within the data slot itself. At runtime, reserved data slots that do not contain valid data are occupied with the `IDLE` symbol, which is a special symbol of the MGT encoded with the 8b/10b characters K28.5 and D21.4 for both clock cycles. By verifying the content of the reserved slots for the presence if the `IDLE` character, the switch detects valid data without additional bits or the usage of headers within the slots.

*reserved, but unused slots*

The equipment of reserved but unused slots with packet-based data, is marked in bit 15 of the first cycle. The transport of neural network data is not affected since a neural network event occupies only 21 bit and this bit is unused. Packet-based data is then placed into the second remaining cycle of the data slot. The equipment of unused reserved slot with packet-based data therefore requires to investigate the first cycle of each reserved slot to detect its actual traffic class. Without this feature, only the presence of the `IDLE` character has to be checked and neither connection-based data nor packet-based data has to be further investigated. Table 4.1 summarizes the detection of the traffic class membership.

| reservation, usage | storage location |
|---|---|
| traffic class (reservation) | switch-internal routing table |
| usage of reserved slots | presence of `IDLE` character |
| usage of unreserved slots | presence of `IDLE` character |
| traffic class of reserved slots | bit 15 of first slot cycle |

Table 4.1: Detection of the traffic class of the data slots. The usage of slots for packet-based traffic that are reserved but unused by connection-based traffic requires an additional bit within each reserved slot.

## 4.3 The Physical Layer

*parts*

The physical layer of the framework consists of the physical interconnections between the `Nathan` network modules in terms of transmission lines, connectors and the basic bit-encoding and decoding processes. This also includes the distribution of the reference clock that is required for the global synchronization of all network nodes. The hardware of the physical layer is provided by the backplane of the framework as well as by the MGTs of the FPGAs on the 16 network modules per backplane (cf. section 2.2). The programmable part of the physical layer is implemented within the VHDL module `phys_sync`. The module provides multiple parallel interfaces for synchronous transmission of data between the programmable logic on the different network modules.

*MCGN requirements*

The specification of the MCGN architecture requires a physical layer with a constant data rate, a constant transmission delay and the possibility to transmit synchronization characters. Constant data rate and constant transmission delays are guaranteed due to the usage of a common clock source for all network nodes. Synchronization characters can be transmitted due to the 8b/10b encoding of the used MGT, which provides 12 extra *K-characters* distinguishable from data bytes. The usage of the FPGA-embedded MGTs therefore reduces the implementation of the physical layer to the careful distribution of the global reference clock and to the appropriate configuration of the MGTs.

### 4.3.1 Network Topology

*network topology*

Each FPGA of the setup provides eight bidirectional links according to the eight MGTs. The links use a separate differential track for each direction. Each FPGA can therefore be connected to eight others. The resulting network topology consists of multiple parts:

1. Four links are routed via the differential backplane connector to FPGAs on other network modules. The backplane provides hardwired transmission lines for this case.

2. The remaining four links are routed to the second differential connector on the top of the network module. Connections between network modules can be added with commercial serial advanced technology attachment (SATA) cables.

3. The single FPGA on the backplane is not included into the hardwired backplane topology. Its eight links are routed to separate connectors on the backplane.

The topology of the hardwired backplane network is illustrated in figure 2.8(a) and figure 2.8(b). It can be interpreted as a 2-dimensional toroidal structure, but also as a 4-dimensional binary hypercube. The four user links of each FPGA as well as the eight links of the backplane FPGA can be used to extend the hardwired topology, to increase dedicated links between network modules or to interconnect multiple backplanes. This single backplane FPGA can be used as a central network gateway to save programmable logic of the network modules or to interconnect the backplane to the control PC.

#### Delays of Transmission Lines

*less that half a clock cycle*

The length of the transmission lines on the modules are 3 cm to 10 cm, whereas the length of the transmission lines on the backplane are 3 cm to 40 cm. The longest lines on the network modules are connected to short connections on the backplane to even the different transmission delays. The total delays of the different lines is about 10 cm to 50 cm. With an expected signal speed of 20 cm/ns on the used FR4 material, this results in 500 ps to 2.5 ns of physical delay. Compared to the frequency of the global reference clock of 156.25 MHz, it can be stated that the delay of the physical transmission lines is less than half a clock cycle of 6.4 ns.

*comparable transmission delays*

This is important for the framework-wide synchronization as is results in the fact that the synchronization of the network modules can be established with the same transmission delay (in terms of clock cycles) between all modules. Variations in the transmission delays due to the sampling of the clock or the data can be canceled by an appropriate adjustment of the internal delay elements with only few cycles of delay.

### 4.3.2 Distribution of the Global Reference Clock

The reference implementation of MCGN on the Stage 1 framework uses a unique global clock source, whose signal is distributed to all network modules and to the FPGA on the backplane. This is for the following reasons:

1. The switching technology within the data link layer of MCGN is based on the global synchronization of all network nodes. To keep the synchronization stable, this requires a unique global clock reference, whose signal is distributed to all nodes (cf. section 3.5).
2. The MGTs require a reference clock signal for its proper operation. It is used for the serialization of network data to be transmitted and for the de-serialization of the data received. The usage of the same clock source avoids frequency discrepancies and thus removes the need for the clock correction logic, which results in a smaller overall transmission delay.
3. The user logic implemented within the FPGA requires a reference clock for the generation of further synchronous clock signals of different frequencies. These

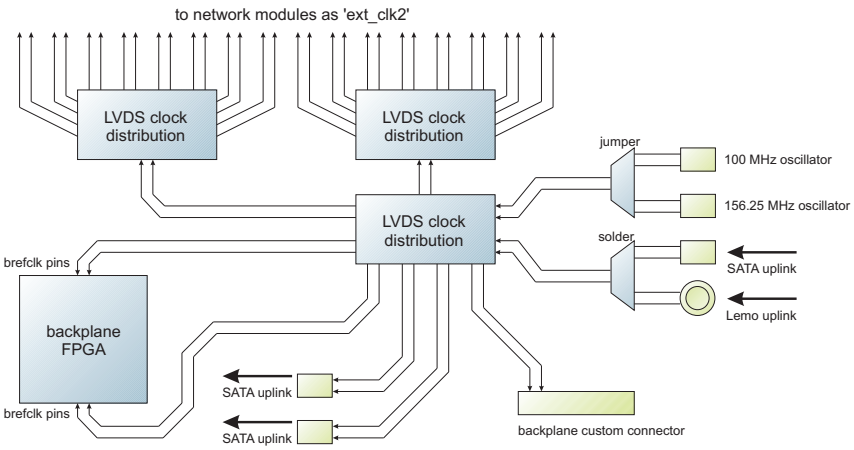



Figure 4.4: Global clock distribution on the 2nd version of the backplane. All transition lines are point-to-point differential with controlled impedance. The uplinks allow to distribute a unique reference clock to multiple cascaded backplanes.

clocks are needed for e.g. the SlowControl, the memory access or the operation of the PowerPC.

*data rates*

All three requirements are satisfied by using a single unique global clock source. The frequency of the clock is determined by the gigabit transceivers, since the frequency of the clock rate directly determines the usable data rate. The reference implementation uses an oscillator with a frequency of 156.25 MHz, which is the highest possible rate to be used with the MGTs. This equals an external data rate of 3.125 Gbit/s and a usable data rate of 312.5 MByte/s per link in each direction due to the 8b/10b encoding scheme. The data is presented in parallel with 16 bit at a rate of 156.25 MHz to the programmable logic at the internal MGT interface.

*global distribution*

The unique oscillator is located on one of the backplanes used. The comparably small size of the setup allows to distribute the global reference clock electrically to all network nodes on all backplanes. Each backplane provides extra connectors for a clock up-link and down-link that can be configured with jumpers to cascade the clock distribution. To keep the jitter as small as possible, the clock is distributed with dedicated chips such that only point-to-point differential transmission lines with controlled impedance are traversed within the clock tree.

*internal distribution*

The reference clock signal enters the FPGA at its bottom-side `brefclk` pin (AD14/AE14) under the name `ext_clk2_p/n`, which allows to use dedicated routing resources and improves the clock jitter for the bottom-side MGTs. The top-side `brefclk` pins are assigned otherwise and thus unavailable such that the `refclk` configuration has to be used instead for the MGTs located on the top of the FPGA fabric (for more details, consult section *clocking* in [157]). The clock signal is further forwarded to the system DCM[2] [154], which generates synchronous derived clocks for the other parts within the programmable logic.

[2]The digital clock manager (DCM) is embedded within the FPGA. Its is basically a delay-locked loop (DLL) circuit that performs frequency synthesis, clock de-skew and phase shifting.

### 4.3.3 The Multi-Gigabit Transceiver

*embedded serial transceivers*

The MGTs of the FPGAs are serial transceivers embedded as ASICs within the programmable logic. It consist of two parts, the physical media attachment (PMA) and the physical coding sublayer (PCS). The MGT is described in detail in [157]. The PMA contains a serializer deserializer (SERDES) circuitry, transmit and receive buffers, clock generators and a clock recovery circuitry. The PCS contains an 8b/10b encoder and decoder, an elastic buffer supporting channel bonding, a clock correction logic and handles the CRC checksum. The external serial transmission lines between the FPGAs are routed to the external pins of the MGTs. Internally, the MGT provides a convenient synchronous, digital and parallel interface to the programmable logic. A Schematic of a single MGT is shown in figure 4.5.

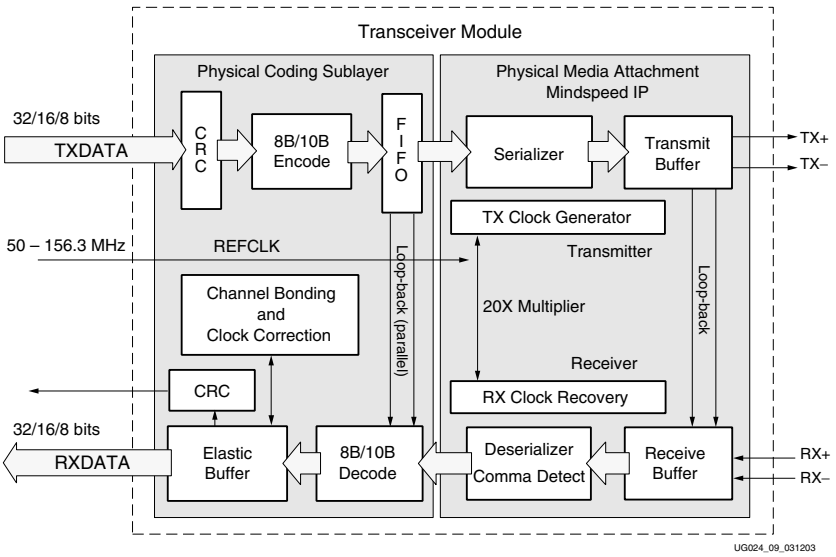



Figure 4.5: Block diagram of the multi-gigabit transceiver embedded into the FPGA of the network module. The MGT contains the PMA and the PSC of the physical layer. The physical serial transmission lines are routed to the TX/RX+/- pins, the internal interface to the programmable logic is synchronous and parallel. Diagram taken from [157].

*data encoding and clocking*

The MGTs use the 8b/10b technique to encode the external data [150]. The clock to sample the data is encoded into the data stream and recovered back at the destination. The MGT requires a reference clock for proper operation. The reference clock can be selected freely in the range of 30 MHz to 156.25 MHz with a resulting transmission rate of 600 Mbit/s to 3.125 Gbit/s. Since the frequency of the reference clock is multiplied internally by a factor of 20, the clock should have a frequency stability of +/- 100 ppm and the clock jitter is limited to about 40 ps at the highest possible data rate [157]. The jitter tolerance is higher for lower data rates, e.g. 330 ps at 800 MHz [16].

### 4.3.4 Configuration of the MGTs

Since the MGTs already contain the PMA and the PCS, the development of the physical layer is reduced to the configuration of the MGTs (cf. figure 4.6). The

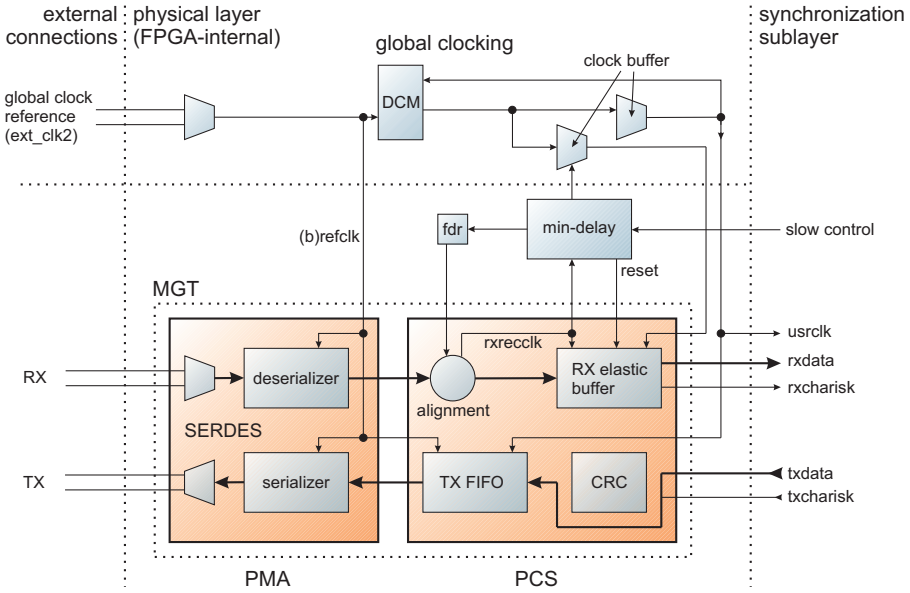



Figure 4.6: Block diagram of the physical layer within the programmable logic. The VHDL module `phys_sync` implements the MGT and configuration logic to reduce the delay of its data path. The upper part of the schematic shows the global clocking. The interface to the synchronization sublayer provides parallel transfers of 16 bit at 156.25 MHz.

configurable parameters concern all parts of the PMA and the PCS. Concerning the external transmission, the MGT can be configured for various differential standards such as PCI Express [108], Fibre Channel [37], Infiniband [62] and also SATA [120]. The reference implementation uses the `GT_CUSTOM` operation mode to allow a more specific configuration. The `MGT` configuration is summarized in table 4.2. A detailed description of all parameters can be found in [157].

*delay reduction*

To reduce the delay of the data path, the receive buffer is modified and the internal CRC logic is disabled (see below). This reduces the data path delay from 40 cycles to only 20 cycles (128 ns), measured from the `TXDATA` interface at the transmitting MGT to the the `RXDATA` interface at the receiving MGT. The electrical configuration parameters (voltage swing, pre-emphasis and clock selection) have been set for an optimal quality of the resulting signal (cf. section 5.1.1).

| parameter | value | feature | configuration |
|---|---|---|---|
| reference clock | 156.25 MHz | coding instance | GT_CUSTOM |
| external data rate | 3.125 Gbit/s | top MGT clocking | refclk |
| interface data path | 2 byte | bottom MGT clocking | brefclk |
| interface rate | 156.25 MHz | receive buffer | minimum delay |
| voltage level | 700 mV | transmit buffer | enabled |
| pre-emphasis | 33 % | CRC logic | disabled |

Table 4.2: Configuration of the multi-gigabit transceivers on the Nathan network modules

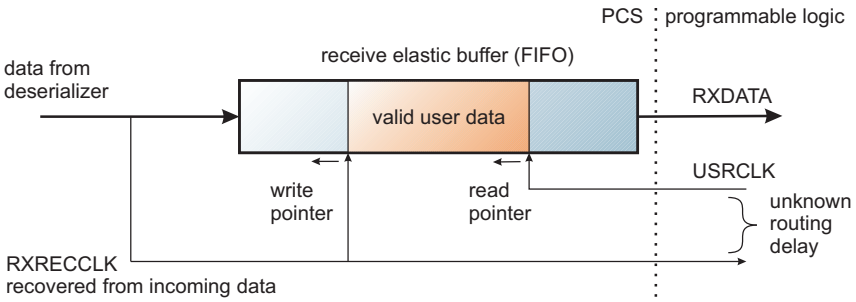



Figure 4.7: The MGT-internal receive buffer. Due to the unknown routing delay between the read and write clocks, the buffer cannot be disabled. Its delay is reduced by minimizing the distance between the write pointer and the read pointer.

### Delay Reduction

*delay reduction*

The application field of the reference implementation requires the transfer of neural network data within isochronous connections. Section 2.4.2 showed that this requires a transmission delay as small as possible for biologically relevant experiments at a certain acceleration factor compared to biology. Under normal conditions, the transmission delay of the MGT is in the range of about 40 clock cycles (256 ns) between the `TXDATA` signal at the transmitter and the `RXDATA` signal at the receiver [157]. Therefore, some modifications to the standard usage of the MGT are applied to reduce the overall delay. The MGT delay can be influenced by the following parameters:

- The bypass of the receive elastic buffer.
- The bypass of the transmit buffer.
- The deactivation of the transmit CRC logic.

The effects of these modifications are described in the following.

*buffer description*

**Adjustment of the MGT Receive Buffer** The receive elastic buffer of the MGT is a FIFO, which is located within the receive path of the PCS of the transceiver. Figure 4.7 shows a schematic of the buffer. The buffer stores $32 \times 2$ data bytes and features two pointers for writing and reading of data, respectively. The write pointer is moved on incoming data that is written to the buffer. This data arrives with the frequency of `RXRECCLK`, which is the clock recovered from the received bit stream. The read pointer is moved in the case data is read from the buffer, i.e. every cycle of the internal `USERCLK`.

*buffer usage*

The buffer is commonly used for three purposes:

1. The crossing of the clock domains from `RXRECCLK` to `USRCLK`.
2. The support for clock corrections if both clocks have different frequencies.
3. The support of channel bonding to group the transmissions of multiple MGTs.

Clock corrections concern the insertion and deletion of special *IDLE* characters within the data stream to compensate for different clock frequencies. The corrections are not needed since the Stage 1 framework distributes a unique clock source

to all network modules. Channel bonding is also not used with the setup. Since the frequencies of the recovered clock and the internal user clock are equal, the two pointers move with the same speed during operation. Therefore, the buffer remains always in its initial state (half-full), which introduces an unnecessary large average delay of about 18 clock cycles (115.2 ns) [155, 157].

*bypassing the buffer*

Although the frequencies of both clocks are equal, the phase relationship between it is still unknown. A complete bypass of the buffer would require to sample the received data with the `RXRECCLK`, but this solution is limited to 125 MHz [33]. To use the maximum possible frequency of 156.25 MHz, the buffer cannot be completely deactivated [156, 157].

*pointer adjustment*

However, the delay of the buffer can be reduced by initially modifying the buffer pointers such that the write pointer is placed just before the read pointer and both pointers move in parallel. This is achieved by disabling the `USERCLK` and thus stopping the read pointer for about 18 cycles, while the `RXCECCLK` is continuously moving the write pointer until it is placed zero or one cycle before the read pointer. The `USRCLK` is then turned on and the two pointers continue to move with that close distance at the same speed. The described procedure reduces the delay of the receiver 14 clock cycles or 89.6 ns. The manufacturer of the FPGA provides a VHDL design that does the described work [77]. It has been integrated into the module of the physical layer (cf. again, figure 4.6).

*transmit buffer not bypassed*

**Bypass of the Transmit Buffer** The transmit buffer acts as a FIFO and is used to cross the clock domain of the internal `USRCLK` to the reference clock `(B)REFCLK`. Both clocks always have the same origin, but an unknown phase shift since both clocks are distributed via different paths to the transceiver. The `USRCLK` is distributed with global clocking resources, whereas the `REFCLK` uses dedicated routing to reduce the jitter. Since the transmit buffer is used only to cross the clock domain between these two clocks, it is therefore of small size. The bypass of the buffer is not recommended [156, 157], hence the buffer is enabled for the reference implementation.

*CRC deactivated*

**Disabling of the CRC Generation** The MGT supports the automatic generation of a 32 bit CRC checksum into the data stream, which is used by the protocols InfiniBand, Fibre Channel and Gigabit Ethernet. The generation of the CRC introduces 6 clock cycles (38.4 ns) of delay. The receive data path is not affected by the decoding process. Since a CRC checksum can be calculated easily within the programmable logic [8, 65] and since the overall delay has to be reduced as far as possible, the CRC is deactivated for the reference implementation.

## 4.4 The Synchronization Sublayer

This section describes the synchronization sublayer, which is specified by MCGN to be implemented on top of the physical layer. It uses the synchronous parallel interface provided by the physical layer to transmit data between directly connected nodes of the Stage 1 framework. The synchronization sublayer provides the following services:

*provided services*

- The framing of the bandwidth according to the selected number $f$ of time slots per frame and the selected slot shift $s_e$ per frame on each link $e$.
- A synchronization of the timing of the network nodes and thus a deterministic transmission of the frame data.
- The provision of a synchronization service for upper-layer proceses to synchronize high-level events.

Since the framework-wide synchronization is established during the network initialization phase, the implementation of the sublayer is split into hardware and software parts. The hardware part has been implemented within the VHDL module `phys_sync`. The software program `mgtsync` accesses the module via the SlowControl and performs the measurement and the algorithms for the delay and timer adjustments that have been described in section 3.5.5. The software is described separately in section 4.10.1.

*hardware overview*

A block-level overview of the synchronization module can be seen in figure 4.8. The module contains the local time counter and the adjustable delay elements as well as the framing logic. The framing logic implements the two framing strategies of MCGN, namely fixed framing and shifted framing (cf. section 3.5.6). The two data paths for transmission and reception are independent of each other and have been implemented separately. The synchronization layer also implements the GSS functionality of MCGN. The signals can be used by the neural network application to synchronously start an experiment on all network nodes.

*timing and framing*

Section 3.5.6 showed that the usage of large frame sizes requires the shifting of the frame boundaries during the framing process. This enforces the framing logic to be implemented in conjunction with the timing and synchronization logic. Therefore, the logic implemented within the `phys_sync` module contains the time counters and the delay elements, but also the compilation of the frame header and trailer, the calculation of the CRC checksum and the multiplexing and demultiplexing of user data into the frame slots.

*absence of traffic classes*

Concerning the data content, it it important to understand that the user data within the data slots is transmitted and received without investigating its content. The synchronization layer is neither aware of the reservation pattern nor of the existence of traffic classes. The multiplexing of connections and packets into the data stream is performed purely by the bypass-switch in the layer above. The synchronization sublayer solely provides the service of transmitting the data slots within synchronized frames between the nodes.

### 4.4.1 Timing of the Network Node

*synchronization condition*

To synchronize the network, the timing of each network node has to be adjusted according to the synchronization definition 3.5.1. For that reason, each network node contains a local periodic time counter with the period of the globally constant duration of the time frame $T = 2f + 2$ clock cycles. A network node is in synchronization state, if all frames arrive properly aligned at all MGTs such that the first data slot of each frame is received at the local time 0 (modulo $T$).

*time distribution to upper layers*

The local timer of a synchronized network node controls the transmission and the reception of all data frames. Since the switching of the reserved slots of isochronous

connections is done without buffering, the timing of the synchronization sublayer is distributed to all layers above the synchronization sublayer that handle connection-based data. The bypass-switch provides the timing of the two data paths at the user interface for connection-based data (cf. again, figure 4.8). Concerning packet-based transfers, the packets are buffered at different locations within the switch. Its transmission and reception depends on the buffer spaces and not on the switch timing.

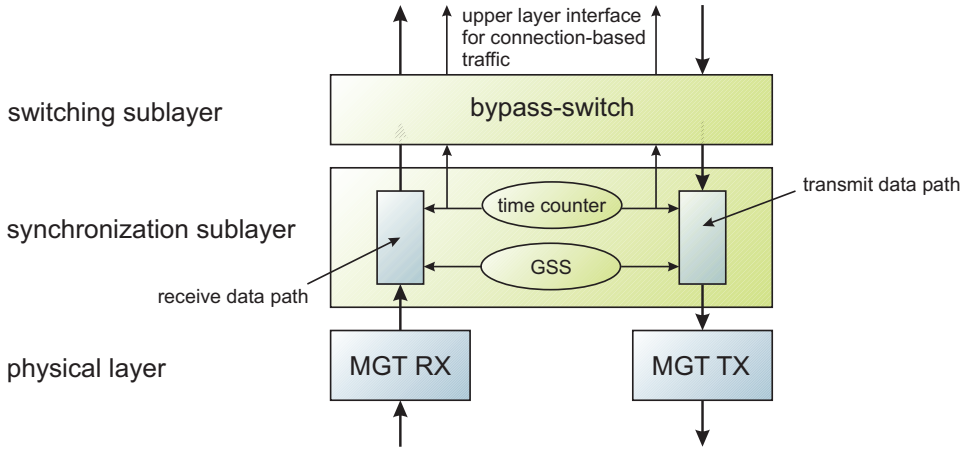



Figure 4.8: Distribution of the local time. The periodic time counter controls the synchronization of the network node to the network. This requires to control the timing of all transmit and receive processes with respect to the local time.

### 4.4.2 Reception of Data

*extraction of traffic qualifiers*

Figure 4.9 shows a schematic view of the receive data path of the synchronization sublayer. The logic is implemented separately for each MGT. The data arrives at the `rxdata` and `rxcharisk` outputs of the physical layer. There is no need to buffer the data within the synchronization sublayer, but its content has to be investigated for the following reasons:

- The de-framing logic detects used or unused slots by checking the presence of an `IDLE` symbol. The result is denoted via the `rxvalid` signal.
- The synchronization logic monitors the correct appearance of the synchronization character `SOF` to verify the correct synchronization. The time of appearance is stored to be read by the synchronization software part to measure the synchronization state.
- The CRC checksum is verified.
- The GSS logic checks for incoming GSS signals (see below).

*zero delay*

Since the data is only evaluated but not modified, the delay of the receive data path of the synchronization sublayer is zero. The data is simply forwarded from the physical layer to the switching layer above.

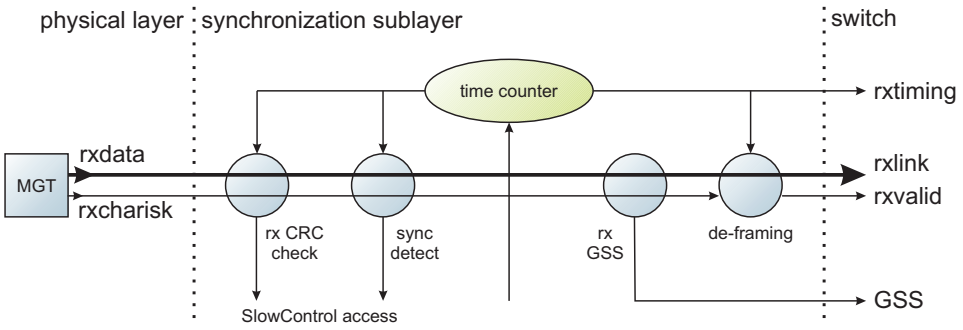



Figure 4.9: Schematic of the receiver data path of the synchronization sublayer. The thick line denotes the data flow from the physical layer through the synchronization sublayer to the switch. The circles denote logic that investigates the data content to extract runtime information. The delay of the data path is zero.

### 4.4.3 Transmission of Data

*compilation of the frame*

Figure 4.10 shows the transmit data path of the synchronization sublayer. Data to be transmitted by the switch first enters the adjustable delay element. The element is implemented using the compact realization within a `SRL16` element of the FPGA, which is a shift register implemented in a single LUT for each data bit. The shift register allows to change the delay value at runtime in the range of 1 to 16 clock cycles. The data is then forwarded to the framing logic, which adds the header, the trailer, the calculated CRC checksum and the GSS content. The framing logic generates `IDLE` symbols to occupy the space of unused data slots depending on the `txvalid` signal.

*variable delay*

If the shifted framing technique is used, the framing logic also controls the delay element to insert and to remove the frame gap at the correct position (see below). The delay of the transmit data path of the physical layer calculates to $2 + \epsilon$ clock cycles, where $\epsilon$ is the value of the delay element.

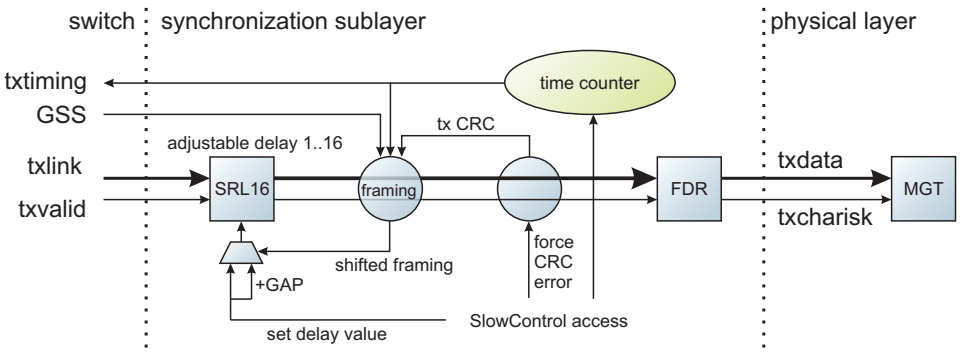



Figure 4.10: Schematic of the transmit data path of the synchronization sublayer. The circles denote asynchronous logic, whereas the boxes denote data buffers (digital registers). Data to be transmitted is delayed by a configurable amount of clock cycles. The framing logic compiles the frame and controls the insertion of the frame gaps to implement the shifted framing technique of MCGN.

### Realization of the Shifted Framing Technique

*motivation*

The shifted framing technique of MCGN moves the frame boundaries and thus the frame gap (i.e. the trailer and the header) between the data slots by a certain number of slots. The technique has three advantages:

- To use arbitrary frame sizes $T$ independent of the network topology.
- To strictly synchronize the time counters of all network nodes.
- To reduce the delay in the data path that is required for the synchronization to a minimum.

The shifted framing technique is described in detail in section 3.5.6.

*implementation*

The shifted framing technique is implemented within the transmit data path of the synchronization sublayer. The feature can also be disabled to save programmable logic in the case it is not required. Shifted framing is implemented by controlling the adjustable delay element. The delay value is changed by the duration of the frame gap (2 cycles) such that the data slots are grouped according to the required frame timing at the destination node. The header and the trailer are written afterward into the frame gap. This solution omits the buffering of the transmit data and reduces the logic consumption.

*simulation example*

Figure 4.11 shows a simulation of the transmit timing path that illustrates the implementation of shifted framing. The switch transmits data within the data slots 0 and 1, which is denoted by the `txvalid` signal. The data slot 2 is unused (`txvalid`=0). The time period $T$ has been selected to 30 clock cycles (192 ns), which implicates 14 usable data slots within the positions 0 to 13. The network topology requires to move the frame gap by 2 data slots backwards ($s_e = 12$). The data received from the switch within the first two slots 0 and 1 is thus sent within the last two data slots 12 and 13 to the destination. The delay element is switched between the delay of 1 and 3 cycles according to the size of the gap of two cycles.

After the timing of the data slots has been corrected, the framing is made, i.e. the trailer and the header are written into the frame gap. The unused data slot that has originally been at position 2 is now at the position of the first data slot 0. Its data is marked unused with the `IDLE` character K28.5 and D21.4 (encoded as $BC95$). Note that the shifted framing technique does only vary the delay of the data slots by the (comparably small) size of the frame gap to cope to the timing at the destination node. The order of the transmitted data slots is preserved.

## 4.4.4 Selection of the Synchronization Parameters

The synchronization software has to program the correct synchronization parameters to the synchronization sublayer within the FPGA during the network initialization phase. The following paragraphs discuss the valid values for the synchronization parameters, namely the possible values for the number $f$ of time slots per frame and the slot shift $s_e$ as well as the required adjustments of the delay elements $\epsilon$ in the data path and the resulting transmission delays $D$ between the inputs of adjacent switches. The numbers are calculated for the Stage 1 framework. The timing of the MCGN network between the Nathan network modules is determined by the following factors (cf. also, figure 3.12):

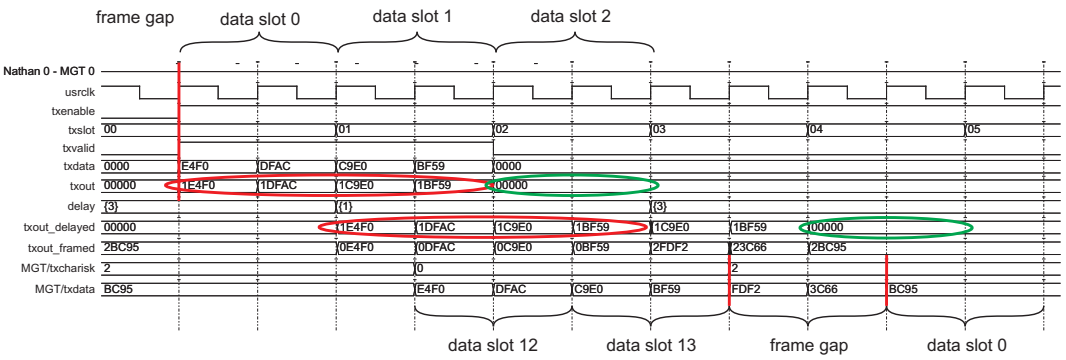



Figure 4.11: Simulated timing of the shifted framing implementation within the transmit data path. The re-framing introduces only the small delay of the frame gap.

*timing constraints*

- The physical delays of the transmission lines between the FPGAs on the `Nathan` network modules. The delays are less than half a clock cycle of 6.4 ns.
- The delay of the physical layer, i.e. the PMA and the PCS of the delay-reduced MGT. The delay from the `txdata` interface to the `rxdata` interface at the destination has been simulated using the provided SWIFT models of the FPGA manufacturer to be 20 cycles. Measurements result in 20 to 21 cycles depending on the affected network nodes (cf. section 5.2.1). The value may differ even for the same MGT in both directions due to data and clock sampling effects.
- The delay of the synchronization sublayer. The receive data path introduces no delay whereas the transmit data path introduces 2 cycles plus the value of the adjustable delay element.
- The internal delay of the switch between its global input and output ports. The present implementation of the switch introduces only a single cycle of delay for the traversal of the internal crossbar.

*resulting transmission delay*

The total transmission delay $D$ between the inputs of two adjacent switches depends on the value of the delay element. It has to be adjusted to cancel asymmetries in the bidirectional physical transmission delays of the links. Due to the regular topology of the network, the following discussion assumes a delay of the physical layer of 20 clock cycles as stated above. The actual adjustment of the delay element may require modifications to the calculated values of up to two additional cycles. According to the mentioned internal delays, the value of $D$ calculates to $23 + \epsilon$ cycles, which implicates an delay of the programmable logic of only 3 cycles in the case the delay element is set to zero. The delay of the programmable logic is therefore only 13 % of the overall transmission delay between the switches. The remaining delay of 87 % is introduced by the (fixed) delay of the MGTs, even after its delay-reductions.

*number of slots per frame*

The implemented neural network application uses an inter-frame gap $G$ as well as a data slot size $S$ both of 2 clock cycles. Since the gap cannot be used for data, the number $f$ of data slots calculates to:

$$f = \frac{T - G}{S} = \frac{T - 2\gamma}{2\gamma} \tag{4.1}$$

for values of $T, S$ and $G$ to be measured in multiples of clock cycles $\gamma = 6.4\,ns$. The possible values for $T$ depend on the synchronization mechanism used. The next two sections describe the possible framing parameters for both framing techniques.

### Parameter Selection using Fixed Framing

*possible frame sizes T*

The first framing technique to be discussed is fixed framing (cf. section 3.5.6). With fixed framing, the frame boundaries are kept constant ($s_e = 0$ on all links $e$) and the selection of the frame size $T$ is directly constrained by the transmission delay $D$ between the switch inputs. The synchronization condition requires equation 3.16 to hold on all global links, which leads to discrete values for $T$ and $f$. Fixed framing requires to adjust the delay elements $\epsilon$ such that $D$ is equal on all network nodes of the backplane according to $\epsilon = D - 23$. Possible configurations are listed in table 4.3. To illustrate the timing of the fixed framing technique, figure 4.12 shows a timing diagram for a selected frame size of $T = 24$.

| set | number of time slots $f$ | frame size $T$ [cycles] | delay element $\epsilon$ [cycles] | transmission delay $D$ [cycles] |
|---|---|---|---|---|
| T=D | 11 | 24 | 1 | 24 |
| | 12 | 26 | 3 | 26 |
| | 13 | 28 | 5 | 28 |
| | .. | .. | .. | .. |
| | 18 | 38 | 15 | 38 |
| T=2D | 23 | 48 | 1 | 24 |
| | 24 | 50 | 2 | 25 |
| | 25 | 52 | 3 | 26 |
| | .. | .. | .. | .. |
| | 37 | 76 | 15 | 38 |

Table 4.3: [Theoretical values for the synchronization parameters of the backplane using fixed. The calculations assumes a physical layer delay of 20 cycles for the delay-reduces MGT. The two sets of numbers refer to examples 1 (upper) and example 2 (lower) of section 3.5.6.

*disadvantage*

Due to the regular topology of the backplane, fixed framing can indeed be used to implement large frame sizes with $T = 2D$ according to example 2, but with the disadvantage that the network nodes are split into two disjoint sets, whose timers are constantly shifted between each other.

### Parameter Selection using Shifted Framing

Shifted framing allows arbitrary values for $f$ and $T$ on all network topologies with only minimum required values of the delay elements (cf. section 3.5.6). The synchronization condition for shifted framing is described by equation 3.18. All timers can be strictly synchronized, i.e. $\delta = 0$ for all adjacent network nodes. As a drawback, there are two disadvantages: a slightly higher consumption of programmable logic

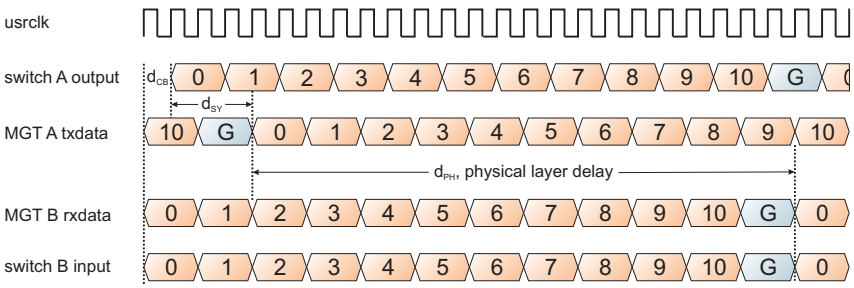



Figure 4.12: Timing scheme between two synchronized network modules A and B with a time frame size of $T = 24$ cycles and 11 usable data slots. Due to the avoidance of buffering, the internal delay of the programmable logic is only 4 cycles. 20 cycles are consumed by the physical layer (i.e., the MGTs).

and a difference in the delay of certain data slots of different positions. The transmission time $D$ is split into two parts $D_<$ and $D_>$ concerning the first and the latter part of the frame slots with the slot positions $i < f - s_e$ and $i \geq f - s_e$, respectively.

*arbitrary frame sizes $T$ with minimum delay*

The delay element $\epsilon$ can be reduced to the smallest value such that equation 3.18 results in integer values for $D$. Since $S = 2$ cycles, this results in $\epsilon = 1$ with $D_< = 24$ cycles and $D_> = 26$ cycles. Furthermore, this results in

$$s_e = 12 \mod f. \tag{4.2}$$

This is the shift value for the Stage 1 framework that results in the smallest transmission delays (24 cycles and 26 cycles for the two frame parts) between the inputs of two adjacent network nodes depending on the selected number of time slots $f$. The number $f$ can therefore be selected only according to the value that results in the optimal QoS delay and jitter depending on the isochronous connection requests of the neural network experiment to be executed.

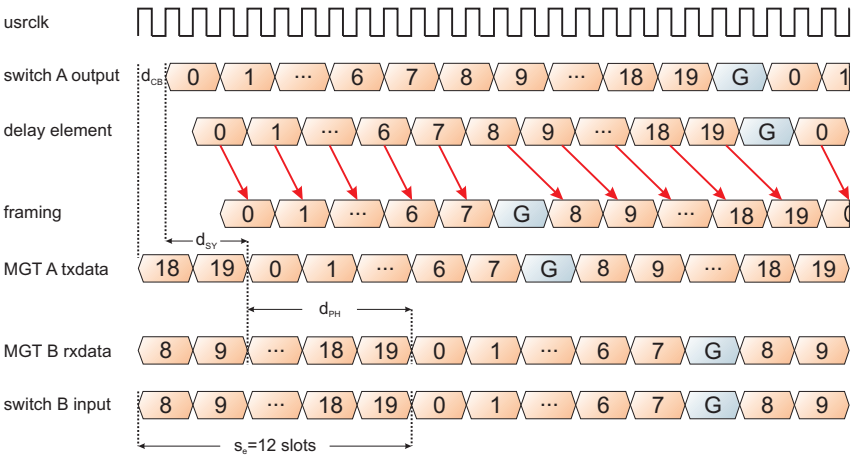



Figure 4.13: Example timing scheme between two synchronized network modules A and B with a time frame size of $T = 42$ cycles and $f = 20$ slots per frame. A slot shift $s_e$ of 12 ensures proper synchronization with a minimum adjustable delay of 1 resp. 3 cycles. The gap is inserted after $f - s_e = 8$ slots.

| number of time slots $f$ | frame size $T$ [cycles] | slot shift $s_e$ [slots] | delay element $\epsilon$ [cycles] | transmission delays $D_</D_>$ [cycles] |
|---|---|---|---|---|
| 9 | 20 | 2 | 1/3 | 24/26 |
| 9 | 20 | 3 | 3/5 | 26/28 |
| 10 | 22 | 1 | 1/3 | 24/26 |
| 10 | 22 | 2 | 3/5 | 26/28 |
| 11 | 24 | 0 | 1 | 24 |
| 11 | 24 | 1 | 3/5 | 26/28 |
| 11 | 24 | 2 | 5/7 | 28/30 |
| 12 | 26 | 0 | 3 | 26 |
| 12 | 26 | 1 | 5/7 | 28/30 |
| 12 | 26 | 2 | 7/9 | 30/32 |
| 13 | 28 | 1 | 7/9 | 30/32 |
| 13 | 28 | 0 | 5 | 28 |
| 13 | 28 | 12 | 1/3 | 24/26 |
| 14 | 30 | 12 | 1/3 | 24/26 |
| 14 | 30 | 13 | 3/5 | 26/28 |
| 14 | 30 | 0 | 7 | 30 |
| 20 | 42 | 12 | 1/3 | 24/26 |
| 30 | 62 | 12 | 1/3 | 24/26 |
| 60 | 122 | 12 | 1/3 | 24/26 |
| 60 | 122 | 13 | 3/5 | 26/28 |

Table 4.4: Theoretical values for the synchronization parameters of the backplane using shifted framing. The framing scheme allows arbitrary frame sizes $T$ for an optimal setting of the number of time slots $f$. The transmission delays $D_<$ an $D_>$ denote the constant values for time slots of both parts of the frame. The calculation assumes a physical layer delay of 20 cycles.

Table 4.4 lists valid synchronization parameters and resulting transmission delays for the shifted framing technique. Slightly larger values for $\epsilon$ may be necessary to compensate for different physical delays or data and clock sampling shifts. Figure 4.13 illustrates an example timing for a selected frame size of $T = 42$ with $f = 20$ time slots and a frame shift of $s_e = 12$ slots.

### 4.4.5 Global Synchronous Signals

*motivation*

MCGN defines the support of global synchronous signals (GSS) and CSS for the globally synchronous execution of local events at all network nodes at the same global time (cf. section 3.5.8). The events are triggered with the precision of the synchronization, which is a cycle of the global reference clock. An example usage is the synchronous start of neural network experiments on all network nodes.

*embedding*

The support for GSS has been implemented within the `sync_signals` VHDL

module according to the description of section 3.5.8. The present implementation of the module only implements the transmission of synchronized signals to all network nodes (GSS) and not between the two end-points of a connection only (CSS). The module is connected to the receive and transmit path of the synchronization logic according to figures 4.9 and 4.10. The signals are transmitted within the reserved `GSS` field of the synchronization header of figure 4.2. The meanings of the 8 bit of the `GSS` field are shown in figure 4.14. Two different signal-IDs are supported: $GSS\#0$ and $GSS\#1$.

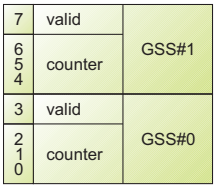



Figure 4.14: Bit meanings of the 8-bit `GSS` field within the frame header. Two different signals can be raised independently to synchronously trigger global events.

*operation*

Each network node can raise a GSS by asserting its corresponding `raise` signal at the local `GSS` interface. The corresponding counting value is then initialized and distributed over the network. The counting value decrements in time and space with the time period $T$. At the time the counter reaches zero, the GSS is executed, i.e. the corresponding `event` signal is triggered locally at all network nodes at the same time.

*implementation*

A schematic of the implemented VHDL module is shown in figure 4.15. Each GSS signal-ID has its own module. The internal counter is set at the time a valid signal is detected within the receive data path of the synchronization sublayer or the signal is raised locally via the GSS user interface. The value of the counter is decremented and forwarded to each MGT via the transmit data path. After the internal counter reaches the value zero, the corresponding GSS event is denoted to the local user logic.

*high-level usage*

The high-level user interface has been implemented according to section 3.5.8. The meaning of each GSS is not pre-defined but depends on the application. The signal distribution requires a delay of about several time frames $T$ until it reaches all nodes. A generic usage would be to use a GSS to initially synchronize local timers within the applications on all network nodes. The timer value can then be referenced in further faster (unsynchronized) communications. As an example, the GSS can be used to synchronously start the neural network experiments on all nodes.

## 4.5 Implementation of the Bypass-Switch

The bypass-switch performs the integration of the two traffic classes: isochronous connections and best-effort packets (cf. section 3.8.3). The switch uses the services provided by the synchronization sublayer to transmit and receive its data. The followings paragraphs describe the implementation of its main functionalities without going into the very details.

*universal design*

Due to the motivation of this thesis, the implementation of the switch is opti-

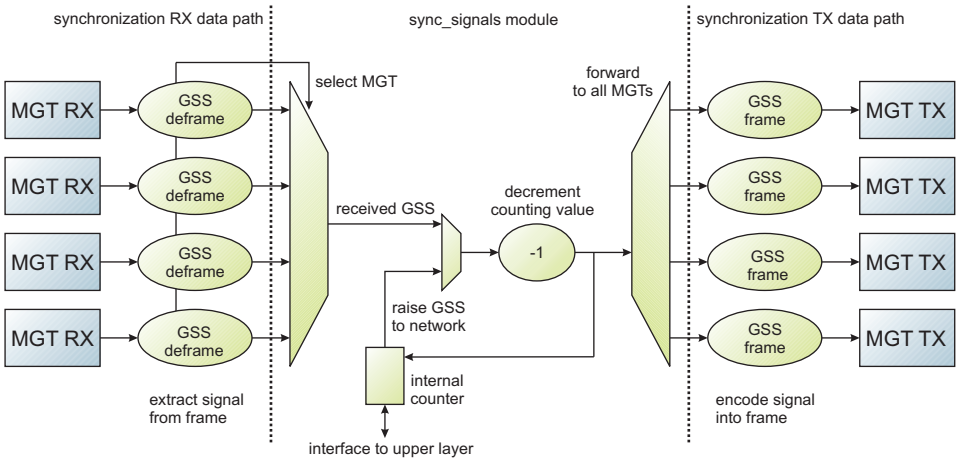



Figure 4.15: Schematic view of the GSS implementation for a single signal-ID. Received signals are forwarded to all adjacent network nodes. The signals are extracted out of incoming frames or raised locally. The down-counter decrements in time and space with the period $T$. The signal is triggered locally on all network nodes at the same time.

mized for the transport network for ANN experiments on the Stage 1 framework. Most of the design considerations have been made to guarantee good QoS-results for connection-based traffic and to reduce the required amount of programmable logic as far as possible. However, the implemented switch is also a reference implementation of the MCGN bypass-switch in general. Due to the configurable design, the implementation can well be used for other applications on the Stage 1 framework, and furthermore within different frameworks that allow synchronized transmissions.

### 4.5.1 Characterization

The implemented bypass-switch provides the following services: *provided services*

1. The in-order transport of priority traffic within isochronous connections with guaranteed QoS. The switch uses an optimized data path for connection-based traffic to reduce the inter-switch delay and jitter to the minimum possible. The injection of data slots to the network is controlled by the slot admission policy to control the data flow.
2. The transport of best-effort data within packets. QoS for packets is provided statistically. Internal buffering of packets allows to cope with variable packet rates. Packet forwarding is controlled by a central packet scheduler. The data integrity of packets is controlled with CRC checksums.
3. Multiple protocols are supported with only minor adaptation for both traffic classes. The packet payload is not investigated, which allows the usage of a wide range of protocols and applications.

Besides this, the implemented switch offers the following features: *main features*

1. A modular and hierarchic design. The implemented buffering technique, crossbar and packet scheduler are parts of standardized functionality. They can be

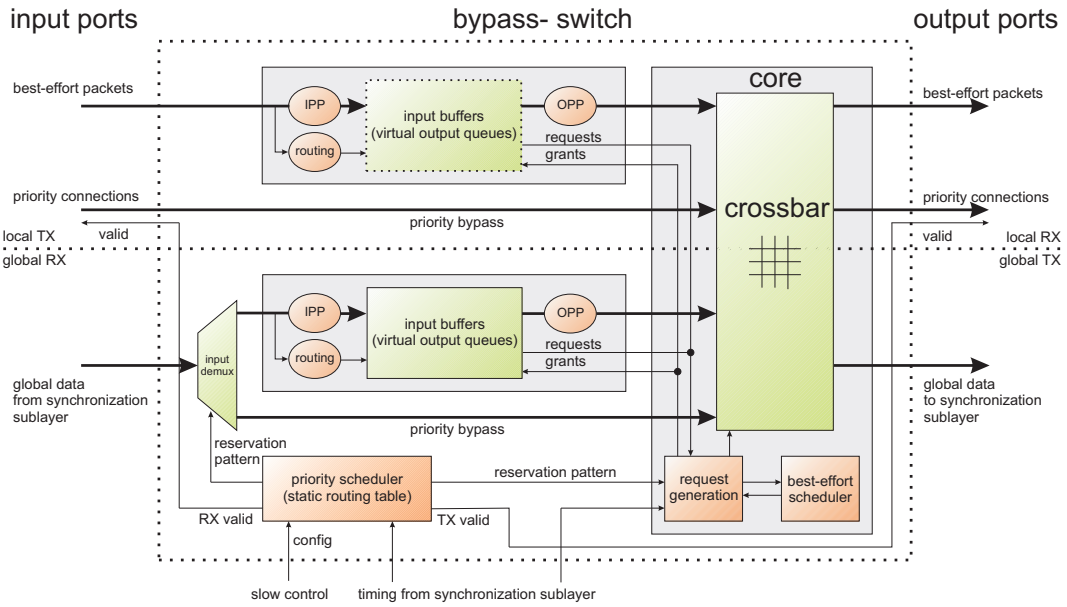



Figure 4.16: Top-level view of the implemented bypass-switch. The data is passing the switch from left (input ports) to right (output ports). The data paths are denoted by thick lines. Priority traffic is forwarded without any buffering, packets are stored in virtual-output queues.

replaced by different implementations easily by preserving the main feature of traffic class integration of the switch.

2. The compact and scalable design is optimized for the usage within the limited space of programmable logic.
3. A rich set of configuration parameters allows to find the best trade-off between the supported features and the amount of programmable logic needed.

*prerequisites*

The switch requires a synchronized physical layer for operation. Concerning the transport network, the switch operates on top of the synchronization sublayer that has been described in the previous section. The synchronization sublayer also provides the necessary timing information to reference the data slots within the periodic framing scheme. Prior to the network operation, the static routing tables for the isochronous connections are configured. The tables are accessed via the configuration software `mgtroute` (cf. section 4.10.3).

### 4.5.2 Overview of the Switch

*digital design*

The bypass-switch is implemented within the VHDL module `switch`. Figure 4.16 shows the top-level view of the switch. It consists of several sub-modules for buffering, multiplexing, switching, scheduling and packet processing. Most of the modules belong either to the transport of priority traffic or to the transport of packet-based traffic. The central module `core` integrates both traffic classes. It contains the crossbar multiplexer, the request logic and the multi-port scheduler.

*priority and best-effort traffic*

The scheduling of the traffic classes is done hierarchically: The priority scheduler

reads the static routing table that contains the reservation pattern of the connections. Reserved data slots are forwarded through the switch according to the external frame timing provided by the synchronization sublayer. The remaining unreserved or unused data slots are scheduled by the best-effort packet scheduler within the core according to the requests of the input-queues.

### Global and Local Ports

*hybrid network node*

The MCGN specification of section 2.2.3 defines that a network node may implement both, the forwarding functionality of an intermediate switch (via global ports) as well as the application functionality that acts as sources or destinations of data transfers (local ports). This has been implemented accordingly for the bypass-switch. Each FPGA on a `Nathan` network module may implement the forwarding of transit data between other modules as well as the application that interfaces its local ANN chip or the shared memory.

*port configuration*

The switch can be configured with a variable number of ports. An $N$-port switch features $N$ bidirectional ports, separated into $N$ input ports and $N$ output ports. Input ports are shown on the left side of figure 4.16 and output ports on the right side. The top-level view of the switch is divided into a local part and a global part. Global ports ($N_g$) are connected to the synchronization sublayer and thus to other network nodes. Local ports are connected to upper layer processes and can be distinguished as *priority ports* and *best-effort ports*. Local priority ports ($N_p$) implement the isochronous interface to upper-layer processes whereas local best-effort ports ($N_b$) are connected to the packet adaptation layer (PAL). The total number of ports therefore calculates to

$$N = N_g + N_p + N_b. \tag{4.3}$$

The example illustration of figure 4.16 shows the case for $N_g = N_p = N_b = 1$. Concerning the Stage 1 framework, a usual configuration of the switch would be to implement a single local best-effort port for shared-memory transfers ($N_b = 1$), few local priority ports ($N_p = 1..4$) and four global ports to use the full hardwired connectivity of the backplane ($N_g = 4$).

*logic reduction*

To reduce the logic consumption of the switch, the input queue that is connected to local best-effort ports to transmit packets can be omitted such that the port is blocked until the crossbar is available. Further logic can be saved by configuring the switch to reject (drop) transmissions between the same port numbers (internal or external loop-backs).

### Data Path of Traffic Classes

*multiplexing and demultiplexing*

Both traffic classes have to be multiplexed and demultiplexed at the global switch ports. The demultiplexing of the traffic classes takes place at the time global data slots enter the switch. The multiplexing is performed by the central crossbar located in the switch core. Data slots of priority connections or of best-effort packets enter the core at different inputs and leave the core multiplexed to the global output port of the switch. The crossbar is controlled by the request generation logic according to

the actual scheduling of the best-effort packet scheduler and the priority reservation pattern.

*data paths*

Priority data slots arriving at global switch inputs are forwarded directly to the central crossbar with only few cycles of delay, depending on the configured switch timing (see below). Best-effort data slots are re-assembled by the input packet processor (IPP) to packets and stored into the appropriate local VOQ that is calculated by the routing process according to the packet destination. Erroneous best-effort packets with an invalid CRC or packets arriving at full queues are dropped without further notifications.

*store-and-forward packet switching*

The switch implements *store-and-forward* packet switching at the VOQs. This is enforced by the method used by MCGN to embed best-effort packets in between reserved slots. It results in the fact that the exact arrival time of the last slot of the packet is unknown. Especially in the case that only reserved but unused slots are available for packet transports, the last data slot may arrive a significant time after the one before. An early request of the scheduler as used with *wormhole* switching or *cut-through* switching would block the data path for an unreasonable time, since best-effort packets are not intermixed. As a drawback, this forces the switch to completely store incoming packets before the central scheduler can be requested, which introduces a latency of a full packet-time at the VOQ.

### 4.5.3 Implementation of the Input Buffers

The input-buffers for best-effort packets have been implemented as VOQs as described in section 3.8.3. The queues are implemented together with the IPP and parts of the request generation logic within the VHDL module `voq`.

*input packet processor*

The IPP gathers the slots belonging to the current best-effort packet out of the slotted data stream by observing the reservation pattern. The calculation of the destination (output) queue is performed by the routing algorithm after the first cycle of the packet has been received. This is possible due to the organization of the packet header (cf. figure 4.3). In the case the queue has enough space and the CRC of the packet is valid, the packet is written into the queue.

*queuing stage*

The implementation of the VOQs exploits the fact that only a single packet arrives and departs at a time. This allows to use a single FPGA-internal DPBRAM to implement all the queues required for the packet-outputs at a certain switch input ($N_g + N_b$ queues). The functionality is implemented within the general-purpose VHDL module `multi_fifo`. Figure 4.17 shows a schematic of the VOQ organization.

*queue size*

A single DPBRAM features 18 kbit of memory, which allows to store 2 kByte of usable data. As an example, a switch with $N_g + N_b = 8$ offers buffering space for 256 bytes or 8 packets of 32 bytes each. The current implementation of 44-byte best-effort packets allows to store up to 4 packets within each queue. Multiple memory elements per input port can be used to increase the buffering space.

*storage efficiency*

The fixed size of each packet simplifies the calculation of the storage address and thus saves programmable logic. In the case the larger logic consumption is acceptable, the storage efficiency can be increased by changing the FIFO algorithm to a dense storage of variable-sized packets or to support variable-sized queues (an example implementation of dynamic FIFO queues using linked lists can be found in [76]). It has to be stated that large buffer sizes indeed reduce the loss-rate, but

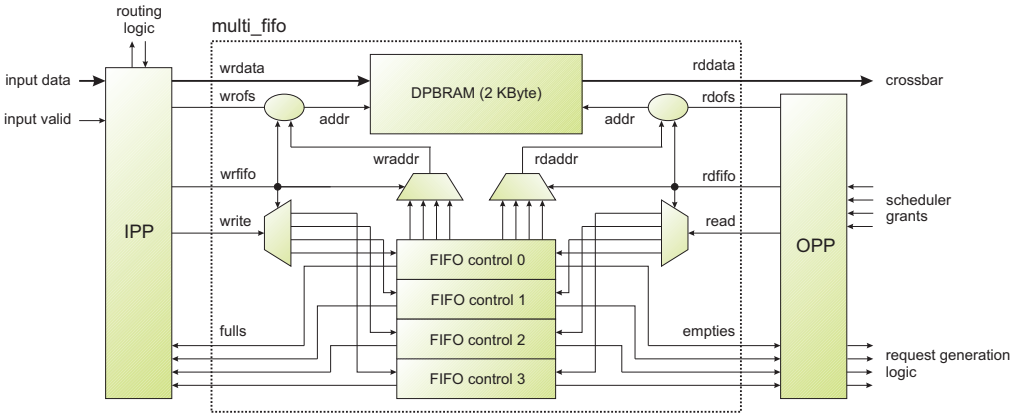



Figure 4.17: Input buffer implementation with VOQs for a switch with $N_b + N_g = 4$. Four independent FIFO queues are implemented within a single DPBRAM element.

however increase the queuing delay of the packets such that very large buffers are often contra productive. Section 5.7 discusses simulations concerning this topic.

*output packet processor*

Each queue containing a packet requests the central crossbar located in the core. Since requests are only raised for packets fully stored, this decouples the input-port timing from the timing of the core-input. The scheduler receives $(N_g + N_b)^2$ requests for the $N_g + N_b$ output ports available for packet transports. The requests are evaluated by the best-effort scheduler that returns select signals. The output packet processor (OPP) reads the packet from the queue and injects its data slots to the crossbar by observing the slot reservation and usage patterns. The request generation logic ensures that the current packet is transmitted completely by isolating the requests of its queue for that time.

### 4.5.4 Implementation of the Switch Core

*parts*

The switch core is the central element of the switch. It contains the request generation logic for the best-effort scheduler, the scheduler itself as well as the crossbar fabric that multiplexes connection data or packet data from the input ports to the output ports. The core is implemented within the VHDL module `core`. Figure 4.18 shows a block diagram of the core.

*operation*

The operation and the timing of the switch core is completely based on time slots. The core is not aware of best-effort packets nor of isochronous connections. Each time slot, the core transfers data slots in parallel from the cores input ports via the central crossbar to its output ports. The content of the data slots is not investigated. Each input ports transmits at most one data slot and each output port receives at most one data slot. There is no internal speedup. The decision of which ports to connect is based hierarchically in two steps:

*priority traffic*

In a first step, all port assignments to be made as its data slots belong to priority connections are calculated. These data slots are indicated by the reservation pattern stored within the static routing table for reserved connections as well as by the current slot usage according to the globally received data. This information is used

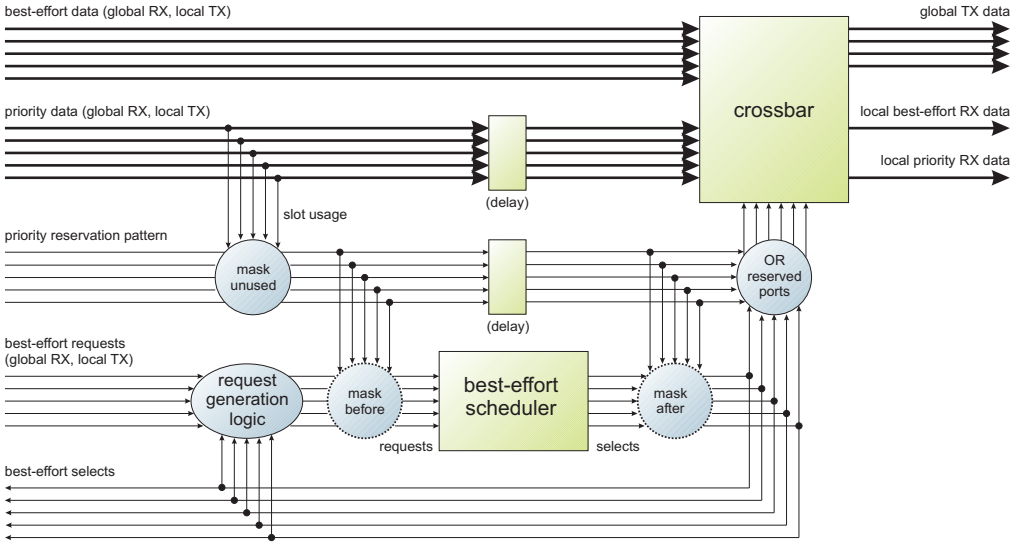



Figure 4.18: The central switch core. Example configuration for $N_g = 4, N_p = 1$ and $N_b = 1$. The timing and functionality is based on data slots. Ports currently belonging to reserved slots are masked out and are unavailable for best-effort transfers. This is done either before or after the scheduling process.

to control the crossbar as well as to denote the best-effort scheduler of the remaining slots available for best-effort packet transfers.

*best-effort traffic*

In a second step, the remaining (unreserved or unused) ports are assigned to best-effort transfers by the best-effort scheduler. For that reason, a request-generation logic ensures that input-output port combinations that currently transmit packets are continued until the packet is completed. The remaining requests are fed to the scheduler that calculates a bipartite matching between the available input ports and output ports. The scheduler implements a fair scheduling without static priorities. The priority assignment can be configured to be respected before or after the scheduling process by masking out the appropriate request or select signals. The resulting assignment is combined with the priority assignment and controls the crossbar.

*timing*

The feature to re-use reserved but unused priority data slots with best-effort data requires to investigate the data content of globally arriving data slots. The masking of the requests to the best-effort scheduler requires the priority data to be delayed for three additional cycles until the scheduler completes and the crossbar assignment has been calculated. Since this additional delay may be unacceptable, the switch features two configuration parameters (VHDL generics) for this case: the parameters `BE_USE_UNUSED_PRI` and `PRIORITY_LOW_DELAY` control the usage of reserved but unused priority slots with best-effort data slots. The first parameter controls the re-usage of the slots and the latter controls the activation of the delay elements. Without the additional delay, the switch cannot occupy reserved and unused priority slots with best-effort slots since this information is not available at the time of the scheduling process.

*space and time complexity*

The complexity of the central core is an important aspect and limits the complexity of the whole switch and thus the number of usable ports. This can be seen since the core implements all parts that cannot be implemented individually for each

input or output port, but require inter-port calculations or data transfers: the request generation logic, the scheduling of all requests and the transfer of the data slots between all input ports and output ports within the crossbar. The space complexity (the number of logic elements required) of these parts grows super-linear depending on the number of ports configured. The time complexity of the implemented logic is bounded by the internal data path. All data paths within the core operate with the rate of the MGTs, i.e. 16 bit at 156.25 MHz.

*critical path*

The most time-critical part of the core and thus the switch is the request generation logic whose resulting requests are fed into the best-effort scheduler. The scheduler has to calculate the resulting selects within the duration of a data slot of 2 clock cycles ($2 * 6.4$ ns). The number of ports significantly increases the scheduling complexity since the scheduler is required to implement a minimum of fairness between all ports. Concerning the reference implementation, the synthesized module of the scheduler is placed to a small area within the FPGA during the positive acknowledgment with retransmission (PAR) process to keep the signal delays of the internal logic small.

### 4.5.5 Implementation of the Central Crossbar

The crossbar is the central part of the core and thus of the whole switch. It performs the physical interconnects between the input ports and output ports. The data is transferred via the crossbar within a single cycle. Each port is 16 bit wide. Since the space complexity of the crossbar is of $O(N^2)$ with the number $N$ of ports of the switch, a space-efficient implementation is important. The crossbar is implemented within the VHDL module `crossbar`. The crossbar of the bypass-switch features separate inputs for reserved traffic and best-effort traffic. It is shown in section 5.7 that this significantly increases the performance for best-effort traffic without reducing the service guarantees for priority traffic.

*logic reduction*

Since local ports implement only its single traffic class, the crossbar does not have to interconnect all combinations of input ports and output ports. Typically, global ports are fully interconnected, whereas local ports are connected only to global ones to save logic. Possible interconnects are controlled with the VHDL parameter `connectivity`. Figure 4.19 shows the schematic of the crossbar module.

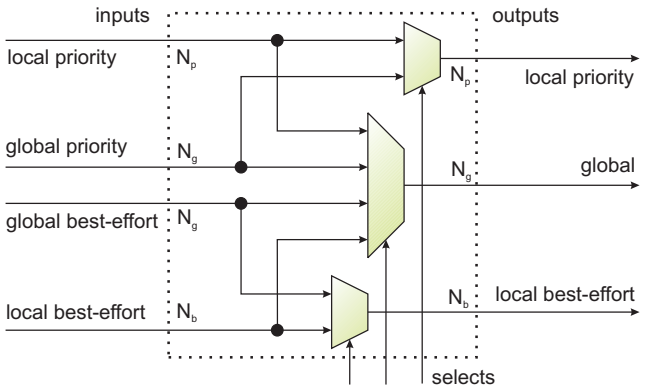



Figure 4.19: Schematic of the crossbar implementation with multiplexers at the output ports.

*implementation*

According to the multiplexer functionality of the crossbar, its operation can be separated for each output port, which equals a single multiplexer for all its possible input ports. Furthermore, all bits are multiplexed in parallel such that a compact implementation for a $n \times 1$ multiplexer has to be used.

### Implementation of the Multiplexer

*different techniques possible*

The multiplexers are the building blocks of the crossbar. The implementation has been made within the VHDL module `mux`. The used FPGA provides different design techniques to create multiplexers that consume different types of resources. Although the designer can let the synthesis tool to select the most effective technique, the module offers the parameter `mux_arch` to enforce a particular technique. The selection depends on the required speed and the amount of free resources of this type. The following techniques are provided by the programmable logic of the FPGA for the creation of a fully-connected crossbar that operates within a single clock cycle (for details see [154]):

- A hierarchic multiplexer consisting of LUT4 and MUXF5 to MUXF8 elements.
- The sum-of-product logic that calculates an OR over AND-ed input signals. The final OR can be implemented in a chain over multiple logic slices by using MUXCY and ORCY elements.
- The usage of tri-states.
- The (mis-)usage of the embedded multiplier blocks. Although this is somehow unusual, each multiplier can be programmed as a shift register to implement a $n \times 1$ multiplexer for $18/n$ bit in parallel. The speed of such a multiplier blocks is sufficient for this case.

*availability*

A synthesis tools usually chooses the first implementation, which consumes LUT4 logic elements. Since LUT4 elements are the basic logic elements of the FPGA and are required for most logic, it is worth to consider the usage of different techniques. Tri-states and multipliers are often unused and its usage allows to save LUT4 elements for other purposes. The embedded multiplier blocks allow for a compact implementation, but use the same routing resources as the DPBRAM elements, which has to be taken into account. Table 4.5 lists the consumed resources of its particular type for two example configurations of crossbars.

| type | control | $9 \times 1$ | | $4 \times 4$ | |
|---|---|---|---|---|---|
| multiplexers | integer | 80 LUT4 | 1 % | 128 LUT4 | 1 % |
| sum-of-products | one-hot | 144 + 32 LUT4 | 1 % | 256 + 64 LUT4 | 3 % |
| tri-states | one-hot | 144 BUFT | 5 % | 256 BUFT | 10 % |
| multipliers | one-hot | 8 MULT18 | 18 % | 16 MULT18 | 36 % |

Table 4.5: Logic consumption of different multiplexer implementations. The port width is 16 bit. The percentage values are calculated for the used `xc2vp7` FPGA type.

*selection*

The first multiplexer implementation requires an integer control signal, i.e. few bits that denote the number of the selected input. The other techniques require a one-hot encoded control signal. The optimal multiplexer implementation therefore

depends on the logic within the control path, e.g. if the AND of the sum-of-product logic can be absorbed within other LUT4 elements. For the implementation of the bypass-switch, the first technique results in the smallest design. A further reduction of the crossbar size can be reached by using multi-stage layouts like Clos networks [17], but this increases the transmission delay and is worth only for large port numbers.

### 4.5.6 Interface to the Best-Effort Scheduler

*purpose*

The purpose of the best-effort scheduler is to decide which of the non-reserved ports of the crossbar are used for packet-based transfers. For that reason, the scheduler calculates the bipartite matching between the two subsets of $n = N_g + N_b$ input ports and $n$ output ports used for best-effort packet transfers and not reserved for priority traffic in the next cycle. Since the request generation logic within the core module of the switch already performs the masking of available ports as well as the buffering of intermediate results, the implementation of the scheduler is reduced to the pure scheduling task without any reference to the slotted timing or the presence of priority.

*modular approach*

To allow a trade-off between logic consumption and performance, the type of scheduler used for synthesis can be selected to the one most practical for a given application. For this reason, the switch uses a generic interface for the scheduler module: Requesting ports are denoted as $n^2$ one-hot encoded signals, the selected ports are presented registered in multiple versions: as one-hot encoded array of $n^2$ signals over all ports, as a set of integer values that denote the selected input for each output and additionally, as an one-hot encoded array of the input and the output ports, respectively. This is useful since the signals that denote the scheduling result are heavily loaded and also time-critical. Due to its complexity, the implementation of two different scheduler types is described separately in section 4.6.

### 4.5.7 Interface to Upper Network Layers

This section describes the interfaces at the local ports of the switch to upper network layers. The bypass-switch features two different interface types: for connection-based transports and for packet-based transports. The interfaces control the data flow between an application and the switch, i.e. the transmission of data into the network and the reception of data from the network. Both types of local ports feature the same data rate of 16 bit at 156.25 MHz. The interfaces are described briefly in the following.

#### Handshake Signals for Isochronous Connections

*timing*

The bypass-switch has $N_p$ local ports to connection-based priority traffic. The interface comprises the receive data path and the transmit data path for each port as well as separate timing information for both directions (cf. section 3.4.6). The timing information provided by the switch denotes the start and the end of frames as well as the slot positions. The timing between both directions is shifted by two cycles. This is since the data paths of both port directions are directly connected to

the crossbar and since the timing of the transmit path is provided a cycle earlier (cf. figure 4.16). Figure 4.20 shows the timing diagram of the interface.

*handshake signals*

The transmission of data is controlled with two handshake signals in both directions: data `accept` and data `valid`. The data is transferred when both, `accept` and `valid` are set to logic 1. At the transmission interface, the switch asserts the `accept` signal a cycle *earlier* such that the application can register the data to be sent, which is sampled in the succeeding cycle. To receive data, the switch denotes valid data by asserting the `valid` signal. The data has to be sampled by the application in the *same* cycle. Since the switch does not implement buffers, an application cannot delay the reception. In fact, the application-side `accept` signal is not evaluated by the switch.

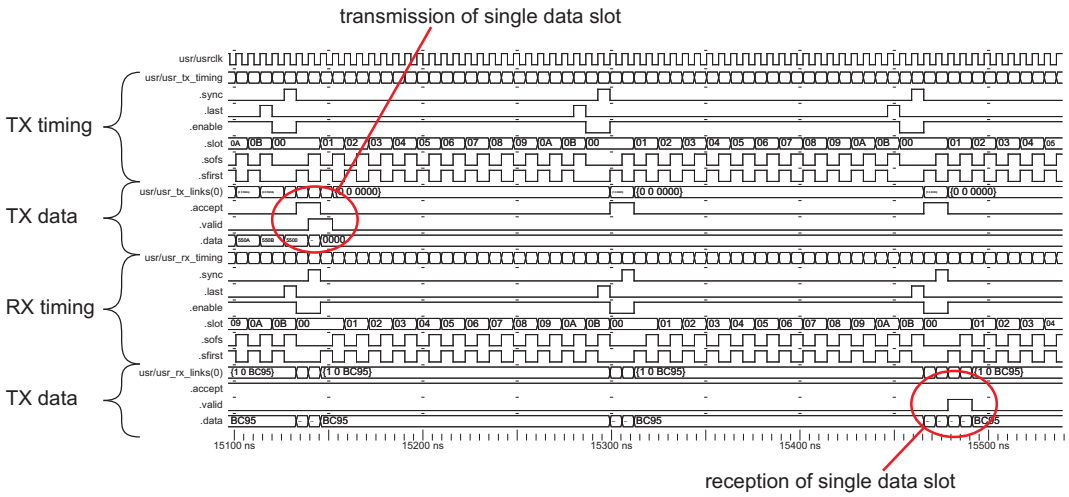



Figure 4.20: Timing diagram of the switch interface of the transmit data path and the receive data path for connections-oriented traffic. The handshake signals are shown for the transmission and the reception of a single slot. The timing information is provided separately for each direction. The framing parameters are $f = 12$ time slots and $S = G = 2$ cycles per slot.

### Handshake Signals for Best-Effort Packets

The transmission and the reception of packet-based data is handled by a separate PAL for each protocol type (cf. section 3.8.4). Since the timing of the switch is based on time-slots, the PAL transforms packets into multiple slots and vice versa. The reference implementation of this chapter uses only shared memory packets as the single packet-based protocol. The interface to the user therefore merely concerns the transmission and the reception of shared memory packets, which is described in section 4.9.3. A description of the interface between the PAL and the switch is therefore omitted.

## 4.6 Implementation of the Best-Effort Scheduler

*example implementation*

The purpose of the best-effort scheduler is to decide which of the non-reserved ports of

the crossbar are used for packet-based transfers. This decision is made each time slot according to the slotted data path of the switch. The MCGN specification does not request a dedicated type of scheduler to be used. The implementation of the switch therefore uses a modular design in which the optimal scheduler for the application can be selected for synthesis (cf. section 4.5.6). This section presents two example implementations of best-effort schedulers. Both schedulers can be used for the packet scheduling within the transport network. The type of scheduler to be chosen depends on the required performance and on the available amount of programmable logic.

*task*

During the operation of the switch, the best-effort scheduler takes a set of $n \times n$ `requests` signals. Each signal corresponds to a particular input/output port combination. A signal that is set to logic 1 denotes a request for the transfer of data slots of packets between the corresponding input/output pair. The task of the scheduler is to calculate the bipartite matching between the ports, i.e. to `select` or to `grant` a particular set of input/output pairs such that each port is assigned to only at most one other (cf. section 1.6). The number of selected pairs has to be maximized. As a second requirement, the scheduler should be fair and not prefer or discriminate a certain port pair. The complexity of the scheduling process is limited by the duration of a time slot of 2 clock cycles (12.8 ns).

*implemented scheduler types*

Section 1.6 discussed the issues and possible solutions for the calculation of such a bipartite matching. For the implementation of the transport network, the following schedulers have been investigated: static priority, TDM, MSM [56], iSLIP with and without multiple iterations [84, 86, 85, 48], the simple 2-dimensional ripple-carry arbiter [59], RPA [59] and DPA [59]. All schedulers have bee implemented within VHDL and have been verified in software (cf. section 5.7). Although MSM has very good results, it has only been implemented in software due to its high complexity of $O(N^{2.5})$.

The following sections exemplarily describe the implementation of iSLIP and the 2-dimensional schedulers including DPA. iSlip and DPA both are fair multi-port schedulers with medium complexity and good performance.

### 4.6.1 Implementation of the iSLIP Scheduler

The operation of the iSLIP scheduler has already been described in section 1.6.2. A more detailed functional descriptions can be found in [84, 86, 85, 48]. The scheduler is implemented within the VHDL module `scheduler_multi`. A schematic of the implemented scheduler is shown in figure 4.21.

*operation*

The $n \times n$-port iSLIP scheduler consists of $n$ *grant* arbiters each belonging to an output port and $n$ *accept* arbiters each belonging to an input port. Grant arbiters and accept arbiters are itself $n$-port round-robin arbiters (cf. section 1.6.1). The overall operation is controlled by a simple state-machine (not shown in the figure). The implemented scheduler operates in two steps:

1. Each grant arbiter takes $n$ request signals denoting the requests from the inputs to that output. The grant arbiter selects (grants) a single input according to its current internal priority.
2. Each accept arbiter takes $n$ grant signals denoting the grants from the outputs to that input. The accept arbiter selects (accepts) a single output according to

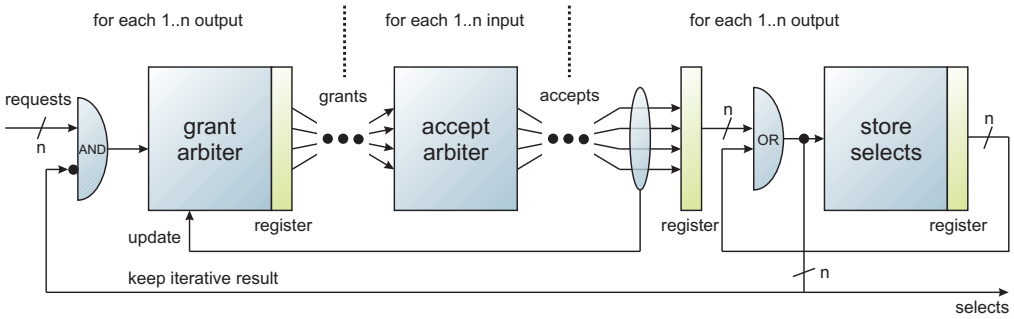



Figure 4.21: Schematic of the implemented iSLIP scheduler. The grant and accept arbiters are instantiated for each input or output port. A state-machine (not shown) triggers the operation of the arbiters and their update process. The iSLIP scheduler calculates a bipartite matching for all $n \times n$ input/output port pairs in parallel within 2 clock cycles. Multiple iterations require additional cycles. Calculated selects are stored temporarily during the iterations.

its current internal priority. The second cycle also contains the update process of the grant arbiter according to the accepts to cope to the de-synchronization updating rule of iSLIP.

The original definition of iSLIP contains three steps, the first step (`request`) is basically the forwarding of the request signals to the grant arbiters.

*time and space complexity*

The implemented scheduler requires two clock cycles for a single iteration. The register stages for the logic signals are denoted by the *register* markings within figure 4.21. The figure also shows that the main amount of logic is consumed by the $2n$ round-robin arbiters that calculate the grant and accept signals. The further logic basically consists of single `AND` and `OR` gates. This results in the fact that the time complexity (critical path) as well as the space complexity (logic consumption) is both determined by the complexity of a single $n$-port round-robin arbiter. For that reason, a fast and space-efficient round-robin arbiter has been developed, which best fits to the FPGA used (see below).

*multi-iterative support*

The scheduler supports multiple iterations for a better scheduling result. It therefore stores the `selects`-signals of intermediate iterations. The total scheduling process for $i$ iterations requires $2i$ cycles. Although the scheduling time is limited to the 2 cycles of the time slot $S$, multiple iterations can be used with other applications that allow larger time slots and thus an increased scheduling time.

### The Tiny Tree Arbiter

*round-robin implementation*

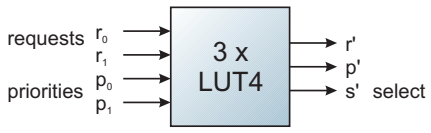
This section presents an efficient implementation of a round-robin arbiter to be used for the iSLIP scheduler. A round-robin arbiter can be implemented using a programmable priority encoder (PPE), whose top-priority input is changed according to the scheduling result. A common implementation for a PPE uses two static-priority arbiters together with a thermometer-encoded mask [48]. In this thesis, a different implementation of a PPE has been developed, the tiny tree arbiter (TTA). It exploits the LUT4 architecture of the given FPGA [154] in a very compact, efficient and scalable design. A comparable result has been independently published in [162].

*programmable priority encoder*

A PPE takes $n$ request signals together with $n$ priority signals as input. The requests can be set arbitrarily, whereas the priorities are one-hot encoded denoting the input with the currently highest priority. The priority of the other inputs descends with increasing input positions and wraps around from the last input to the first. The output of the PPE can be the set of $\log_2 n$ signals as the binary representation of the selected input number or a one-hot encoded set of $n$ *select* signals.

*TTA element*

The PPE created for the TTA arbiter arbiter consists of small building blocks, the TTA elements. Each element has four inputs and combines the requests and priorities of two inputs and calculates a single request, priority and select signal. The logic table of the TTA element is such that multiple elements can be stacked to a large tree without further logic in between. Each element is very tiny and requires only three LUT4 elements of the given FPGA type. Figure 4.22 shows a block diagram of the TTA element and its truth table.




| $p_0$ | $p_1$ | $r_0$ | $r_1$ | $p'$ | $r'$ | $s'$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 * | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |

Figure 4.22: (left) Schematic of the developed TTA element. (right) truth table of the TTA element. The hierarchical design allows to stack multiple elements to create a large scheduling tree without further logic in between. Since each element is very tiny, a round-robin arbiter based on a TTA is fast and compact.

*TTA tree calculation*

The scheduling decision is calculated as follows: The priority output $p'$ is the OR-ed value of both $p$-inputs such that the priority property of the prioritized input is percolating to the top-most element. The request $r_0$ has priority over the request $r_1$ in the case that neither has explicit priority. In the case of $p_0 = 1$, the request $r_1$ has second highest priority, whereas in the case of $p_1 = 1$, $r_0$ has the *least* priority out of *all* inputs. A request $r'$ is forwarded upwards in the case that any or both of the two inputs are requesting. The select output $s'$ denotes which input to select in the case that one of the two inputs of the element is finally selected. The situation for the case marked with an asterisk in the table of figure 4.22 is special, since the element has priority but the least prioritized input out of all inputs is solely requesting. Forwarding the request would select this element (and thus $r_0$) independently from all other input requests due to its priority. By using the encoding of the table, the element is selected only in the case that *no* other input is requesting, which corresponds to its least priority. If no request is set at all, the currently top-priority input is selected.

*calculation of selects*

The selected input is calculated only out of the $s'$ outputs of the tree elements. Its is denoted by the binary representations of the values of the intermediate $s'$ outputs, starting from the top-most element and going back the tree according to each intermediate $s'$ value. The output of the binary select representation requires $\log_2 n$ calculations with increasing complexity (the most significant bit equals the $s'$ output of the three, whereas the least significant bit has to be calculated out of all $s'$ outputs). The one-hot encoded selects are calculated independently for each output. The calculation requires only $\log_2 n$ inputs of intermediate $s'$ to be considered for each select output.

*creating a TTA round robin arbiter*

A round-robin arbiter can be build easily around the TTA by registering the one-hot representation of the selects and feeding the correct next priority to the TTA input. Since both, the selects and the priorities are one-hot encoded, the priorities equal the shifted selects output of the schedule and can thus simply be feed back without further logic. The shift ensures that the element currently selected gets the least priority next. Figure 4.23 shows an example implementation of a TTA-based round-robin arbiter with $n = 8$[3].

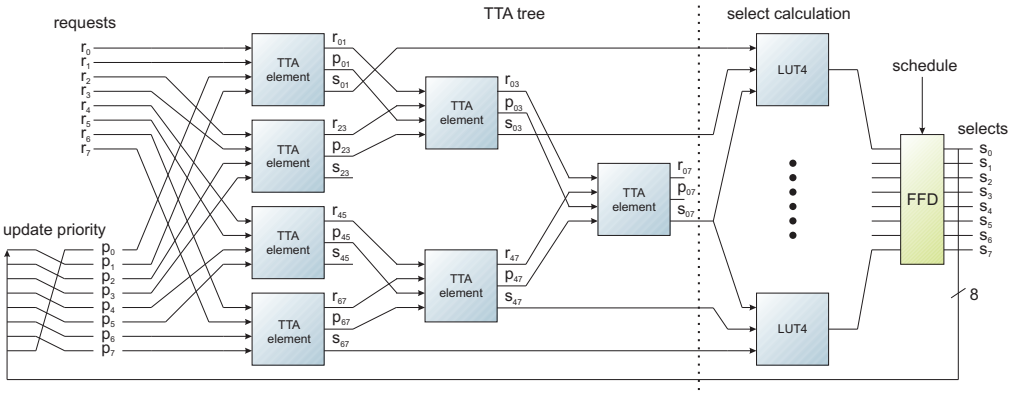



Figure 4.23: Example of a 8-port round-robin arbiter based on the TTA tree with one-hot encoded select outputs. The whole arbiter requires only 27 LUT4 elements within 4 stages of logic and 8 final select flip flops. Only two of the eight select calculation LUT4s are shown.

*verification*

Although the operation of the TTA element is easy to understand, the functionality has been tested with different tree sizes by systematically verifying the selects for all possible input request and priority patterns. However, a theoretical proof of its correct operation is not given here for readability. This can easily be done using induction starting from a single TTA element.

*space and time complexity*

The space and time complexity of a TTA arbiter clearly depends on the number of tree elements. A tree of input-size $n = 2^d, d \in \mathbb{N}$ requires $d$ stages of elements. The first stage has $n/2$ elements and the succeeding stages half the number with every stage. The total tree has thus

$$e_n = \sum_{k=0}^{d-1} 2^k = \frac{1-2^d}{1-2} = 2^d - 1 = n - 1 \tag{4.4}$$

[3]Extra registers to store the internal priority bits are required to control the priority update process like it is required for the iSLIP scheduler.

TTA elements. Each element requires three LUT4 elements except the last, which requires only a single one (its outputs $p'$ and $r'$ are unused). Additional logic is required for the calculation of the select outputs. For $n \leq 16$, the one-hot select calculation requires only a single LUT4 for each select output. The total number of LUT4s therefore calculates to:

$$s_n = 3 \cdot e_n - 2 + n = 4n - 5. \tag{4.5}$$

The space complexity of an TTA round-robin arbiter with $n$ inputs is therefore $O(n)$ only.

The time complexity is given by the number of LUT4 elements within the critical path from the request and priority input to the select outputs. It is clear from the design that the critical path includes $d + 1 = \log_2 n + 1$ LUT4s, resulting in a time complexity of the arbiter of only $O(log(n))$.

*larger sizes*

For sizes of $n > 16$, the linearity in the number of LUTs depends on the number of LUTs required to calculate the select output, i.e. to compare the $d$ intermediate $s'$ signals of the tree with the binary representation of the position of the given select output. For this sizes, the final amount of logic required depends on the synthesis tool. As an example, a round-robin arbiter with $n = 64$ can by synthesized in a total of 267 LUT4, thus 80 LUT4 are needed to calculate the 64 select outputs.[4]

*arbitrary port numbers*

The size of the TTA tree and thus the arbiter is not limited to input numbers $n$ of $n = 2^d$. Different input numbers can be used easily by first taking the smallest tree of size $n' > n$ that holds $n' = 2^d$ and simply disabling unused requests and priorities, which will effectively reduce the final logic consumption. The priority assignment for the round-robin arbiter has to shift the select outputs and wrap at the input $n - 1$. Since only requested inputs are selected and since the top-priority input is selected in the case of no requests, only valid outputs in the range $0..n - 1$ are still selected. This has also been verified by simulating all possible input patterns for different sizes of $n$.

*future work*

It has finally to be stated that the TTA design can also be extended to upcoming FPGA technologies based on a 6-input LUT logic [153]. Each element then takes 3 request and priority inputs and has two bits for the $s'$ signal, thus requiring a total of four LUT6 elements.

### 4.6.2 Implementation of Two-Dimensional Schedulers

The second type of scheduler that has been exemplarily implemented as the best-effort packet scheduler for the bypass-switch is based on a 2-dimensional arrangement of small and fast arbiter cells [141, 59].

The scheduling task of selecting input/output port pairs is implemented by assigning each arbiter cell to a particular port pair. The selection of port pairs is thus transformed to the "selection" of cells. During the scheduling process, each cell whose port pair requests the scheduler gets a dedicated request signal. The requests are forwarded - or rippled - through the 2-dimensional arrangement of the cells until

[4]The results have been achieved by using the synthesis tools Xilinx XST 6.3.3 [158] and MAP 6.3.3 [158], which have been optimized for small area and targeted to the Virtex-II Pro [154] technology of the FPGA of the framework

a scheduling has been found. To avoid multiple selections of a particular port, each cell forwards is selection status to adjacent cells.

*three different types*

Three different schedulers of this kind are implemented, namely the simple 2-dimensional ripple-carry arbiter, RPA and DPA [59]. The schedulers differ in the arrangement of the basic arbiter cells and thus have different complexity and scheduling results. The latter is the only scheduler that ensures fairness, has a good scheduling result and an acceptable space and time complexity for the usage as the best-effort scheduler of the transport network. All schedulers are implemented within the VHDL module `scheduler_multi_2d`. The implementation is described briefly in the following.

### The Simple 2-Dimensional Ripple-Carry Arbiter

*arbiter cell*

The arbiter cell of the two-dimensional ripple-carry arbiter consists of two inputs (denoted as `north` and `west`), two outputs (`south`, `east`), a request and a grant signal for its corresponding port. The `north` and `west` inputs denote a present arbitrations for inputs or outputs with higher priority, whereas the cell itself ripples the arbitration status to cells with lower priority via its `south` and `east` outputs. Figure 4.24 shows a schematic and the corresponding logic of the cell.

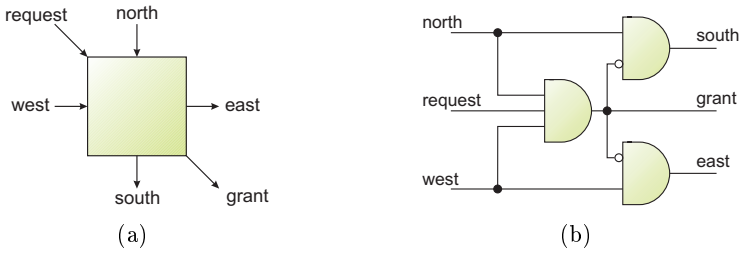



Figure 4.24: Arbiter cell of the 2-dimensional ripple carry arbiter. (a) Schematic (b) Logic. (after [59])

*arrangement*

Figure 4.25 shows the cell arrangement in a $N \times N$ matrix, where $N$ is the number of input and output ports. The rows correspond to the inputs and the columns correspond to the outputs. The lines between the cells denote the forwarding of the requests and *not* to the data of the ports. For the highest row and the left-most column, the `north` and `west` inputs are set to logic 1, respectively. The arbiter matrix ripples requests from the top-left cell (corresponding to the input/output pair with the highest priority) to the bottom-right cell (with lowest priority).

*complexity and performance*

The two-dimensional ripple-carry arbiter can be implemented very efficiently and performs well. Its space complexity is clearly $O(N^2)$. The time complexity corresponds to the time to ripple the requests from the top-left cell to the bottom-right and is thus $O(2N-1) = O(N)$. Its main drawback is the unfairness due to its static priority that prefers the input/output pair assigned to the top-left cell.

### Improvements of the 2-Dimensional Arbiter

*introduction of fairness*

To ensure fairness, the top-priority cell has to be moved between successive arbitrations such that each cell has the opportunity to get the highest priority. The work

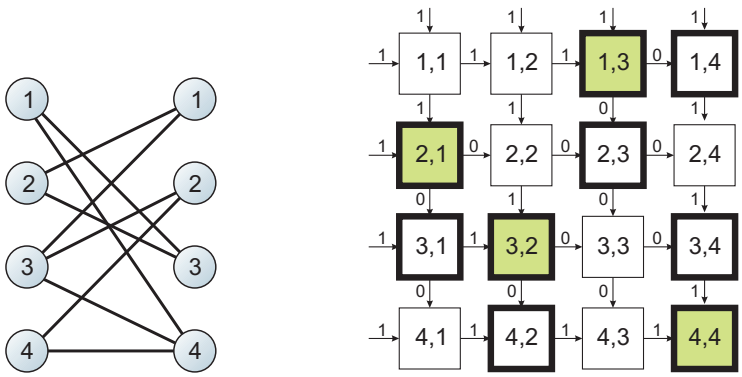



Figure 4.25: Schematic of the simple 2-dimensional ripple-carry arbiter of 4 ports. (a) The corresponding requests from input ports (left) to output ports (right). (b) The cell arrangement of the arbiter. Cells of requesting input/output pairs have thick borders, cells of granted pairs are shaded.

in [141] proposed an extension of the 2-dimensional matrix to a close toroid by using a cyclic feedback of the outputs from right and bottom to the inputs at the left and top, respectively. Since this requires a logic loop for a single-cycle operation, the work in [59] changes the arrangement of the cells to introduce fairness and also allows a single-cycle operation without logical feedback loops. The resulting designs are the rectilinear propagation arbiter (RPA) and the diagonal propagation arbiter (DPA). Both design are exemplary shown in figure 4.26 for a 4-port scheduler.

*duplicated mask-able logic*

The improved designs solve the feedback problem by a duplication of logic beyond the boundaries of the initial design. Furthermore, the single arbiter cell is extended by a mask input, which is used to enable and disable the cell. Enabled cells operate as described, whereas disabled cells never grant and forward a logic 1 to the `east` and `south` outputs to give priority to the succeeding cells at the right and bottom side. During the arbiter operation, $N^2$ cells are enabled. Fairness is introduced by changing the top-priority cell, which is done by cycling the group of currently enabled cells.

*rectilinear propagation arbiter*

The RPA arbiter uses a duplication of logic beyond the right and bottom side of the initial ripple-carry layout. The subset of enabled cells consists of a squared region of $N^2$ cells, i.e. all $N^2$ cells can have the top-priority. RPA calculates the same scheduling result as the initial design with cyclic feedbacks. As a drawback, it requires $(2N-1)^2$ cells and thus suffers from an increased amount of logic. Concerning the propagation delay, the worst-case logic path is extended to the delay of $4N-3$ cells. By considering that only the delay of the functional part within the enabled square is of interest, this can further be optimized by denoting the logic synthesis tool of the required partial transmission delays between groups of cells.

*diagonal propagation arbiter*

The DPA arbiter arranges each $N$ independent arbiters (which have no input and no output in common) in diagonal rows. The area of enabled cells comprises $N$ diagonals. A cycling of the priority is done by a vertical shift of the area that contains the enabled cells. Consequently, the DPA arbiter has $N$ different priority states in contrast to the RPA arbiter. Although the DPA arbiter calculates a different

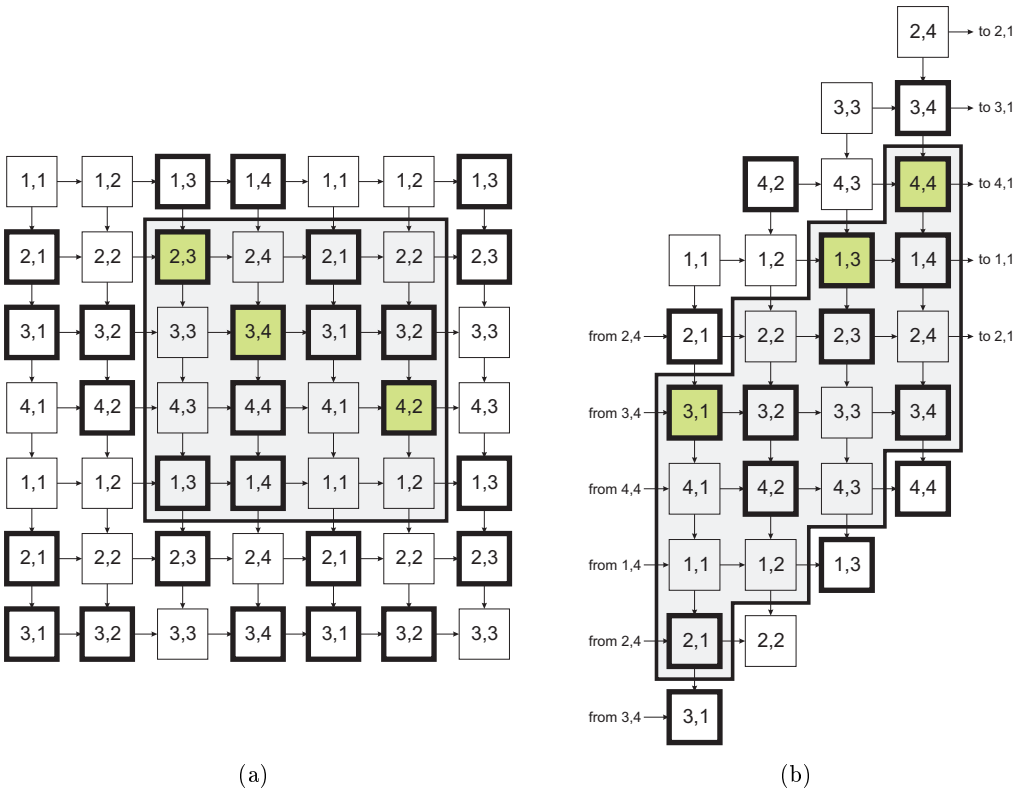



Figure 4.26: RPA and DPA arbiters for a 4-port scheduler. (a) RPA arbiter with highest-priority cell at (2.3) (b) DPA arbiter with highest priority diagonal rooted at (3,1) (after [59])

arbitration, its performance is comparable to the RPA arbiter [59]. As the advantage, its space complexity is reduced to $(2N-1)N$ arbiter cells. The propagation delay is also reduced to about $N$ cell-delays. This can be seen comparably to the RPA arbiter, when it is assumed that only the delay within the enabled subset of cell arbiters is of interest. The DPA arbiter is therefore more space-efficient as well as faster as the RPA arbiter.

### 4.6.3 Summary

*implementation*

Two scheduler architectures have been presented: the iSLIP scheduler and a set of 2-dimensional schedulers. All schedulers have been implemented as described. Simulation results of the performance of the schedulers are given in section 5.7. The configurable design allows the user to choose the appropriate scheduler according to the required performance and the complexity of the resulting digital design:

- The iSLIP scheduler is fair and relatively compact. It requires $2N$ round-robin arbiters, which can be implemented space-effcient using the TTA design. Its drawback is the medium performance for single iterations. Multiple iterations would require to increase the slot size to more than two cycles.

- The simple 2-dimensional ripple carry arbiter has a compact design and a good performance. Its drawback is the missing fairness, which might starve out the least-priority port-pair completely during operation.
- The RPA and the DPA designs are fair and perform well. The space complexity of the DPA arbiter is much smaller.

The type of scheduler to be synthesized can be selected within the VHDL description of the switch core. Since the limited amount of programmable logic is a main design issue, the iSLIP scheduler or the DPA scheduler are suggested to be selected in most cases.

## 4.7 Routing of Best-Effort Packets

*general definition*

Routing is the task of selecting the global route for a packet between its source node and its destinations node in the case that both nodes are not directly connected. The knowledge of the selected route is required at each node to forward a packet to the correct output port of the switch. The routing algorithm can be performed at the time the packet is transmitted into the network, statically during the network initialization or dynamically and independently at each network node.

*MCGN routing*

The specification of MCGN does not denote a particular routing algorithm for best-effort packets. The routing decision can be performed at two different stages: at the reception of the packet within the data link layer or within upper network layers to support a more general routing algorithm (cf. section 3.8.5). The latter variant requires to move all incoming packets locally to upper layers before the forwarding, which introduces a significant delay. This method has not yet been implemented. The first method has been implemented and is discussed in the following.

*routing function*

According to section 3.8.5, the routing process can be defined as a function or algorithm $\Omega$, that assigns the local output port depending on its arguments like the position of the destination node, the position of the local node, the protocol type of the packet or certain network status information. The function is called at each network node for every packet that enters the switch at a global port as well as for all packets that are transmitted at a local port into the network.

*implementation constraints*

The topology of the backplane, the implementation of the bypass-switch and the DSM constrain the implementation of such a routing algorithm in the following ways:

- The regular topology of the network allows to implement a simple routing algorithm.
- The implementation of the bypass-switch requires a routing decision to be made within a single cycle at the time a best-effort packet arrives at the switch. The header of the packet provides all necessary information within the first cycle of the packet.
- The simple stop-and-wait automatic repeat request (ARQ) protocol of the DSM implementation requires a routing algorithm that keeps the packet order to perform best (cf. section 4.9.6).

### 4.7.1 Description of the Implemented Algorithm

*online algorithm*

The implemented routing algorithm calculates the correct output port for an incoming packet according to the content of the packet header. It performs a routing decision within a single cycle to forward the packet after the first data cycle has been received. In the case a packet arrives at a global input port, the result of the function is used to determine the correct place within the input queue - which is in fact a virtual output queue and thus sorts the packet according to its output. In the case the packet is transmitted locally, the queue may be bypassed and the corresponding output port is requested directly (cf. figure 3.34 and figure 4.16). The routing algorithm is implemented within the VHDL module `routing_dynamic` as a VHDL function.

*inter-backplane routing support*

The implemented algorithm performs two main tasks: the routing within a single backplane and the routing between multiple backplanes. A routing decision is made only according to the position of the local network node and the position of the destination network node. The position of the local network node is known since all network nodes are initialized via the SlowControl at the very first start of the framework operation.

*cubic coordinate system*

The implemented routing algorithm exploits the regular arrangement of the 16 `Nathan` modules within the backplane. Figure 2.8 shows that the network topology can be illustrated as a 2-dimensional toroidal structure or a 4-dimensional binary cube. The routing algorithm uses the cubic representation of the topology to perform dimensional routing [100]. The position of each network node can be represented by a 4-dimensional coordinate, which is simply a tuple of 4 bit being each 0 or 1 and denoting the position within its dedicated dimension. The topology of the network allows to change (invert) the actual position of the packet within a dimension simply by forwarding the packet to one of the four output ports that correspond to the hardwired physical backplane links. Table 4.6 lists the cubic backplane-coordinate of each network node. The coordinates belong to the MGT instances (`x1y0`, `x2y0`, `x1y1`, `x2y1`) of the used `xc2vp7` FPGA type.

| network node | cubic coordinate | network node | cubic coordinate |
|---|---|---|---|
| 0 | (0,0,0,0) | 8 | (0,1,0,1) |
| 1 | (1,0,0,0) | 9 | (1,1,0,1) |
| 2 | (1,0,1,0) | 10 | (1,1,1,1) |
| 3 | (0,0,1,0) | 11 | (0,1,1,1) |
| 4 | (0,0,1,1) | 12 | (0,1,1,0) |
| 5 | (1,0,1,1) | 13 | (1,1,1,0) |
| 6 | (1,0,0,1) | 14 | (1,1,0,0) |
| 7 | (0,0,0,1) | 15 | (0,1,0,0) |

Table 4.6: List of the backplane coordinates of the `Nathan` network modules according to the cubic coordinate system of figure 2.8 used for the routing of best-effort packets. The change of a certain coordinate requires to use the same hardwired MGT link on all network modules.

*online routing process*

To perform the routing task, the routing algorithm first converts the logical

position of its network node as well as the position of the destination both to its cubic coordinates. Each bit that differs between the two coordinates corresponds to a coordinate that has to be changed to reach the destination. This process is simplified by the fact that a certain coordinate corresponds to the same MGT on each network module, which allows to use the same algorithm on each module without further configurations. The number of differing bits equals the number of hops to be passed.

*in order delivery*

The routing algorithm could in principle select any of the differing dimensions to change next, i.e. to forward the current packet to one of the appropriate output ports. This would also allow to adapt to the amount of packets that are currently stored in the various VOQs and thus to support a little congestion control. This feature is currently not used since it may lead to out-of-order delivery of the packets, which would reduce the performance of the implemented stop-and-wait algorithm within the transport control logic of the DSM (cf. section 4.9.6). The routing algorithm therefore selects the dimensions in ascending order.

*interconnection of backplanes*

To interconnect multiple backplanes, the routing algorithm supports the usage of the four additional MGTs of each FPGA (cf. section 2.2.3). The backplanes are switched by using dedicated gateway nodes. In the case the local backplane number and the backplane number of the destination node differs, the routing algorithm first forwards the packet to the gateway node using the above process. The gateway node then uses one of the extra output ports to reach the destination backplane, where the packet is forwarded to its destination node as described above.

### 4.7.2 Summary

*advantages*

The implemented routing algorithm transforms the linear coordinates of the network modules into a cubic coordinate system to simplify the routing task to dimensional routing. It has the following advantages:

1. It is fast and simple to implement and thus consumes only few logic.
2. It keeps the packet order.
3. The dimensional routing avoids live-locks and dead-locks. This means that the packet is ensured to reach its destination without circling or being blocked.
4. It allows to support adaptive routing in the case the implemented ARQ algorithm handles out-of-order packets with only minor performance reduction.
5. The same logic can be implemented within all network nodes. No further configuration or routing table definitions are required.

*future development*

Future development may use an additional link out of the four arbitrary usable links at *each* network node to increase the coordinate by a fifth dimension instead of using a gateway node. This indeed would lead to better performance, but however, this would also increase the required amount of logic within each network node. Since the routing algorithm is implemented within a single file, it is easy to exchange the algorithm in the case that other, even irregular topologies, are used in the future.

## 4.8 Transport of Neural Network Data

*purpose*

The last sections presented the reference implementation of the MCGN switching architecture for the lower network layers of the transport network. The purpose of the upper layers is to use the isochronous connections provided by MCGN for the delivery of neural network data between the ANN chips `HAGEN` and `Spikey`, which are distributed on several `Nathan` network modules (cf. figure 2.13). This requires to connect the interface of MCGN for isochronous connections to the ANN controllers of the two chips. However, it has already been stated that the development of the network interface of the controllers is not complete.

*demonstration of features*

To demonstrate the feasibility of the implemented transport network for the transmission of neural data, the following section first summarizes the features of the isochronous transport service provided by the network with respect to the neural data of the chips. The succeeding section describes a demonstrator application for the usage of the isochronous connections. The demonstrator application has been implemented within programmable logic and can be used as a reference for the further development of the ANN controllers.

### 4.8.1 Provided Transport Service for Neural Network Data

*bandwidth requirement*

The transport requirements of the two ANN chips `HAGEN` and `Spikey` have been described in section 2.4.2. The tables 2.4 and 2.5 list the estimations for the amount of neural network data to be transported within isochronous connections during the experiments. The `HAGEN` chip requires a constant data rate.

*unknown spike event rates*

In contrast to that, the mean and peak frequencies of integrate-and-fire neurons within a neural network may differ drastically (see e.g. [15]). Consequently, the `Spikey` chip produces a statistical event rate, which is influenced by the following factors:

1. The topology of the investigated neural network. This results in certain inter-chip connections.
2. The timing parameters (speedup) of the synapses and neuron circuits of the ANN chip according to the implemented neuron model.
3. The strength of the external excitatory network stimulation as well as the internal excitatory feedback.
4. Statistical fluctuations in the data rate caused by physical properties such as the temperature of the chip.

These topics depend on the particular experiment. All topics may be fixed or changeable according to the intention of the experimenter. The resulting spike frequencies can hardly be predicted for a given configuration. To evaluate the performance of the transport network, the following paragraph calculates possible physical spike frequencies (event rates) according to the throughput provided by the network.

#### Characterization of the Connection-Based Transport Service

*provided features*

The implementation of the transport service provides the following features for the transport of data within isochronous connections:

- A configurable number $N_p$ of local ports. The ports are bi-directional with independent interfaces for transmit and receive. The interface is 16 bit at 156.25 MHz and provides a usable bandwidth of up to 312.5 MByte/s (minus the waste by the frame gap).
- A transport within multiples of data slots of 2 clock cycles (32 bit).
- A delivery with guarantees QoS. The data rate is limited according to the number of slots, the throughput is guaranteed. The mean delay mainly depends on the number of network hops, the jitter depends on the slot order within the frames (cf. section 3.7).
- The possible number of connections per node is limited by the number of time slots per reservation period (the framing parameter $m$, cf. section 3.6).

*grouping of neurons*

To improve the performance, all inter-neuron connections between the same two end-points can be aggregated within a single inter-node connection. The has two advantages: First, it reduces the amount of connections and thus the jitter due to the lower amount of required slots per reservation period. Second, it can be assumed that the grouping of neurons improves the bandwidth usage of the `Spikey` chip. This is since the events are generated statistically and the combination of statistical data rates results in a closer Gaussian deviation of the total data rate (*law of the large numbers*).

### Guaranteed Throughput

*throughput dependencies*

The most important value is the guaranteed throughput of the network. This depends on the following factors:

- The topology of the hardware framework. The backplane topology can be extended by using the four additional MGTs connects of each FPGA. The physical bandwidth $w$ equals on all links.
- The framing parameters of the MCGN data frame, e.g. the number $f$ of time slots per frame. This determines the fraction of usable bandwidth $w_f$.
- The fraction of bandwidth to be reserved for isochronous connections. Although reserved data slots may be used for best-effort traffic, it is possible to limit the number of reserved slots to improve the shared memory performance.
- The efficiency of the connection mapping algorithm. The mapping algorithm succeeds or fails according to the bandwidth requests and the selected framing parameters and thus determines its possible values.

To give some numerical values, the framing parameter $f$ has to be fixed. The available bandwidth $w_f$ can then be calculated by equation 3.1 and equation 3.2. A typical value for the framing parameter $f$ is 60 data slots per frame, since it is dividable in different reservation periods and also results in a small waste by the frame gap. The available bandwidth $w_c$ per inter-chip connection $c$ depends on its number $m_c$ of reserved slots and calculates to $w_c = w_s \cdot m_c$ (cf. equation 3.4).

*calculation of spike rates*

**Example Calculation for the Spikey Chip** Concerning the `Spikey` chip, a spike event is encoded into a single data slot of 4 byte. The maximum rate $\nu_f$ of spike events to be transported on each physical link is thus:

$$\nu_f = \frac{w_f}{4 \text{ byte}}. \tag{4.6}$$

By using $f = 60$, $w = 312.5$ MByte/s and $S = G = 2$ cycles $= 12.8$ ns, the maximum event rate per physical link $\nu_{60}$ results to:

$$\nu_{60} = 76.84 \text{ MHz}. \tag{4.7}$$

This leads to the maximum *number* of neurons, whose spike events can be transported per physical link depending on the actual average spike frequency $\overline{\nu}$ of the single neurons. Table 4.7 lists typical values for the neuron numbers according to different mean spike frequencies in the case that the mapping algorithm succeeds to occupy 100 % of the slots. The table also lists the possible number of neurons per slot in the case that 5 reservation periods of $m = 12$ slots per period are configured.

| mean physical event rate $\overline{\nu}$ | number of neurons per physical link | number of neurons per slot of reserv. period, $m = 12$ |
|---|---|---|
| $1 \cdot 10^5$ Hz | 768.4 | 64.0 |
| $2 \cdot 10^5$ Hz | 384.2 | 32.0 |
| $3 \cdot 10^5$ Hz | 256.1 | 21.3 |
| $5 \cdot 10^5$ Hz | 153.7 | 12.8 |
| $1 \cdot 10^6$ Hz | 76.8 | 6.4 |

Table 4.7: Possible number of neurons whose spike events can be transported via a single physical link depending on the mean spike frequency $\overline{\nu}$ of the single neurons. The calculation use a framing parameter of $f$=60 slots per frame divided into 5 reservation periods of $m = 12$ slots each.

The calculations show that only the events of parts of the 384 neurons of a `Spikey` chip can be transmitted to other chips. In the case that inter-chip connections require to cross multiple links, the total amount of bandwidth usable for a single connection is further reduced. The results to calculate the available physical event rates depending on the neural network topology and on the resulting connection mapping onto the physical topology. The influence of the possible network topologies on the available bandwidth for isochronous connections is continued in more detail in section 5.4.

### Guaranteed Bounded End-to-End Jitter

*hop-independent for single-slot connections*

The jitter of the isochronous connections calculates according to section 3.7.4. The implementation uses $S = G = 2$ clock cycles. Furthermore, the regular topology results in comparable transmission delays and the same slot shift $s_e$ on all links (cf. section 4.4.4). The jitter therefore calculates according to equation 3.39 to

$$J_c \leq 2 \cdot m + 1 \tag{4.8}$$

clock cycles, where $m$ is the size of the reservation period. The maximum value arises for *single-slot* connections ($m_c$=1) and is independent on the number of network hops.

If multiple slots are assigned for a single connection, the jitter is reduced and depends on the slot distribution within the reservation period. The minimum possible jitter $J_c$ is three clock cycles.

### 4.8.2 Demonstrator Application for Isochronous Transfers

This section describes the demonstrator application for connection-based traffic, which is part of the digital design. Its purpose is to demonstrate the usage of the isochronous connections provided by the transport network. It can be used as a starting point for future developments. It is implemented within the VHDL module `reference_app`. An overview of the demonstrator application can be seen in figure 4.27. It provides the following features:

*features*

- The application is connected to the transport network via the connection-based interface of the bypass-switch. The number of ports can be configured. Separate data paths for the transmission and the reception of isochronous data are used.
- The application measures the delay of the isochronous connections in multiples of clock cycles.
- Independent memory blocks for each local port store data to be transmitted and data received to verify the reliability of the transport.
- A pseudo-random logic generates traffic patterns with a configurable probability for each data slot to simulate neural-network events from the `Spikey` chip.
- The global synchronous signals (GSS) of the synchronization sublayer controls the synchronous start of the transmissions on multiple nodes.

The application is controlled via the SlowControl. The several functionalities are demonstrated with the software `mgtroute` of section 4.10.3 after the network has been properly synchronized and the routing tables of the switches have been configured. The following section is reduced to a brief description of the functionalities of the digital design. The evaluation of the measurements is presented in section 5.3.2.

#### Measurement of Application-Layer Delays

*reflection of test data*

The application is able to measure the delays of the isochronous connections at the application layer between different network modules with the precision of a single cycle. To do so, the application transmits a dedicated value to the network within a single time slot. At the same time, a timer is started, which counts clock cycles of 156.25 MHz. The data is returned to the application either within the data link layer (by configuring the routing tables) or at the application layer (by configuring the application at the end-point of the connection to reflect all received data directly to the transmit interface). At the time the data value arrives back at the initially sending application, the timer is stopped. The test requires an appropriate route to be configured, either a round-trip through the network or a bi-directional route between the sending application and the reflecting application. This can be done with the program `mgtroute`. The measurement and the reflection logic is implemented for the local port 0 only.

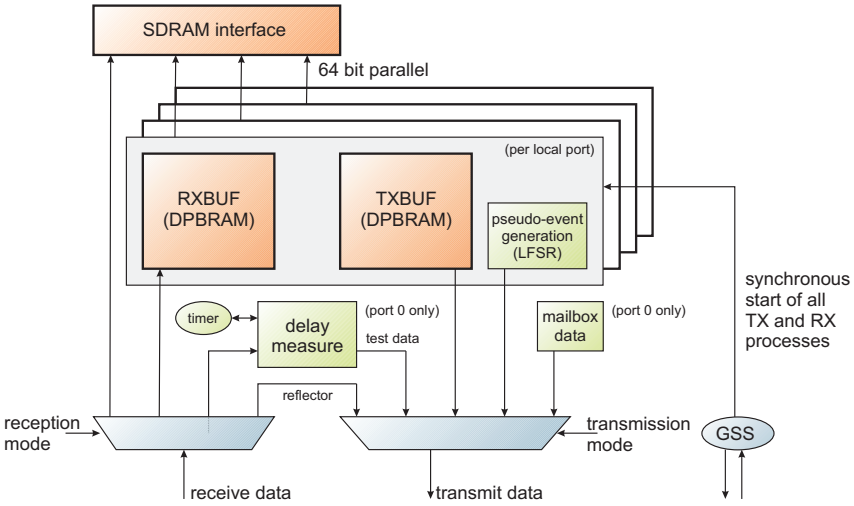



Figure 4.27: The demonstrator application for isochronous connections can be used as a starting point for the development of an interface between the transport-network and the ANN controllers. It features separate dual-port block-rams for data receipt and transmit at each port, it measures the delays of the isochronous connections, supports a synchronous distributed operation and a configurable data rate. All functionalities are accessed via the SlowControl of the framework (not shown).

### Traffic Generation

The application features two different methods to generate traffic:

- The periodic transmission of a dedicated data word (mailbox data) to verify the forwarding process of the switches.
- The playback of a configurable data block.
- The traffic generation with a pseudo-random probability for each time slot.

*transmission of dedicated values*

The first method allows to write a dedicated value into a mailbox-register of the application. The value is transmitted within each possible time slot. The forwarding of the data through the network can be verified by checking the appearance of the data word on the affected network nodes. This is done by configuring the transport network to sample the received frames periodically into a DPBRAM of the FPGA (not shown in the figure). This has been used for low-level debugging during the development of the network.

*memory playback*

The playback functionality uses the DPBRAM blocks of the FPGA. Each port contains its own transmit buffer (`TXBUF` in figure 4.27) of 2 KByte. At the time the transmission is started, the design sends the content of the buffer slot by slot until all content has been sent.

*pseudo-random events*

The pseudo-random transmission of data can be used to simulate an ANN controller of the `Spikey` chip as the data source. The transmission probability per slot can be configured in 16 steps to up to 100 % slot usage. The numbers are generated using a linear feedback shift register (LFSR) [109], which allows a compact implementation within the programmable logic. Each port contains a separate LFSR, which can be initialized individually.

The start of the transfer is performed synchronously on all network nodes with the precision of a cycle of the global reference clock. This is done by using the GSS service of the synchronization sublayer. The $GSS\#0$ is used for this case. *synchronous start of transmission*

#### Traffic Reception and Data Sampling

The received traffic can be sampled to two different locations for a verification: *two sample locations*

- In a separate DPBRAM block of the FPGA for each port.
- To the local SDRAM chip of the `Nathan` network modules for all local ports in parallel.

Both storage types have different advantages. A single DPBRAM is fast enough to store the data of a single port, even if data is constantly received in each cycle. As a drawback, a DPBRAM is only 2 KByte in size.

In contrast to that, the SDRAM chip features 512 MByte of memory, but cannot fast enough be independently accessed by each port. The reason is that the storage of the received data of multiple ports to different address blocks of the SDRAM requires to switch its internal banks, which causes multiple cycles of delays each. For that reason, the sampling of the received data is implemented by the consecutive storage of the data of all ports in parallel. As an example, the usage of four local ports of 16 bit requires to store 64 bit at a rate of 156.25 MHz. The data is only written in the case that at least a single port receives valid data. The probability of the pseudo-random data generator can be reduced to keep to the speed of the SDRAM. *SDRAM limitation*

## 4.9 Distributed Shared Memory

The distributed shared memory (DSM) subsystem is the packet-based transport service of the implemented transport network. Its purpose is the on-demand exchange of larger amounts of non-neural data (cf. section 2.4). Example data includes the neuron plasticity values (weights), neural configuration data as well as communication data for the processes running on the local PowerPCs. The memory transfers are executed between the SDRAM chips on the `Nathan` network modules and also the single FPGA on the backplane. Although developed for this application, the DSM is a general high-level transport mechanism feasible to be used with other applications as well. *motivation*

The DSM provides a convenient interface for processes within the programmable logic. Data transfers within the DSM are initiated by accessing a global, virtual and linear address space in which all possible destinations are included. The DSM protocol implements a client-server architecture with reliable end-to-end connections between the requesting process and the destination memory module. The protocol handles network layer issues and transport layer issues such as fragmentation, flow control and in-order delivery. Since the interface of the DSM to the programmable logic equals the access of the local SDRAM chips, existing algorithms can be extended to operate on global data instead of local memory with minimum programming effort. *characterization*

The DSM subsystem is designed for on-demand data transfers, for which the time *traffic class*

of the transmitted requests cannot be predicted and for which not the transmission delay and jitter, but the *data reliability* is the main QoS qualifier of interest. Consequently, DSM data is transported within best-effort packets of the underlying MCGN implementation and not within isochronous connections.[5] Best-effort packet transfers have higher delay and jitter than isochronous connections but do not require the reservation of physical bandwidth in advance.

### 4.9.1 Overview

Figure 4.28 shows a block schematic of the DSM subsystem. It consists of multiple processes implemented in different network layers:

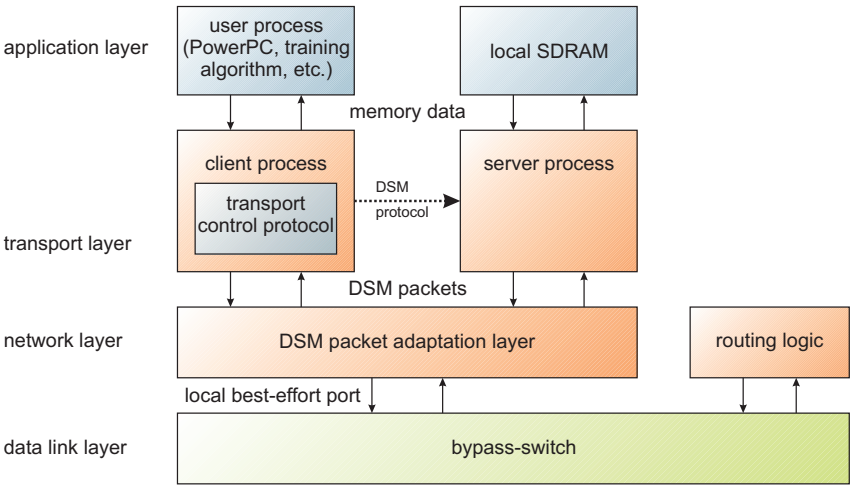



Figure 4.28: Overview of the distributed shared memory subsystem. The client-server design provides reliable exchange of SDRAM contents using global best-effort packet transfers.

1. The DSM protocol uses a high-level client-server design. User processes are connected to the DSM *client process*. The SDRAM memory module is accessed by the DSM *server process*. Each network node may implement a server process and multiple client processes independently.
2. The user interface supports data transfers of up to 4 KByte with a single request. The DSM client can be accessed via the SlowControl, via a `ramclient`-interface [125] or by the local PowerPC via its PLB [152].
3. The transport layer is implemented asymmetrically within the client process only. It ensures reliable end-to-end communication by using an ARQ algorithm [143]. Lost packets are repeated transparently to the user process.
4. The PAL interconnects the DSM subsystem to a local best-effort port of the bypass-switch. It multiplexes and demultiplexes outgoing and incoming packets for the client and server processes within the layers above.
5. The routing of the best-effort packets is performed by a compact and efficient routing algorithm, which supports multiple backplanes (cf. section 4.7).

[5] The third global transfer mechanism of the framework, the SlowControl, is designed merely for configuration purposes.

All parts of the DSM subsystem have been implemented within the programmable logic of the FPGAs on the `Nathan` network modules and on the backplane. The latter may contain a client only since the backplane does not provide local SDRAM memory.

The following sections describe all parts of the DSM subsystem. First, an overall functional description is given including the memory model and the packet format. The succeeding sections describe the client process, the user interface, the server process and the implemented transport control protocol in more detail.

### 4.9.2 Functional Description

*client-side*

The DSM protocol operates as follows: A user process signals the memory read and write requests to the client it is interfaced. The global destination of the memory transfer is included in the address of the request. The client compiles a DSM packet header with the appropriate command and sends the packet to the server using best-effort packet transfers of the MCGN implementation. Since packets may be dropped on its global route due to erroneous links or full VOQs, the transport process within the client ensures that dropped packets are re-transmitted to provide a reliable service. It also respects the maximum packet size of the network and divides larger memory transfers in multiple smaller packets that fit into the VOQ buffers of the switches. The packet forwarding is performed according the routing algorithm that calculates the intermediate path between the network nodes (cf. section 4.7).

*server-side*

After the arrival of the packet at the server, the memory operation is executed at the local SDRAM. The server process then compiles and transmits an acknowledgment (ACK) packet back to the client. The ACK packet also contains the read data in the case of a memory read command. The server process has been developed to be of small size to save programmable logic. It does not implement any transport layer functionality except the return of the ACK. The server handles all incoming packets independently from each other.

#### Memory Model

*memory consistency model*

The reader may notice that the implemented DSM subsystem misses an important characteristic aspect of a general DSM architecture: a *memory consistency model* [107]. Such a model is required to handle the problem of multiple write and read operations by different client processes to the same global address, e.g. to update local caches after global writes.

*coherency protocol*

A memory consistency model is usually implemented using a *coherency protocol* that specifies extra transmissions between the network nodes [107]. The coherency protocol specifies how to inform processes on other network nodes of changed data or how to implement a central global directory of data changes. Another solution is to lock specific addresses to ensure the exclusive access to the corresponding memory location (see e.g. [147]).

*implemented model*

The alternative memory model provided by the implemented DSM protocol has the following features:

- There is no memory consistency model. The user of the framework has to be aware of this while designing algorithms that use DSM data transfers.

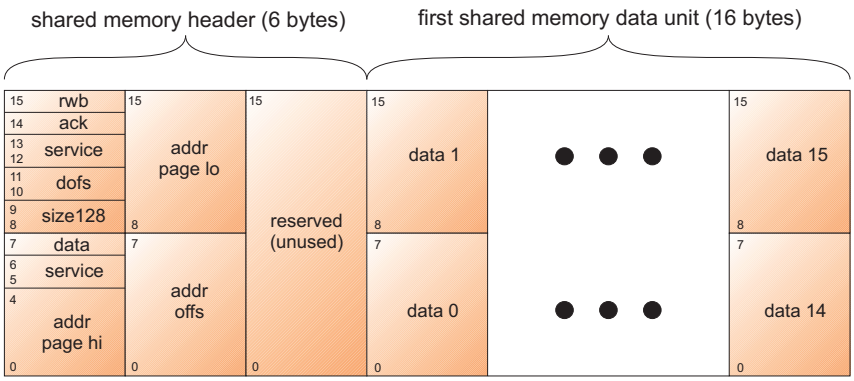



Figure 4.29: Data format of a shared memory packet.

- All addresses within the DSM are globally unique, i.e. a request to a specific address is executed at the same physical memory location independently of the location of the requesting process.
- The global address space is divided between all participants. The current packet header allows to assign a window of 16 Mbyte of each physical memory to be accessible by the DSM. The position of the window within the local memory module is configurable at the server.
- Memory transfers are addressed and executed with the granularity of *data units* of 128 bit (16 byte), which is the minimum amount of data to be read or written to the double data rate (DDR)-SDRAM chips. A PowerPC cache line transfer of 32 byte therefore requires two data units to be transported.
- The address space is further divided into multiple memory *pages* of 4096 bytes each (256 data units). Memory transfers are executed only within a single page. This is to reduce the amount of programmable logic required for address comparators and counters.

*consistency model by user*

Even having these limitations, the DSM provides a convenient and reliable service to globally move large-scale amounts of data. The user has to define the meaning, the reading processes and the writing processes for each memory location used with the DSM in advance. This enforces a clear intention of the timing and the logical behavior of the application. The software executed on the PowerPCs is able to explicitly invalidate cache lines to enforce the updates of read data or to flush write data by using dedicated CPU commands.

### Packet Format

DSM packets are the payload of best-effort packets and are identified with a protocol type of 0 within the best-effort header (cf. section 4.2.2). Since the best-effort header occupies a single data slot, the DSM packet starts at the second slot of a best-effort packet. The header is a result of the first development of the simple ARQ algorithm of the transport layer (see below). It is defined within the VHDL module `shm_pck`. Figure 4.29 illustrates the DSM packet format and table 4.8 describes the meanings of the several fields.

| field | meaning |
|---|---|
| rwb | denotes a read (1) or write (0) transfer |
| ack | the frame is a server ACK |
| service | service id of packet source (client $n$, server) |
| dofs | offset of first data word to support PLB *target word first* |
| size128 | number of valid data units in packet |
| data | denotes the frame contains valid (read or write) data |
| seq | sequence number of packet |
| addr page | page address (in data units) |
| addr offs | offset address within page (in data units) |

Table 4.8: Description of the fields of the shared memory packet header.

*header field*

Most header fields are self-explaining and are not further discussed here. The fields `rwb`, `ack` and `data` denote whether the frame contains read data, write data or is an ACK of the server. The `dofs` field is reserved to support the *target word first* functionality of the PLB of the PowerPC [152, 61]. This functionality is currently not implemented. The header further contains a 16-bit field that is reserved but currently unused. It can be used for later improvements of the ARQ algorithm to store larger sequence numbers or to support a larger address space. The current address field of 20 bit allows to access an address range of 16 Mbyte on each destination. The global source and destination nodes are not included into the DSM header, since this information is already stored into the best-effort packet header.

*data field*

The DSM transfers data only in multiples of data units of 128 bit (16 byte). The present reference implementation uses a fixed size for best-effort packets of 44 byte. Since 6 bytes are used for the best-effort packet header and trailer, one or two DSM data units can be transported per packet, which allows to transfer a complete PowerPC cache line within a single packet. The packet size can be configured to match upcoming applications.

### 4.9.3 The Client Process and the User Interface

*purpose*

The DSM client is interfaced by the user process and contains the transport control functionality. It is connected to the underlying MCGN network via the PAL for the DSM protocol type. The client has been implemented within the VHDL module `shm_client`. A schematic of the client is shown in figure 4.30.

*user requests*

To execute a DSM transfers, the user process denotes a request by asserting the `req` signal together with the qualifiers `address`, `rwb` and `size`. The client answers the user process via the `ack` signal at the time the request has been taken over. The user process may set a new request immediately.

*data transfers*

The data is transferred via the `wr_data` and `rd_data` signals at the time the client asserts the `wr_strobe` and `rd_strobe` signals, respectively. The timing of the read path and write path is not correlated with the `req` and `ack` signals. The user process has to consider that no further buffering is performed within the switch. The user process cannot delay or inhibit data transfers. Figure 4.31 shows a simulation

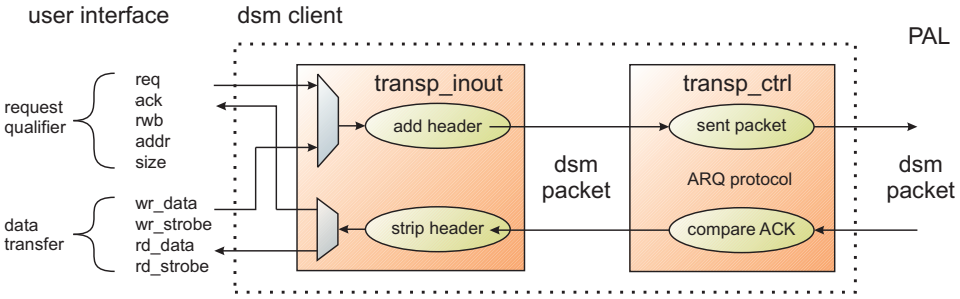



Figure 4.30: Schematic of the DSM client process containing the transport control functionality.

of two example transfers. A read request for a single data unit is followed by a write request for three units. The corresponding data is transferred accordingly.

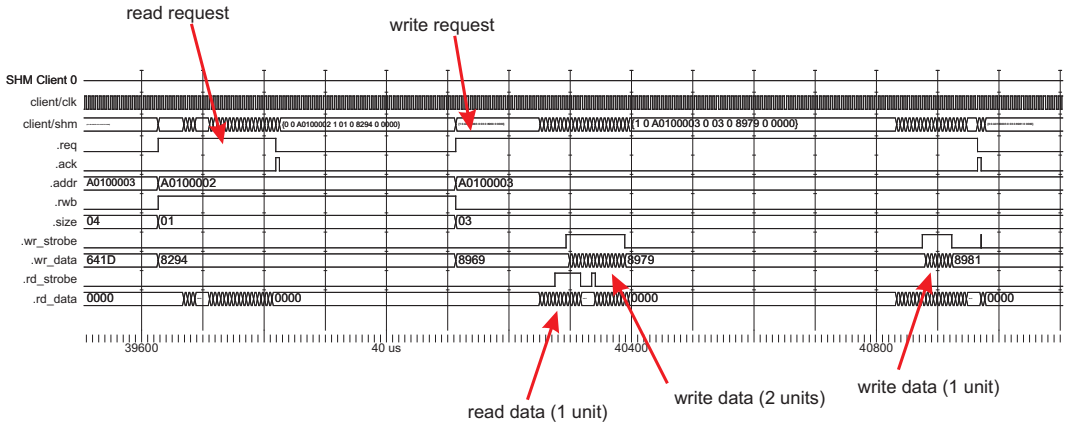



Figure 4.31: Timing diagram of the user interface to the DSM client process. The user process requests a read and a write transfer. The data is transferred accordingly.

*implemented user modules*

To demonstrate the access to the DSM, multiple modules have been implemented in VHDL. The DSM can be accessed via the SlowControl (`cm_shm_cache`) and with a dedicated test module that writes and verifies the written data (`cm_shm_test`). To facilitate the usage of the DSM subsystem, the VHDL module `ramclient_shm` has further been developed. Its purpose is to seamlessly add framework-wide high-level communication to existing processes within the programmable logic. For that reason, it provides the same interface used to access the local SDRAM memory chip [125], but converts the requests to the DSM interface of the client process. The module contains the necessary buffers and aggregates multiple requests to subsequent addresses into a single DSM request.

### 4.9.4 The Server Process

*purpose*

The DSM server process is connected to the local memory chip and processes the memory operations which it receives from the client via the network. The server executes a memory operation independently for each incoming packet. The address of the packet header is mapped into a local window of 16 MByte. The position of

the window within the local memory can be configured freely. After the memory operation has been finished, the server transmits an ACK packet back to the client. The server process is implemented within the VHDL module `shm_server`.

*no transport control*

As already stated above, the server does not implement a transport control protocol. It uses the module `tranp_inout` only to compile and to strip the DSM header to and from the packet. Packets or ACKs that get lost are detected by the transport protocol implemented within the client that re-requests the memory operation.

*functional description*

A schematic of the server process is shown in figure 4.32. Incoming write data is sent to the local SDRAM immediately. The packet qualifiers (address, size etc.) are stored for the transmission of the ACK packet. The server process uses two FIFO queues. The request FIFO stores multiple requests for the case that different clients are requesting DSM operations at the same time. The read data FIFO is necessary since the server has to wait until all read data from the SDRAM is available. Each received packet is finally acknowledged with an ACK packet.

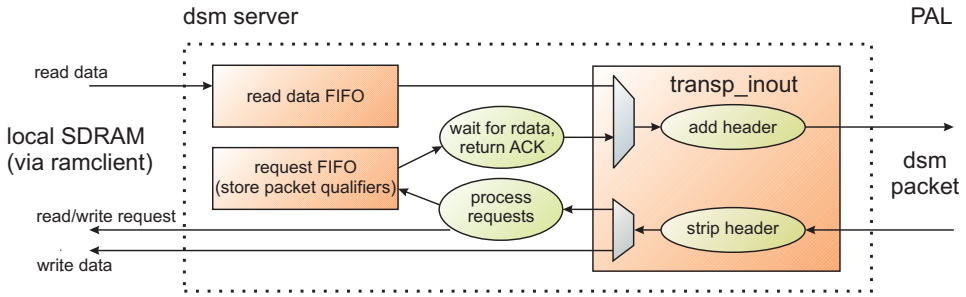



Figure 4.32: Schematic of the DSM server process. Write data is sent to the memory immediately, read data is collected in a FIFO until the requested amount is available. Each received packet is independently acknowledged.

### 4.9.5 The DSM Packet Adaptation Layer

*purpose*

The DSM packet adaptation layer (PAL) is the interface between the DSM client or server processes and the local-best-effort port of the bypass-switch. Its purpose is to hide the slotted timing of the switch before the higher-level processes to simplify their design. The PAL performs basically three tasks:

1. It performs the conversion between data slots and packets. The packets are exchanged with the client and the server via a 16 bit bus at 156.25 MHz with additional strobe-signals and header information.
2. It multiplexes a single best-effort port of the switch to multiple clients and to the server. This is used to reduce the complexity of the switch. It is also possible to use a single PAL for each DSM process.
3. It adds and removes the best-effort packet header to and from the DSM packet.

*implementation*

The implementation consists of two separate and independent data paths for transmission and reception of DSM packets. Each data path features a small state machine to extract header information and to control the data flow. No buffering of packets is performed. The PAL has been implemented within the VHDL module `packet_io`. Figure 4.33 shows a block schematic of the PAL.

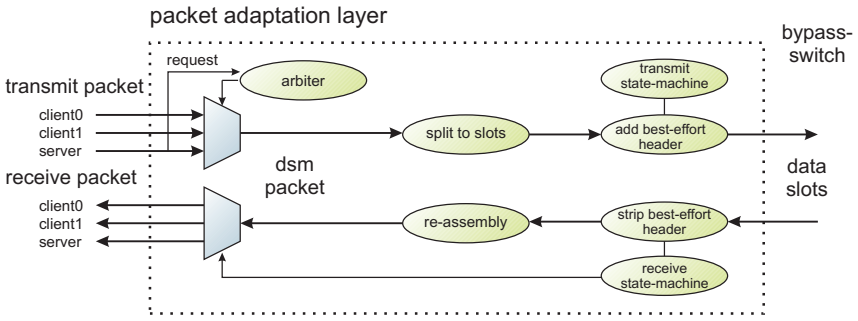



Figure 4.33: Block schematic of the DSM packet adaptation layer.

### 4.9.6 Transport Control Protocol

The purpose of the transport control protocol is to hide the packet-based (and thus unreliable) nature of the underlying data link layer to ensure a reliable end-to-end communication between the client process and the server process. The functionality of the transport control protocol is located within the transport network layer. The main part is implemented within the client process only (cf. figure 4.28). The error handling and the necessary re-transmissions due to lost packets are done transparently to the DSM user process. The functionality is implemented within the two VHDL modules `transp_inout` and `transp_ctrl`. The development has been done in cooperation with Christian Gutmann.

#### Provided Services

The transport control process of the DSM implements an ARQ or PAR [143, 110] protocol with the following functionality:

1. Larger memory transfers are fragmented into multiple packets of each up to the maximum size of a best-effort packet.
2. All packets are delivered in the correct order.
3. Lost packets are automatically re-transmitted. Packets can get lost at the input queues of the switches due to invalid CRC checksums or full queues.
4. A simple flow control mechanism between the two end-points adapts to the packet loss rate to avoid buffer overflows at the server or congestion within the network.

*limited complexity*

The implementation of the ARQ algorithm performs a modified version of the common *stop-and-wait* algorithm, which is a small and simple protocol for the above tasks [143, 110]. The transport process does not implement a complex algorithm as used by the protocols TCP/IP [11] or X.25 [143]. The central implementation allows a seamless update to more complex protocols later. However, the correct implementation of a protocol like TCP/IP is far to complex to fit into the programmable logic of the given FPGA.

#### The Implemented Stop-And-Wait Algorithm

*smart sender / dumb receiver*

The intention of the development has been to follow the rule *smart sender / dumb*

*receiver* [110], which concerns an asymmetric implementation that minimizes the complexity of the receiver. This can be done since the provision of the above features requires only the sending side or the initiator of a request (the DSM client) to handle the complex part of the protocol. The listener (the DSM server) only reacts on incoming packets with the sending of a packet of its own and never sends a packets initially.

*algorithm description*

The algorithm can be described with a state-machine: At the beginning, the client process and the server process are in the `idle` state. At the time a user requests a DSM transfer, the client sends the appropriate command packet to the server and enters the `waitack` state. After the packet arrives at the server process, the server leaves the `idle` state and executes the requested memory operation. The server then sends a positive ACK packet back to the client process and re-enters the `idle` state. Data to be written or being read is send together with the command packet or the acknowledge packet, respectively. At the time the ACK packet arrives at the client, the client reports an acknowledgment to the user and re-enters the `idle` state. The client process further transmits a sequence number together with each command. The sequence number is changed with every succeeding command and returned by the server process to unambiguously identify the original packet, the ACK belongs to. The algorithm is illustrated in figure 4.34.

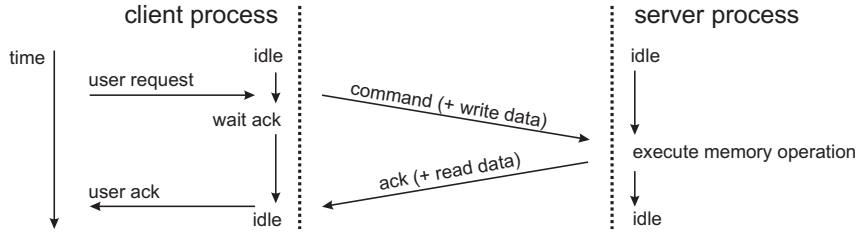



Figure 4.34: Illustration of the the implemented stop-and-wait algorithm.

*error detection*

After the client has transmitted its command packet, it starts a local timer counting down from a certain initial value. In the case that the command packet or the ACK packet gets lost, the timer reaches zero, which causes the client to retransmit the command packet. This process is illustrated in figure 4.35. As a difference to the commonly known stop-and-wait algorithm, the implemented version adapts the initial values of the timers to the packet loss rate of the network. Timers that are restarted are initialized with increasing values to implement a minimum of flow control and congestion control.

### Stability

Although being quite simple, the implemented version of the stop-and-wait algorithm ensures reliable end-to-end packet transports. It succeeds to handle the following tasks:

1. Packets dropped due to invalid CRC checksums or full buffers of the VOQs or of the server process are re-requested automatically after the client timer reaches zero. *packet loss*

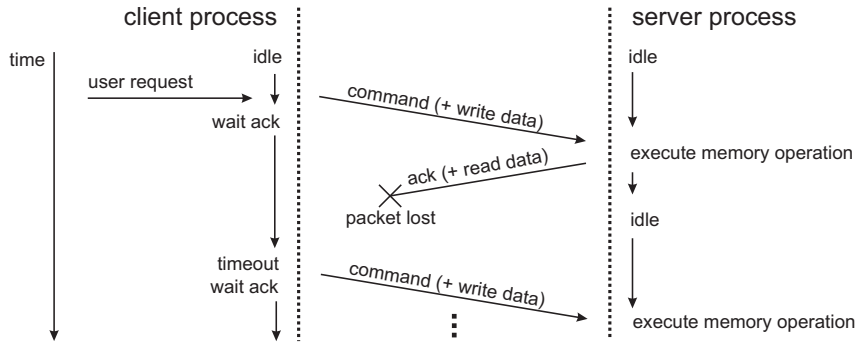



Figure 4.35: The stop-and-wait algorithm automatically re-sends a packet if the positive acknowledge is timed out.

*packet order*

2. Packet order is preserved since the sequence number identifies the ACK for each packet. The implemented algorithm accepts only the single currently outstanding packet. Packets that are delayed and out of order are rejected (dropped).

*flow control*

3. Flow control is performed since the client requests the next packet not earlier the current packet is acknowledged. This protects the server from being flooded by a single client.

*congestion control*

4. Congestion control is performed only very basically by the adaptive timer value that is adjusted according to the loss rate of the network.

*duplicate ACKs*

5. The implemented algorithm ignores duplicate acknowledges for packets already acknowledged to avoid the *sorcerer's apprentice syndrome* [10, 143][6].

*requirement to the routing algorithm*

The small size of the sequence number furthermore requires a deterministic routing algorithm that selects the same route for all packets to be routed between the same source and destination nodes. Non-deterministic or adaptive routing algorithms require larger sequence numbers to ensure a correct identification of a received packet and to avoid ambiguities, since different routes could cause large delay differences. Furthermore, the implemented algorithm requires a routing algorithm that provides in-order delivery. Although the out-of-order delivery of packets does not smash the transport protocol, is results in very low performance due to the resulting packet drops.

### Implementation of the Transport Control Protocol

*fragmentation*

The transport control protocol of the DSM subsystem has been implemented in two different VHDL modules that are part of the DSM client and server processes. The module `transp_inout` interfaces the user process, compiles the DSM packet header and performs the fragmentation of requests for large amounts of data into multiple independent requests to keep the server process simple.

*ARQ algorithm*

The ARQ algorithm stop-and-wait is embedded within the client process within

[6]The sorcerer's apprentice syndrome is an effect caused by a flaw in the implementation of the stop-and-wait protocol within the TFTP [134, 135] network protocol. It leads to the infinite transmissions of duplicate packets.

the module `transp_ctrl`. The module consists of two state machines that write and read packets from the central packet buffer. Since the present implementation supports only a single outstanding ACK, the buffer stores only a single packet. The module also contains the timeout counter, which is controlled by the state machine that reads the packets from the buffer. The data of received packets is accepted only if its sequence number matches the currently outstanding packet. Figure 4.36 shows a block schematic of the module.

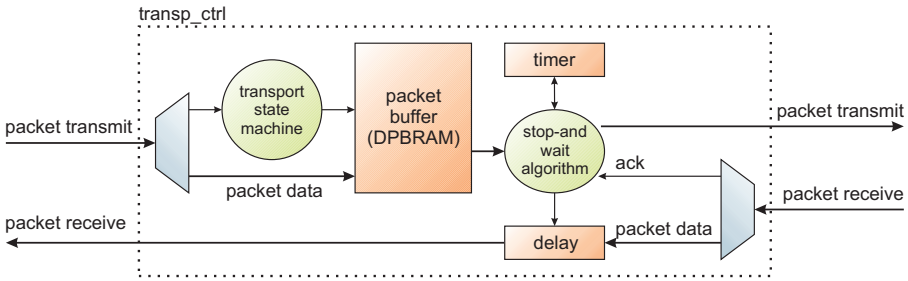



Figure 4.36: Schematic of the implemented stop-and-wait algorithm. A single outstanding ACK packet is supported, which requires to buffer the current packet for possible retransmissions.

### 4.9.7 DSM Performance

*statistical QoS results*

In contrast to the QoS calculation of connection-based priority traffic, the transmission of DSM data within best-effort packets does not allow to give exact QoS guarantees for the delay and for the achievable throughput, but provides QoS in a statistical manner: The values for the packet delay and the throughput differ between multiple transfers on the same route and also between different routes and network nodes. This is caused by two main factors: First, by the external framing of priority data that periodically inserts gaps of two cycles between the data slots. Second, by the interaction with other DSM transfers due to the occupancy of intermediate buffers and the resulting fraction of bandwidth that is granted by the best-effort schedulers.

*cycle-accurate simulations*

The following paragraphs discuss the resulting DSM performance in terms of the latency at the user interface and the achievable transfer rate. The results have been achieved with a cycle-accurate simulation of the digital design. The discussion makes the following assumptions: all intermediate buffers are empty, no reservations for priority data have been made and all packets use the maximum possible size of up to two data units (32 byte).

#### Simulation of the Latency

*sources of latency*

The latency of a DSM request is the time a user process has to wait for a DSM read or write operation to complete. This is an important factor especially for small transfers of e.g. control or status data. Although this is not the main intention of the DSM, it is worth to consider. The latency of a DSM operation is determined by the round-trip time (RTT) of a single DSM command and its succeeding ACK. To estimate the RTT, several delays have to be considered:

1. The main delay is caused by the transport logic within the client for the buffering of the packet, the read-out and the delay for the verification of the ACK.
2. The server requires time to store the header of the incoming packets into its local FIFO queue to be able to send the corresponding ACK packet. The server further has to wait for the arrival of the read data from the SDRAM in the case of read operations.
3. The physical layer introduces a fixed amount of delay for the synchronization and the physical transmission. The exact number of cycles depends on the selected framing parameters (cf. section 4.4.4).
4. A significant amount of delay is introduced by the VOQs since the switch is forced to use store-and-forward packet switching. Incoming packets are completely stored before the scheduler is requested and the packet is forwarded.
5. Few additional clock cycles are required to traverse network layers, the crossbar or the multiplexer within the PAL.

*delay values*

The delays of the physical layer and of the data link layer (i.e. the bypass-switch) have to be considered multiple times according to the number of intermediate switches between the network nodes of the client process and the server process. It has been stated above that exact delay values cannot be predicted. Table 4.9 lists typical delay values for the round-trip of a 32-byte DSM packet.

| location | tasks | typical delay |
|---|---|---|
| DSM client | transport logic, buffer packet, verify ACK | 17 cycles |
| DSM server | store header in FIFO, retransmit ACK | 16 cycles |
| bypass-switch | storage and read-out of VOQs, best-effort scheduler | 30 cycles |
| synchronization sublayer | synchronization | 5 cycles |
| physical layer | physical coding & transmission | 20 cycles |

Table 4.9: Typical delay values of DSM transfers. The delays of the data link layer and of the physical layer have to be considered at each network node of the round-trip route of the packet. The exact values differ by few cycles between between different packets and network nodes (see text). All intermediate buffers are expected to be empty.

*latency for single reads*

The resulting latency for a single operation depends on the number of intermediate network hops. Table 4.10 lists typical latency values for read transfers of a single data unit of 16 byte. The values are acquired with a cycle-accurate simulation of the digital design and denote the time at the user interface between the initial request and the arrival of the first read data. Write requests have comparable latencies. The interface acknowledges write requests at the time the write data is taken over by the client.

Note that the main latency is caused by the lower network layers for the buffering within the VOQs and by the internal data paths of the MGTs. These delays can hardly be improved and do not belong to the DSM subsystem.

| number of intermediate hops | typical latency | |
|---|---|---|
| single-hop | 181 ± 8 cycles | 1.16 ± 0.05 us |
| 2 hops | 289 ± 8 cycles | 1.85 ± 0.05 us |
| 3 hops | 396 ± 9 cycles | 2.53 ± 0.06 us |

Table 4.10: Simulated typical latency values for read transfers of a single data unit of 16 byte. The values are measured at the user interface of the client and denote the time between the request and the arrival of the first read data.

### Simulation of the Transfer Rate

*calculation of transfer rate*

The throughput or transfer rate is mainly determined by the implemented stop-and-wait algorithm, which operates with a single outstanding packet. Most of the time, the algorithm is waiting for the returning ACK. Since the succeeding packet is sent immediately at the time the ACK arrives, the overall transfer rate of the algorithm depends on three factors:

1. The RTT of the packet and its ACK packet.
2. The amount of valid data units that are transferred within a packet.
3. The fraction of data slots that is available for best-effort transfers.

Taking these factors into account, the transfer rate calculates to:

$$\text{transfer rate} = \frac{\text{valid data per packet}}{\text{RTT}} \cdot \text{fraction of slots available} \qquad (4.9)$$

The RTT to be taken into account for the measurement of the throughput is the time between the transmission of a DSM command packet and its returning ACK, both measured at the transport control process (and not at the user interface). The resulting values are smaller than the RTT values measured at the user interface of table 4.10. Due to the fixed packet size, it can be assumed that two data units (32 bytes) are transmitted per packet to maximize the transfer rate.

*throughput for successive writes*

Table 4.11 lists simulated values for the RTT and the resulting DSM transfer rate for typical synchronization parameters of the framework. The data slots have been estimated to be all available for best-effort transfers. The efficiency is calculated according to the maximum physical bandwidth of 312.5 MByte/s.

*improvement of the ARQ algorithm*

The values show that the transfer rate of the simple stop-and-wait algorithm is far below the physical bandwidth. The transfer rate can be increased significantly by a more complex version of the ARQ algorithm. Since the ARQ algorithm is implemented solely within the VHDL module `transp_ctrl`, it can be updated easily in future. However, a commonly used variant like TCP/IP [11, 143, 110] is far to complex to be implemented within the limited amount of the programmable logic of the given FPGA, which enforces a more simple version to be used.

### Future Development

*sliding window protocol*

An improved ARQ algorithm is required to constantly transmit packets and to allow

| number of intermediate hops | typical RTT | typical delay-bw. product | stop-and-wait transfer rate | efficiency |
|---|---|---|---|---|
| single-hop | 146 cycles | 292 byte | 34.3 MByte/s | 11.0 % |
| 2 hops | 255 cycles | 510 byte | 19.6 MByte/s | 6.2 % |
| 3 hops | 364 cycles | 728 byte | 13.7 MByte/s | 4.3 % |
| 4 hops | 473 cycles | 946 byte | 10.6 MByte/s | 3.3 % |

Table 4.11: Simulated typical RTTs and resulting transfer rates for single-hop and multi-hop distributed shared memory transfers. The simulations have been performed for successive write transfers with 32 byte of memory data per packet.

for a larger amount of outstanding non-acknowledged packets. The necessary amount of buffering space for such a *sliding window* algorithm is determined by the *delay-bandwidth product* of the client-server connection. The delay-bandwidth product equals the amount of data 'on the line' that has to be sent by the client until the first ACK packet arrives. This is the minimum amount of packet data that has to be stored by a sliding window protocol for the most effective usage of bandwidth. Table 4.11 also shows the delay-bandwidth products for the different number of hops (two byte are transfered per cycle).

*implementation*

The implementation of the protocol may keep to the central packet buffer within the transport logic of the client of the stop-and-wait implementation, but requires an independent access to lost packets for re-transmissions (no FIFO logic). Each packet gets its own independent counter for the timeout detection. Incoming acknowledgments have to be assumed to arrive out of order to consider lost packets. An acknowledgment removes its dedicated packet from the buffer. Example algorithms are the *go back n* algorithm or the *selective repeat* algorithm, which are both explained in detail in [143].

*theoretical maximum*

The maximum possible transfer rate of an optimal ARQ algorithm that is constantly transmitting packets depends on the packet size. This is since the unusable part of each packet is 6 cycles (12 byte) for the best-effort header, the trailer and the DSM header. Table 4.12 shows the theoretical maximum transfer rates depending on the packet size. The amount of bandwidth wasted by the frame gap of the synchronization has not been considered, such that the full bandwidth is the 312.5 MByte/s of the MGTs.

| shared memory data per packet | best-effort packet size | efficiency | maximum theoretical transfer rate |
|---|---|---|---|
| 16 byte | 28 byte | 57 % | 179 MByte/s |
| 32 byte | 44 byte | 73 % | 227 MByte/s |
| 48 byte | 60 byte | 80 % | 250 MByte/s |
| 112 byte | 124 byte | 90 % | 282 MByte/s |

Table 4.12: Theoretical optimal transfer rates of an ARQ algorithm that is continuously transmitting data packets. The values are calculated for the case that all data slots are available for best-effort packets. The synchronization frame gap has been ignored.

### 4.9.8 Summary

The section described the implementation of the distributed shared memory (DSM) subsystem of the transport network. The DSM uses the best-effort packet transfers of the underlying MCGN network to provide on-demand remote access of the distributed SDRAM chips on the `Nathan` network modules to processes within the programmable logic of the FPGAs. The DSM basically consists of a client process and a server process. The client process implements the user interface and contains the transport control protocol. It has been shown that the implemented DSM features reliable end-to-end communication. The QoS performance in terms of latency and transfer rates have been discussed. The latency is mainly determined by the lower network layers and can hardly be improved.

*improvements of the transfer rate*

The resulting transfer rate is presently sub-optimal, but can be improved easily due to the modular design with the following modifications:

1. The usage of the sliding-window algorithm as the ARQ protocol. Although not very complex to implement, this requires multiple packet buffers and a larger amount of programmable logic.
2. The enlargement of the packet size. This improves the fraction of usable data compared to the total data to be transferred. However, a large packet size requires larger buffers at the VOQs and thus results in larger RTTs due to the store-and-forward packet switching mode of the switch.
3. A variable packet size. The present implementation uses a fixed packet size for all packets. A variable size for best-effort packets reduces the size of the ACK and thus improves the RTTs.

The main improvement can be reached by replacing the stop-and-wait algorithm with a sliding window protocol in the case that the additional consumption of logic is acceptable.

## 4.10 Software Development

Most of the software components that have been implemented for the transport network are executed during the initialization phase of the MCGN network and are responsible for the network's synchronization, mapping and configuration. Further functionalities are implemented to demonstrate the features of the transport network at runtime. The software is executed on the control PC and accesses the hardware via the SlowControl (cf. section 2.2.4). The software consists of the following programs:

1. The executable `mgtsync` is used to control the synchronization process, to verify the synchronization state and to measure and list the transmission delays $D$ between the switches of the network nodes. *synchronization*
2. The executable `mapping` performs the connection mapping process, which includes the quantization of bandwidth, the routing of the connections and the calculation of the contention-free slot assignment. The program further generates the content of the routing tables of the switches and the high-level software simulation (cf. section 5.6). *mapping*

*routing table access*

3. The executable `mgtroute` is used to configure the routing tables of the switches, e.g. to list its content, to write the mapping result or to modify particular routes by hand. It further controls the module of the demonstrator application for connection-based traffic to verify the transmissions within the reserved slots of the isochronous connections.

*network generation*

4. The executable `generate_network.py` is used to generate pseudo-random networks. The networks can use up to all neurons and synapses of the hardware. The generated test-networks are used to evaluate the performance of the connection mapping algorithm.

*packet switch simulator*

5. The executable `switchtest` is a software simulator for a general input-queued packet switch with VOQs including the bypass-switch. It is used to evaluate the performance of different best-effort schedulers under various traffic conditions. Another purpose is the verification of the digital design against the simulator.

The programs `mgtsync` and `mgtroute` access the programmable logic. The program `mapping` performs the *connection mapping* algorithms described in section 3.6. It further contains file-IO functionality to read the neural netlists, to read the network topology and to write the routing tables. Since the central tasks and algorithms of the network initialization phase (synchronization and mapping) are already described in chapter 3, the following description is reduced to the basic functionalities. The interested reader may consult the source code or the command-line help of the programs for further details.

### 4.10.1 Synchronization

The program `mgtsync` is part of the synchronization sublayer and accesses the module `phys_sync`. The program is controlled via command-line parameters and does not require further input. It purely operates on the hardware or dumps control information to the screen. `Mgtsync` performs the following tasks:

- The control of basic `MGT` functions like the serial alignment or the reset.
- It writes the framing parameters to the hardware, e.g. the number $f$ of time slots per frame, the delay $\epsilon$ of the delay elements and the logical slot shift $s_e$. Note that the slot duration $S$ and the frame gap $G$ are presently hard-coded into the VHDL design, such that `mgtsync` is not aware of slot sizes. The frame size $T$ is controlled via the number of slots $f$. The values of $f$ and $s_e$ are both denoted in multiples of clock cycles.
- The control of the minimum-delay logic, which reduces the delay of the receive buffer of the MGTs.
- The measurement of the transmission delays $D$ between the inputs of adjacent switches.
- The establishment of the framework-wide synchronization and the check of the synchronization state.

Most functions simply equal the setting or reading of status bits within the hardware design. In the following, the two main functions are described in more detail: the

measurement of the transmission delays and the synchronization of the network modules. The results of these operations are discussed in section 5.2.

### Measurement of the Transmission Delays

*process*

The transmission delays $D$ between the inputs of two adjacent switches are measured as described in section 3.5.5. To do this, the number $f$ of slots per frame is set to a large number. The value $D_0$ is measured by additionally setting the delay elements $\epsilon$ to 0. After the synchronization logic is configured, the arrival times of incoming frames are measured in multiples of clock cycles according to the local time counters. The transmission delays are then calculated according to equation 3.11.

### Synchronization of the Network

*general synchronization process*

To synchronize the network, the synchronization sublayer contains a single periodic timer at each network node as well as an adjustable delay element within the transmit data path of each MGT. The global synchronization is established by adjusting the timers and the delay elements until the synchronization condition is met at each node, i.e. all frames arrive at all MGT inputs properly aligned to frame boundaries. The synchronization condition is detected by the `phys_sync` module within the programmable logic by verifying the arrival times of the `SYNC` characters of the data frames (cf. section 4.4.1). It has been shown that the synchronization can be established such that all timers at all network nodes are strictly synchronized by using the shifted framing method. The network is synchronized by synchronizing currently unsynchronized switches one by one via already synchronized adjacent switches to the network. A node is synchronized to an adjacent node by adjusting its timer and the delay elements on the corresponding link and by keeping the timer of the synchronized node. If all timers have been set, the remaining delay elements between the switches are set.

*timers and delay adjustment*

The synchronization process has to take care for the order in which the switches are synchronized. The reason is that two timers might not be able to be adjusted such that the delays at both switches are measured to the same value. This is caused by the necessary sampling process of the distributed reference clock at each network node, which can cause the measured delays to differ by a single cycle. In this case, the adjustment enforces to increase one of the two delay elements on the bidirectional link for an even timer adjustment. The problem is that the bidirectional measurement does not allow to decide *which* of the two timers to modify. In this case, the timers of both switches are therefore not physically synchronized, but rather logically.

*accumulation of time shifts*

If multiple adjacent switches are synchronized to each other within the same network segment, these errors can accumulate. The problem is illustrated in figure 4.37(a). The figure shows four switches, which are synchronized in clockwise order. The numbers written at the switches denote the absolute shifts in their local timers caused by asymmetric samplings of the links. The switches A to C are already synchronized and D has been synchronized to C. Finally D has to be synchronized to A. The timer shift between A and D is three cycles due to the selected order. Since all timers are adjusted, this last step is done purely by adjusting the delay elements.

Although this effect does not hinder the establishment of the synchronization, this is generally unwanted since the transmission delay should be kept as small as possible.

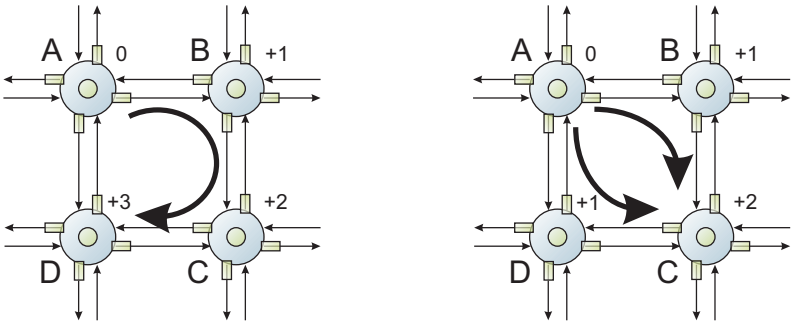



Figure 4.37: Synchronization order of adjacent network nodes. Each node features a single timer and a separate delay element at each link output: (a) Accumulation of timer shifts of adjacent network nodes due to a non-optimal synchronization order; (b) An improved order reduces the timer shifts.

*multi-pass synchronization*

As a general solution for all network topologies, this effect is reduced by a careful selection of the order of the nodes such that long topological circles are avoided. This can be done by executing the synchronization in multiples passes. It starts at a particular node and is extended node by node via all adjacent nodes to the boundaries of the network. Multiple nodes are selected in each pass. The selection comprises all network nodes with the largest number of adjacent nodes that have already been synchronized in previous passes (cf. figure 4.37(b)). The described process is implemented within the executable `mgtsync`.

### 4.10.2 Connection Mapping

*functionality*

The program `mapping` calculates the contention-free slot assignment to configure the routing tables of the switches according to the required connections from given neural netlists. The connection mapping is the second step of the initialization phase after the synchronization has been performed. The important algorithms of the connection mapping process have been described in section 3.6. The data flow of the mapping program is illustrated in figure 4.38.

*input data*

The program `mapping` takes the following information as input:

1. The neural netlist, organized as an adjacency list of the synaptic connections of each neuron to other neurons. The neurons are simply numbered.
2. The neuron placement, which is a location assignment for each neuron of the netlist to its physical counterpart within an ANN chip. The placement is calculated previously and is part of the information to be provided by the user. This neuron mapping should minimize the resulting inter-chip bandwidth requests for isochronous connections (cf. figure 2.11 of section 2.3.3).
3. Command-line parameters, which control the synchronization setting, the division of the frames into mapping periods as well as the number of local ports

of the digital design to adjust the amount of programmable logic. The number of slots per connection can be increased for heavily loaded connections.

4. A hardware description of the network topology in terms of backplanes, network modules and its interconnecting MGTs. The parts are organized in textfiles.

The netlist, the neuron mapping information and the description of the hardware topology are provided as text-files. The mapping program tries to find a valid mapping without further control by the user. The algorithm may succeed or fail during its execution. Besides the given netlist or the hardware topology, the success mainly depends on the configured number of usable time slots per reservation period. After the program completes, the following information are calculated:

1. A neuron-interconnectivity matrix that illustrates the connectivity of the given neural netlist.
2. Statistical information about the neural network: the distribution of the number of outputs or inputs per neuron, the distribution of the number of neurons per connection etc.
3. The routing tables for the switches.
4. Detailed routing information of single inter-neuron connections for the ANN controllers. This is required for the cycle-accurate simulation for large-scale neural networks on the framework (cf. section 5.6).

### Functional Description

*data flow*

The `mapping` program executes several stages. No further user interaction is required. In particular, `mapping` performs the following tasks:

*generation of connection requests*

1. The program converts the netlist information of inter-neuron connections and the physical positions of the placed neurons into requests for isochronous connections between the chips. To reduce the number of isochronous connections, all synaptic connections between neurons on the same source and destination chips are grouped to a single connection. The maximum number of connections is therefore $n(n-1)$ with $n$ being the number of chips. The number of synaptic connections that are transported within the same isochronous connection determines its bandwidth requirement. The bandwidth requirement depends solely on the number of *source* neurons and is independent of the number of the used synaptic inputs on the destination chip.

*bandwidth quantization*

2. Algorithm 3.6.1 of section 3.6.2 is executed to assign an appropriate number of time slots to each connection according to its bandwidth demands. The user can control the maximum number of neuron data transported within a single slot with a command-line parameter. Without setting this parameter, the program assigns a single slot for each connection.

*routing*

3. The connections are routed using algorithm 3.6.2. The algorithm selects a set of consecutive physical links between the end-points of each connection. `Mapping` uses the *shortest-path* algorithm from Dijkstra [143] for this as discussed in section 3.6.3. The algorithm uses link weights to calculate a cost for each

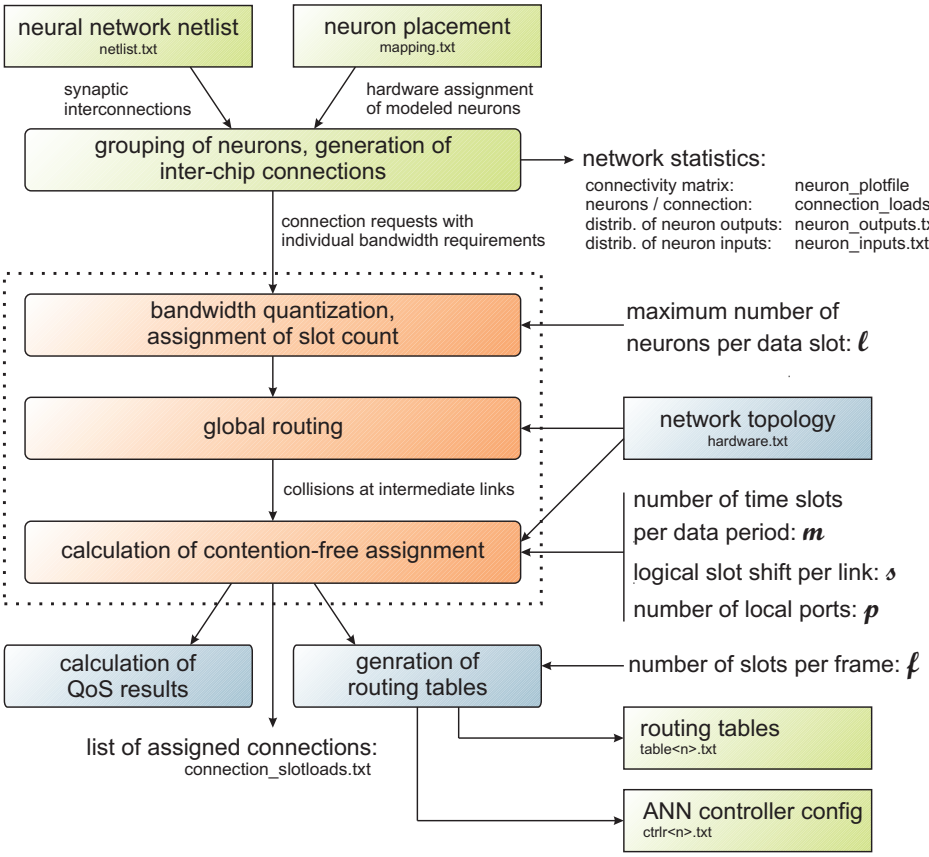



Figure 4.38: Schematic of the data flow of the program `mapping`. The dotted rectangle contains the three main algorithms of the connection mapping process: quantization, routing and contention resolution. The parameters m, l, s and p are denoted via command line.

possible route. After a route has been selected for a connection, the weights of the used links are increased by its number of time slots to balance the routing of succeeding connections. Short-range connections are routed before long-range connections.

*calculation of contention-free assignment*

4. The program calculates the contention-free assignment of the slot positions of each connection at each link. The used algorithm does not differ between fixed framing and shifted framing. Fixed framing is implemented simply by setting the slot shift $s_e = 0$ at all links. The assignment problem is solved by a conversion into the problem of finding a vertex-coloring of a directed graph as described in algorithm 3.6.4 of section 3.6.4. The vertex coloring is implemented with the selection rule from Brelaz [13, 146]. The rule denotes the next vertex to color. It is selected as the vertex with the largest number of different colors of colored adjacent vertices. The maximum number of colors equals the number of time slots per reservation period.

*generation of routing tables*

5. The result of the vertex coloring is transformed back to the initial slot assignment problem. The calculated colors of the vertices represent slot positions at the start nodes of the connections. The slot positions denote the times when

the corresponding application is allowed to transmit data to the network. The slot positions at intermediate links result out of the slot shifts $s_e$ as described in section 3.6.4. The assignment is used to generate the routing tables, a list of connections and a table that assigns single inter-neuron connections to local switch ports. The latter information is required for the high-level software simulation of the setup (cf. section 5.6).

6. The program calculates the resulting QoS delay and jitter values for each connection as well as statistic results about the performance of the slot assignment. This implies the mean values for delay, jitter and the available physical bandwidth calculated over all connections. *calculation of statistical results*

The results of the mapping process are printed to screen and also stored in textfiles. The performance of the connection mapping program mainly depends on the performance of the vertex color algorithm. The algorithm mail fail during the steps 3 or 4. In this case, the user can restart the program with two modifications: First, larger reservation periods for a finer division of the bandwidth into time slots. Second, an increased amount of local ports to reduces contentions at the local ports. The latter requires to synthesize the hardware design with a larger number of local ports. The results for different neural networks are evaluated in section 5.5.

### 4.10.3 Configuration of the Routing Tables

The program `mgtroute` controls two different parts of the digital design: the routing tables within the switches and the functionalities of the demonstrator application for connection-based traffic (cf. section 4.8.2). The routing tables store the input port numbers and output port numbers that have to be connected for each slot $0 \ldots f-1$. The routing tables are generated by the mapping program described in the previous section. The program `mgtroute` is controlled by command-line parameters. In particular, the program can be used for the following tasks: *features*

1. To modify and list single table entries.
2. To modify and list whole routes between distant switches.
3. To load the complete set of routing tables generated by the mapping program.
4. To access the demonstrator application for connection-based traffic, e.g. to start synchronous transfers by using the GSS and to verify the content of the transmitted data within the receive buffers.

After the routing tables have been configured, `mgtroute` can be used to verify the transmission of data within isochronous connections. The program writes random test-data into the transmit buffers of the demonstrator application and initiates the synchronous transfer of data within all connections by raising a GSS at the node 0. It verifies the appearance of the transmitted data at the correct addresses within the receive-buffers at the destination nodes. The test checks the correct transfer with the precision of a global clock cycle of 6.4 ns. The program finally prints the calculated error rates for the physical transmissions.

### 4.10.4 Generation of Pseudo-Random Networks

*generation of test-networks*

The performance of the connection mapping algorithm has been evaluated using a set of pseudo-random neural networks. The networks have been created with the script `generate_networks.py` written by Dr. Johannes Fieres. The script is able to use all available physical neurons and synapses for the generation of neural networks of different topologies. The term *pseudo-random* means that the available connectivity-resources of the chips (e.g. the available inputs of the network blocks) are divided at random over the neurons.

*features*

The script has the following configurable features:

1. A configurable number of chips, neurons per chips, synapses and inputs per network block.
2. A configurable physical topology. This also allows the creation of feed-forward networks in the case that the available inter-chip connections are denoted to the script accordingly.
3. An option to control the synaptic connectivity between the neurons depending on the physical hop-distance of the transport network between the chips. This is used to model a distance-dependent connectivity and to reduce the network load.

The generated networks are stored into two files that specify the resulting netlist (`netlist.txt`) together with a valid neuron mapping of the netlist to the ANN hardware (`mapping.txt`).

#### Description of the Generation Algorithm

*input-count limitation*

As discussed in section 2.3.3, the limited input count (the number of synapse drivers) of the network blocks of the ANN chips hinders arbitrary networks to be mapped to the hardware without losses of neurons or synapses. In particular, fully connected networks or randomly connected networks can be implemented only within a single network block. The algorithm of the script therefore keeps to the limited input count and generates networks with a reduced connectivity that can be mapped to the given ANN hardware without losses.

*distribution of inputs*

The algorithm is illustrated in figure 4.39. It divides the available inputs of each block in multiple groups. Each group belongs to a particular chip. The number of groups and their size depend on the selected network topology. The inter-neuron connections between the ANN chips are generated with respect to the groups: For each input, a neuron of the corresponding ANN chip is selected at random as its source neuron. The destination neurons of the input can be all neurons of the corresponding network block. Inter-neuron connections between the source neuron of the input and local neurons of the chip are established by enabling the synapses of the corresponding synapse driver. The number of synapses to be enabled (i.e. the number of destination neurons) depends on the desired hardware efficiency and connectivity.

*randomness against hardware efficiency*

The resulting networks have a pseudo-random character, since the source neurons and the destination neurons are selected at random out of the possible ones. Furthermore, the grouping of inputs is done equally among the possible chips. Although

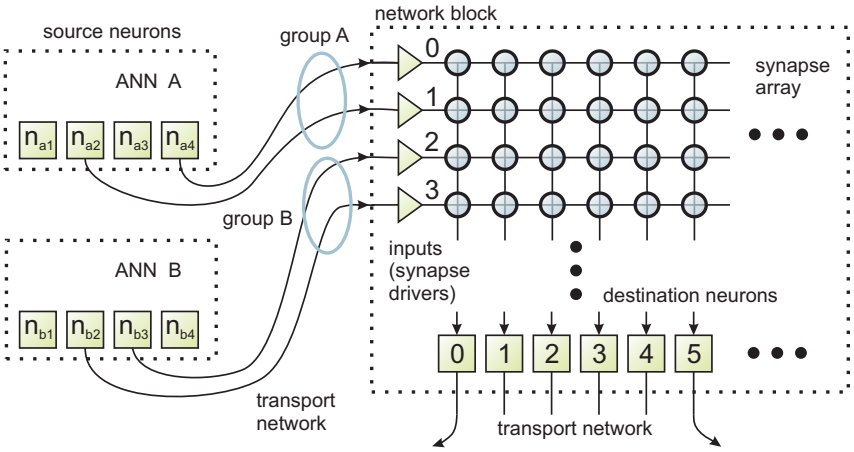



Figure 4.39: Generation of pseudo-random networks. The inputs are divided into groups and assigned to the ANN chips. The particular neurons at the source ANN and at the destination ANN are selected at random. All source neurons of a block share its local destination neurons depending on the fraction of the local synapses to be used.

the algorithm allows to use up to 100 % of all neurons and all synapses, the user has to trade the resulting randomness against the hardware efficiency: The usage of all available synapses results in the fact that all neurons that transmit data to the same network block share the same destination neurons. Furthermore, the possible numbers of destination neurons is reduced to multiples of the neurons per network block.

*pseudo-random, 16 Spikeys*

**Example** Consider a backplane fully equipped with 16 `Spikey` ANN chips. The backplane provides the total of 6144 neurons and 1572864 synapses within 32 network blocks. To use all available hardware resources, the 256 inputs of each network block are divided into 16 groups according to the 16 ANN chips. Each group is assigned $256/16 = 16$ source neurons from its corresponding ANN. Since all synapses are activated, the data from each input is forwarded to all local neurons. The resulting network requires $16 \cdot 15 = 240$ inter-chip connections to be implemented within isochronous connections. Each connection transports the data of $\leq 2 \cdot 16$ neurons to the two network blocks on the destination chip (the probability that any source neuron is selected at random for *both* network blocks on the destination chip is small). The inter-neuron connectivity matrix of the resulting network is shown in figure 4.40.

### 4.10.5 High-Level Simulation of the Packet-Switch incl. Scheduler

The software `switchtest` simulates an input-queued packet-switch. It has been developed in conjunction with the digital design of the bypass-switch. The switch simulator features a large set of configuration parameters and can be used as a universal tool to model input-buffered packet switches with VOQs. Although there exist a number of software simulators for computer networks and switches (see e.g. [2, 132, 102]), the switching technology developed in this thesis required an individual solution.

The switch simulator has been used for two different purposes during the devel- *purpose*

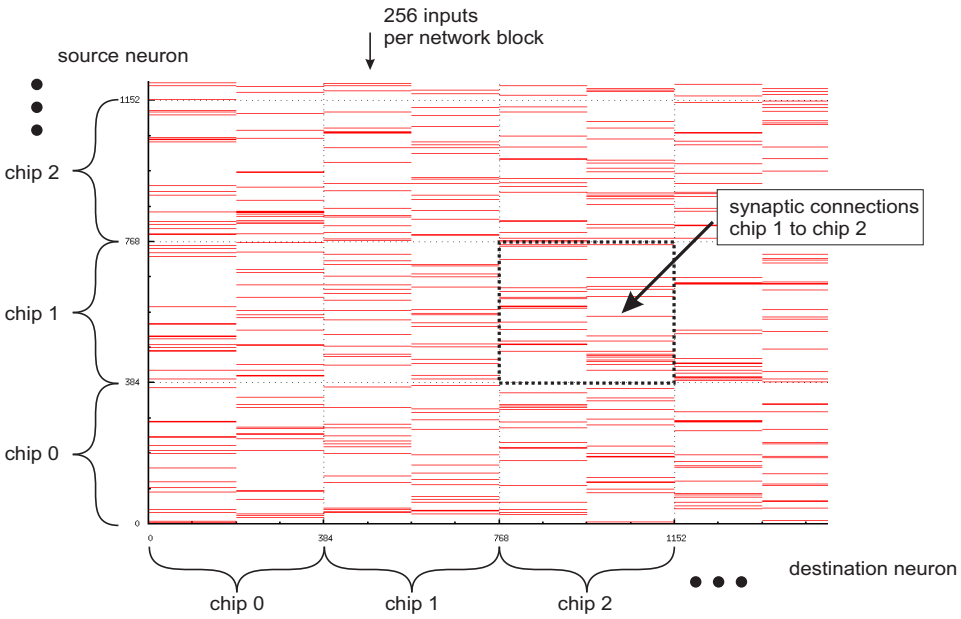



Figure 4.40: Extract of the inter-neuron connectivity matrix of a generated pseudo-random network of 16 `Spikey` ANN chips with the maximum possible number of synaptic connections. Each inter-chip connection transports the data of up to 32 neurons.

opment of the transport network:

- The simulation of the performance of the bypass-switch under different traffic conditions, different schedulers and different configurations with focus on the result of the best-effort packet scheduler and the required sizes of the input-queues. This helped to select the scheduler to be implemented in the digital design that is best adopted to the problem.
- The verification of the correctness of the implemented digital design during its development against the existing software implementations.

The software is written in the object-oriented language C++. An overview of the simulated switch is shown in figure 4.41. The simulator has the following features:

*input-queued switch*

- The architecture of the modeled switch is the bypass-switch described in section 3.8.3. The modeled parts of the switch have been reduced to the parts necessary to evaluate the performance of the best-effort scheduler. That is, the software models the arrival process of the packets, the storing of the packets into the VOQs, the appearance of reserved slots, the queuing bypasses and the central scheduler.

*slot-based timing*

- The timing of the simulation is based on consecutive time slots to model the MCGN framing scheme. The grouping of slots to time frames is not modeled since this is invisible to the best-effort scheduler of the bypass-switch. Packets can be placed in a single slot or also be extended over multiple slots. Packets from different inputs are not merged within consecutive time slots according to the MCGN specification.

- The packet are transported as best-effort traffic in unreserved slots. The arrival patterns of both traffic classes can be configured: best-effort packets are generated with uniform Bernoulli i.i.d. arrivals. Reserved traffic can be configured with different periodic reservation patterns. The interaction between both classes is controlled by enabling or disabling the queuing bypass. *reserved and best-effort traffic*
- The software description of the evaluated best-effort scheduler is accessed via a universal object-oriented interface. This allows to easily investigate further scheduler types and also queuing policies for their feasibility for the digital design. *object-oriented scheduler interface*
- A large set of parameters allows a detailed modeling of the incoming traffic (packet sizes, burst modes), the architecture of the switch (queue sizes, port numbers, enabling of the queuing bypass) or the behavior of the schedulers (types, iterations, duration of the calculation). The same architecture-specific parameters are also available to configure the digital design. *large set of parameters*

The advantage of the developed simulator compared to other simulators is its rich set of configuration parameters. The bypass-switch is only a sub-case of the possible configurations. As an example, a common (slot-based) input-buffered packet-switch with VOQs is modeled by the simulator by disabling reserved traffic and by reducing the packet-size to a single slot.

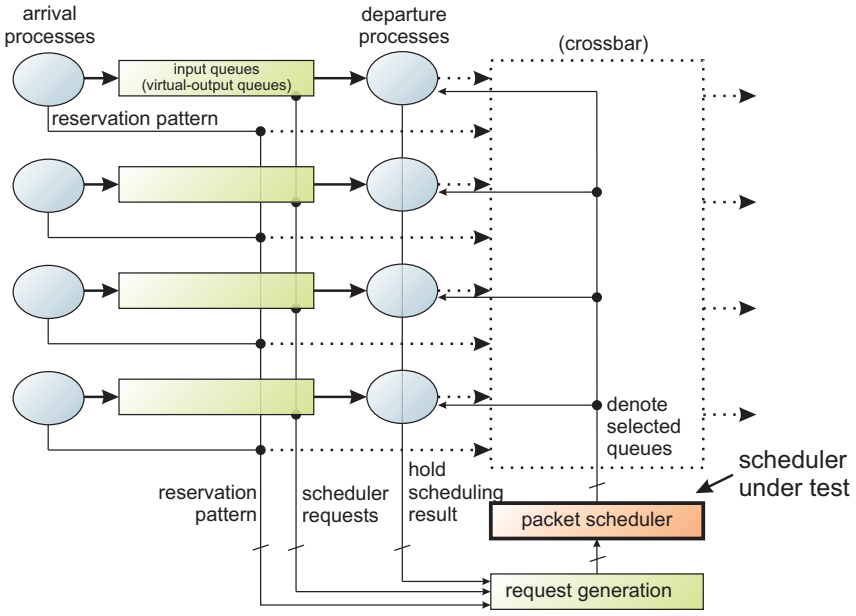



Figure 4.41: Block-level schematic of the developed software for the simulation of the behavior of an input-queued packet-witch. The data path is reduced to the storing of the packets into the queues and its removal from the queues (thick lines). Dotted parts illustrate the functionality of the bypass-switch, but are not existent within the simulation. Thin lines denote control signals (requests, selects, etc.). Crossed lines demote a bus of multiple signals.

### Functional Description

The switch simulator models the number of packets within the queues over time *modeled parts*

according to the scheduler results. Since no data is transferred, the crossbar and the output ports of the switch are effectively not modeled. The result of the scheduler is taken to denote the affected queues which packets are to be taken out. The simulator does not distinct the ports between local ports or global ports, but handles each port identically.

*data flow*

The switch operates in a slotted timing. Incoming traffic is generated by an independent arrival process at each port. The arrival process creates packets with the configured number of slots with a given network load (arrival probability per time slot). The packets are then stored into the queues or dropped depending on the queue occupation. Queues that contain packets request the central best-effort scheduler. The scheduling is executed each time slot. Ports that are marked as reserved in the following slot are excluded from the scheduling. The scheduling result is denoted to the departure processes, which remove the packets from selected queues slot by slot. According to the framing scheme of the bypass-switch, the requests to the scheduler from the queues keep to the currently departing packet until it is transmitted completely. To evaluate the performance of the scheduler, the simulator collects statistical information like the number of stored or dropped packages or the average packet delay during runtime.

**Usage**

*command-line usage*

The switch simulator is controlled via command-line parameters. After been executed, it simulates the operation of the switch for a given number of time slots and logs the number of stored and dropped packets. It finally calculates statistical results, namely the mean packet delay, the delay deviation and the throughput. The results are printed to screen and logged to a file.

*verification of the digital design*

To verify the digital design of the switch against the software simulator, the simulator can be configured to log the packet arrival patterns and the calculated departure patterns in a text-file. This file is then interpreted by the testbench of the digital design.The performance results of the different simulated schedulers are discussed in detail in section 5.7.

## 4.11 Summary

*implemented parts*

This chapter presented the implementation of the transport network for the research with hardware neural networks on the Stage 1 framework. The implementation consists of functional block of a digital design described in VHDL as well as of C/C++ software. The transport network uses the MCGN network architecture for its lower network layers to provide isochronous connections and packets, both transmitted via the gigabit network of the framework. It has been shown, how the FPGA-internal MGTs are used together with a global reference clock to establish a framework-wide synchronization. The reference implementation of the bypass-switch has been described. Two schedulers have been exemplarily implemented: The iSLIP scheduler and variations of 2-dimensional crossbar schedulers. The network initialization phase is executed with additional control software: to control the synchronization, to calculate the contention-free slot assignment and to access the routing tables of the switches.

*isochronous connections and packets*

Since the network interface of the ANN controllers is not yet finished, a demonstrator application has been developed to evaluate the suitability of the provided isochronous connections. The application performs the synchronized transmission of pseudo-events via isochronous connections and can be used as a starting point for further development. The implementation of a DSM has been presented. The DSM uses the best-effort packets of the underlying MCGN network for a framework-wide transport of SDRAM-memory data. A simple ARQ protocol has been implemented to ensure stability and data reliability.

*universality*

The implementation of the transport network has been made towards a low consumption of programmable logic. This is further eased due to the MCGN concept, which moves the complexity of the contention-free routing from online to offline. Both parts, the digital design and the software, are modular and hierarchic. The communication between both parts use a standardized interface, the SlowControl of the framework. Due to this, the transport network is a general tool. It is assumed that the developed network can be applied to other applications that feature programmable logic and software with only minor modifications.

# Chapter 5

# Evaluation

*This chapter describes the evaluation of the implemented transport network, including the reference implementation of the MCGN architecture for its lower network layers. It is shown that the implemented network is well suited for the research with hardware neural networks within the FACETS Stage 1 framework of the Electronic Vision(s) group. The chapter first evaluates the performance of the lower network layers, namely the reliability of the physical data, the transmission delays and the stability of the global synchronization. The suitability of the transport network for the interconnection of ANN chips is shown by the evaluation of the provided bandwidth depending on different neural network topologies. The isochronicity of the provided connections is verified by cycle-accurate online measurements. Finally, simulation results are shown to discuss the performance of the implemented best-effort schedulers.*

## 5.1 Evaluation of the Physical Layer

The physical layer of the transport network consists of the backplane hardware and the MGTs, which are embedded within the FPGAs. Concerning the digital design, the implementation of the physical layer is reduced to the configuration of the MGTs. The configuration is required to maximize the reliability of the transmitted data.

### 5.1.1 Measurement of the Data Reliability

*physical dependencies*

The data reliability of the MGT transmissions depends on the qualities of the external signals, which itself depends on the physical environment such as the jitter quality of the oscillator of the global reference clock, the power stability, the type of the connectors or the routing of the differential signal traces (see e.g. chapter 3 of [157]). The physical properties of the backplane and the `Nathan` network modules are described in [46].

*measurement*

The MGTs can be adjusted to the physical environment with two parameters: the pre-emphasis (10 % to 33 %) and the differential voltage swing (400 mV to 800 mV) of the transmitter circuit. The influence of both parameters has been tested with a digital design that contains a plug-able module for the dynamic re-configuration of the MGT parameters at runtime [26]. The test-design constantly transmits packets of 64 byte at all MGT outputs. Each packets contains pseudo-random patterns generated with a LFSR [109] plus the MGT-internal 32-bit CRC checksum. The data integrity of the transmissions is monitored at the destination by verifying the CRCs and counting the transmission errors.

*illustration of results*

The results of the measurements are illustrated in the figures 5.1 and 5.2. The measured values denote error-rates per 64-byte packet. Each diagram has four parameters: the `Nathan` locations, MGT location as well as the selected voltage level and pre-emphasis. The MGT locations `SW`,`SE`,`NW`,`NE` correspond to the center bottom and top MGTs of the `xc2vp7` FPGA, whose external pins are routed to the backplane connector (cf. section 4.3.2 or [46]).

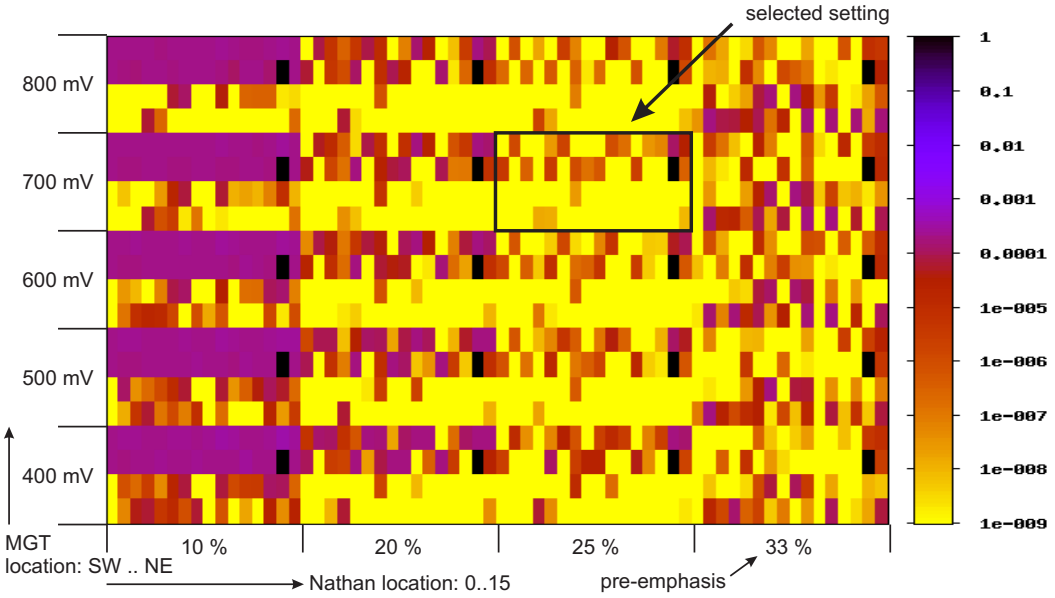



Figure 5.1: Measurement of the MGT error rates for packets of 64 byte sorted by electrical parameters. The parameters 700 mV and 25 % pre-emphasis have been selected for all MGTs.

*resulting error rates*

It can be seen from the diagrams that the two MGTs at the bottom of the FPGA feature lower error rates. This is since the bottom MGTs are clocked by the `brefclk` with lower jitter and have shorter trace lines on the `Nathan` modules. There is no set of parameters that results in error-free transmissions. The best results are achieved for the parameters 700 mV and 25 % pre-emphasis. This values have been set for all MGTs as the standard configuration parameters for all digital designs (cf. also, table 4.2). For this setting, the packet error probabilities are $10^{-5}$ to $10^{-9}$, which correspond to a slot error probability of $6.3 \cdot 10^{-7}$ to $6.3 \cdot 10^{-11}$ and bit error

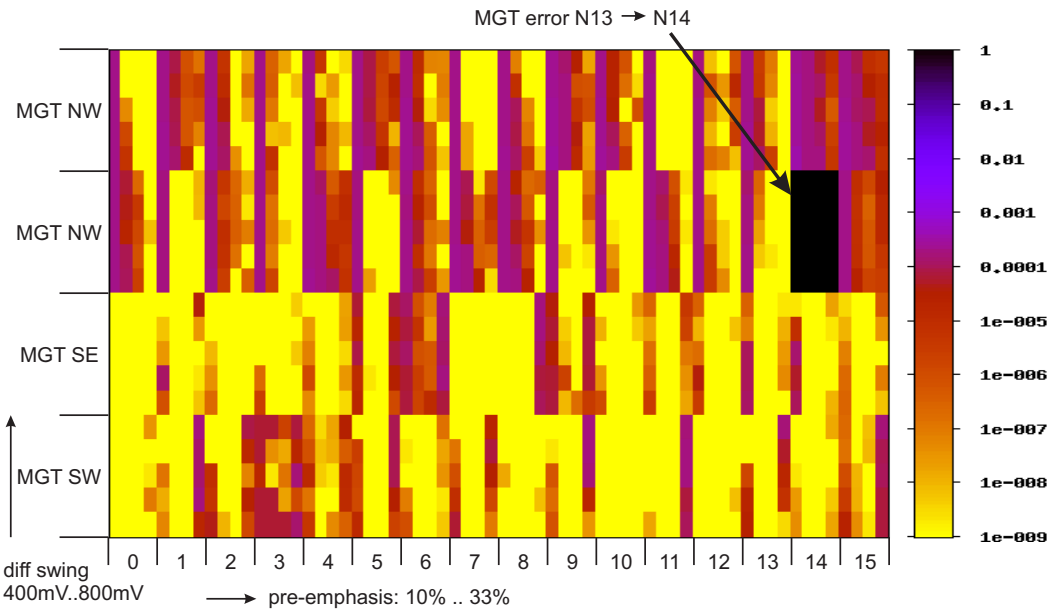



Figure 5.2: Measurement of the MGT error rates for packets of 64 byte sorted by physical locations. The two MGTs at the bottom of the network modules feature lower error rates. The lower MGTs use the `brefclk` with lower jitter and have shorter signal traces on the `Nathan` network modules.

probabilities[1] of $2 \cdot 10^{-8}$ to $2 \cdot 10^{-12}$. The mean-times between errors result in 16 ms to 163 s at the selected line rate of 3.125 Gbit/s.

## 5.2 Evaluation of the Synchronization Sublayer

The purpose of the synchronization sublayer is to ensure a deterministic timing of the framework. During the synchronization process, the internal time counters of each node are synchronized and the number $f$ of slots per frame, the logical slot shift $s_e$ of each link $e$ and the resulting transmission delays $D_e$ between the inputs of adjacent switches are fixed. Due to the regular topology of the backplane, the synchronization is established with the same values $s_e = s$ and $D_e = D$ on all links. The following sections evaluate the resulting values for $s$ and $D$ and measure the synchronization stability.

The tests haven been performed with a fully equipped backplane with 16 `Nathan` modules. The digital design within the FPGAs comprises the synchronization module `phys_sync`, a single isochronous switch (without best-effort support) and the demonstrator application for isochronous connections of section 4.8.2.

### 5.2.1 Measurement of the Transmission Delays

*setup*

In a first step, the minimum possible values $D_0$ of the physical transmission delays between adjacent switches are measured and compared to the simulated values. This

[1] In fact, the exact bit error rate (BER) is unknown do to the 8b/10b encoding of the transceivers. The denoted error rates correspond to the user data.

is done with the software `mgtsync` of section 4.10.1. The measurement is done by monitoring the local arrival times of incoming frames with the setting $\epsilon$=0, $s_e$=0 and by applying the minimum-delay-patch for the MGTs. The resulting delays are calculated by using equation 3.11.

*delay results*

Table 5.1 shows the measured results. It can be seen that the measured transmission delays vary between 22 and 23 clock cycles. Non-integer values denote measured delays of an even and also an uneven number of cycles in both directions of a bidirectional link. The synchronization sublayer features an internal delay of three clock cycles. The measured delays of the physical layer (the data path from MGT to MGT) are therefore

$$D_0 = 19 \ldots 20 \text{ cycles}. \tag{5.1}$$

The simulation of the digital design with the Modelsim [94] simulator resulted in 20 cycles for the delay-reduced MGTs (cf. 4.4.4). Both values are in very good accordance.

| Nathan | MGT_SW | | MGT_SE | | MGT_NW | | MGT_NE | |
|---|---|---|---|---|---|---|---|---|
| module | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ |
| 0 | 22/ 22 | 22.0 | 23/ 22 | 22.5 | 23/ 23 | 23.0 | 11/ 35 | 23.0 |
| 1 | 22/ 22 | 22.0 | 22/ 22 | 22.0 | 23/ 23 | 23.0 | 17/ 29 | 23.0 |
| 2 | 22/ 22 | 22.0 | 29/ 17 | 23.0 | 23/ 23 | 23.0 | 23/ 22 | 22.5 |
| 3 | 22/ 22 | 22.0 | 29/ 16 | 22.5 | 23/ 23 | 23.0 | 23/ 23 | 23.0 |
| 4 | 22/ 22 | 22.0 | 29/ 16 | 22.5 | 11/ 34 | 22.5 | 23/ 23 | 23.0 |
| 5 | 22/ 22 | 22.0 | 11/ 35 | 23.0 | 17/ 28 | 22.5 | 22/ 23 | 22.5 |
| 6 | 16/ 29 | 22.5 | 17/ 28 | 22.5 | 28/ 17 | 22.5 | 29/ 17 | 23.0 |
| 7 | 29/ 16 | 22.5 | 23/ 22 | 22.5 | 34/ 11 | 22.5 | 35/ 11 | 23.0 |
| 8 | 24/ 22 | 23.0 | 22/ 23 | 22.5 | 41/ 5 | 23.0 | 35/ 10 | 22.5 |
| 9 | 22/ 24 | 23.0 | 28/ 17 | 22.5 | 21/ 24 | 22.5 | 33/ 11 | 22.0 |
| 10 | 40/ 4 | 22.0 | 35/ 11 | 23.0 | 24/ 21 | 22.5 | 41/ 4 | 22.5 |
| 11 | 4/ 40 | 22.0 | 16/ 29 | 22.5 | 5/ 41 | 23.0 | 23/ 22 | 22.5 |
| 12 | 22/ 23 | 22.5 | 16/ 29 | 22.5 | 17/ 29 | 23.0 | 22/ 23 | 22.5 |
| 13 | 23/ 22 | 22.5 | 17/ 29 | 23.0 | HWERR | | 4/ 41 | 22.5 |
| 14 | 23/ 21 | 22.0 | 22/ 22 | 22.0 | HWERR | | 11/ 33 | 22.0 |
| 15 | 21/ 23 | 22.0 | 22/ 23 | 22.5 | 29/ 17 | 23.0 | 10/ 35 | 22.5 |

Table 5.1: Example measurement of the minimum transmission delays $D_0$ of all 64 links of a fully connected backplane. The delays vary due to the external sampling of the distributed reference clock and the data alignment of the serial transceivers. One link between the modules 13 and 14 is measured to be defect.

### 5.2.2 Establishment of the Synchronization

The synchronization is established with the software `mgtsync` of section 4.10.1. This has been tested for multiple different combinations of frame sizes $f$ and the corresponding slot shifts $s_e$. Since the measured values of $D_0$ correspond to the simulated values of $D_0$, the possible combinations of $f$ and $s_e$ equal the values listed in table 4.4.

*successful synchronization*

The synchronization has been established successfully. Table 5.2 exemplarily lists the output of the program `mgtsync` after a synchronization with $f$=11 slots per frame

(22 cycles) and $s_e$=0. An output with similar values is achieved with the settings $f$=60 slots and $s_e$=12 slots (with the command-line parameters $f$=120 cycles and $s$=24 cycles), which results in the same values for the delay elements. It can be seen that most of the delay elements are set to the value 1 as expected from table 4.4. Since the synchronization fixes the transmission delays $D$ on all links to the same value, a repeated measurement of the transmission delays (with the same number $f$ and unchanged delay elements $\epsilon$) after the synchronization results in measured timer values of $2 \cdot f + 2$ cycles and a calculated time shift of 0 on *all* links as expected (cf. table 5.3).

| Nathan | MGT_SW | | MGT_SE | | MGT_NW | | MGT_NE | |
|---|---|---|---|---|---|---|---|---|
| module $A$ | $B$: state | $\epsilon$ | $B$: state | $\epsilon$ | $B$: state | $\epsilon$ | $B$: state | $\epsilon$ |
| 0 | 1: sync | 1 | 15: sync | 1 | 3: sync | 1 | 7: sync | 1 |
| 1 | 0: sync | 1 | 14: sync | 1 | 2: sync | 1 | 6: sync | 1 |
| 2 | 3: sync | 2 | 13: sync | 1 | 1: sync | 2 | 5: sync | 1 |
| 3 | 2: sync | 0 | 12: sync | 1 | 0: sync | 1 | 4: sync | 1 |
| 4 | 5: sync | 0 | 11: sync | 1 | 7: sync | 1 | 3: sync | 1 |
| 5 | 4: sync | 2 | 10: sync | 1 | 6: sync | 2 | 2: sync | 1 |
| 6 | 7: sync | 1 | 9: sync | 1 | 5: sync | 0 | 1: sync | 1 |
| 7 | 6: sync | 2 | 8: sync | 2 | 4: sync | 1 | 0: sync | 1 |
| 8 | 9: sync | 1 | 7: sync | 2 | 11: sync | 2 | 15: sync | 1 |
| 9 | 8: sync | 2 | 6: sync | 1 | 10: sync | 1 | 14: sync | 1 |
| 10 | 11: sync | 3 | 5: sync | 1 | 9: sync | 2 | 13: sync | 1 |
| 11 | 10: sync | 0 | 4: sync | 1 | 8: sync | 1 | 12: sync | 1 |
| 12 | 13: sync | 1 | 3: sync | 2 | 15: sync | 1 | 11: sync | 1 |
| 13 | 12: sync | 2 | 2: sync | 1 | 14: sync | 2 | 10: sync | 2 |
| 14 | 15: sync | 1 | 1: sync | 2 | 13: HWERR | 0 | 9: sync | 2 |
| 15 | 14: sync | 2 | 0: sync | 1 | 12: sync | 1 | 8: sync | 1 |

Table 5.2: Synchronization of 16 network modules on a single backplane with $f = 11$ slots per frame. All modules are synchronized to a single clock cycle of 156.25 MHz. The adjustable delay elements in the data path are set to values of 0 to 3 cycles. The forwarding of the data slots is deterministic between the nodes to the precision of the synchronization. The link failure from of 13 to node 14 is ignored during the synchronization.

### Synchronization Stability

The stability of the synchronization is constantly checked within the `phys_sync` module of the digital design by monitoring the time of the appearance of the `SYNC` character. Due to the usage of the unique global clock reference, the synchronization remained stable for several days up to weeks (until the test has been aborted). Since the duration of single neural network experiments is typically in the range of seconds up to minutes, the stability of the synchronization is assumed to be sufficient.

*days to weeks*

| Nathan | MGT_SW | | MGT_SE | | MGT_NW | | MGT_NE | |
|---|---|---|---|---|---|---|---|---|
| module | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ | $A_F/B_F$ | $D_0$ |
| 0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 1 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 2 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 3 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 4 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 5 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 6 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 7 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 8 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 9 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 10 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 11 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 12 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |
| 13 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | HWERR | | 24/ 24 | 0.0 |
| 14 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | HWERR | | 24/ 24 | 0.0 |
| 15 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 | 24/ 24 | 0.0 |

Table 5.3: Repeated measurement of the transmission delays $D$ after synchronization. All nodes of the backplane have been synchronized and the delay is exactly 24 clock cycles between all adjacent nodes.

## 5.3 Verification of the Transport of Isochronous Data

Neural network data between ANN chips is transported via isochronous connections. The characteristic feature of the isochronous connections is the guaranteed QoS. It has already been shown that the QoS features in terms of throughput, delay and jitter are guaranteed by design, i.e. by the deterministic timing of the synchronization sublayer and the reservation of slots. The particular values depend on the framing parameters and on the efficiency of the mapping algorithm.

The following paragraphs evaluate the performance of the isochronous connections from the application layer. Since the throughput, the delay and the jitter rely on the deterministic timing of the network, the investigated aspects are the correct timing and the data reliability. The tests have been made with a digital design that contains the demonstrator application for isochronous connections. The measurements use the software `mgtroute` of section 4.10.3.

### 5.3.1 Measurement of Application-Layer Delays

*principle of the measurement*

The measurement of the transmission delays has been done using a bi-directional route between two network nodes as illustrated in figure 5.3. The application at one network node (A) is selected to be the sending application, the second end-point (B) is configured as a reflector. The reflection can be implemented either on the data link layer (by configuring the routing tables) or at the application layer (by configuring the application at node B to reflect all incoming data). The application A transmits a dedicated data value at the connection interface and counts the delay

until its arrival in multiples of clock cycles. The measurement is not affected by the wait-jitter at the interface since the counter is started at the time the data is transmitted into the connection.

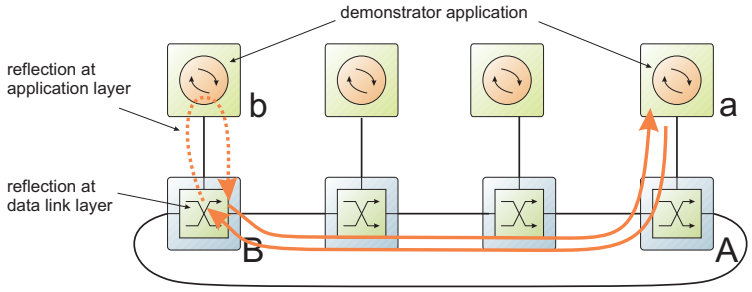



Figure 5.3: Measurement of the delays of isochronous connections. Reflections can be implemented at the data link layer (by configuring the forwarding tables) or at the application layer (by reflecting the incoming data). The illustrated example examines a bi-directional connection of three hops.

*theoretical values*

The theoretical value for the transmission delays $D_c$ of a connection $c$ calculates out of the transmission delays $D$ between the switches according to equation 3.22. This delay is increased due to the implementation within the programmable logic for the registered transmission and the registered reception of the data at the connection interface of the switch, which requires a single cycle each. By denoting $h$ as the intermediate number of network hops, the theoretical delay therefore calculates to:

$$D_c = 2 \cdot h \cdot D + 4 \text{ cycles} \tag{5.2}$$

for the reflection at the application layer and

$$D_c = 2 \cdot h \cdot D + 2 \text{ cycles} \tag{5.3}$$

for the reflection at the data link layer.

*effect of synchronization technique*

The values for $D$ slightly depend on the framing technique. For fixed framing ($s$=0), the values $D$ are constant at all links and for all time slots. For shifted framing, the values depend on the position $i$ of the slot at the link: The delay $D_<$ for $0 < i \leq f - s$ and the delay $D_>$ for $f - s < i \leq f$ differ by the size of the gap (cf. section 3.7.3). If multiple hops are passed, the slot position $i$ is shifted by $s$ at each link such that the particular hop delays may be $D_<$ and also $D_>$ on different hops. Connections that occupy a single slot therefore have a constant transmission delay (by ignoring the wait time at the interface).

*measured results*

The connection delays have been measured for different round-trips in the network with a different number of network hops. The measurements have been performed with a synchronized network for three sets of framing parameters $f$, $s$ and $D$. All transmissions use the start slot $q_c = 0$. Table 5.4 lists the measured round-trip delays. The measured values match exactly the theoretical ones from the equations 5.2 and 5.3 (at the last set of parameters, both values $D_<$ and $D_>$ have to be considered). Furthermore, the same values have also been verified within the cycle-accurate simulation of the digital design with the `Modelsim` [94] simulator. This shows that the transport network has successfully been implemented to provide isochronous connections with a deterministic timing to the application layer.

| f | s | D | route | hops | reflection | cycles | time | per hop |
|---|---|---|---|---|---|---|---|---|
| 11 | 0 | 24 | (0) | 0 | data link | 2 | 12.8 ns | - |
| 11 | 0 | 24 | (0,1,0) | 2 | data link | 50 | 320.0 ns | 160.0 ns |
| 11 | 0 | 24 | (0,1,1,0) | 2 | application | 52 | 332.8 ns | 166.4 ns |
| 11 | 0 | 24 | (0,1,2,2,1,0) | 4 | application | 100 | 640.0 ns | 160.0 ns |
| 11 | 0 | 24 | (0,..,3,3,..,0) | 6 | application | 148 | 947.2 ns | 157.9 ns |
| 11 | 0 | 24 | (0,..,4,4,..,0) | 8 | application | 196 | 1254.4 ns | 156.8 ns |
| 12 | 0 | 26 | (0,1,0) | 2 | data link | 54 | 345.6 ns | 172.8 ns |
| 12 | 0 | 26 | (0,1,1,0) | 2 | application | 56 | 358.4 ns | 179.2 ns |
| 12 | 0 | 26 | (0,1,2,2,1,0) | 4 | application | 108 | 691.2 ns | 172.8 ns |
| 12 | 0 | 26 | (0,..,3,3,..,0) | 6 | application | 160 | 1024.0 ns | 170.7 ns |
| 12 | 0 | 26 | (0,..,4,4,..,0) | 8 | application | 212 | 1356.8 ns | 169.6 ns |
| 60 | 12 | 24/26 | (0,1,0) | 2 | data link | 50 | 320.0 ns | 160.0 ns |
| 60 | 12 | 24/26 | (0,1,1,0) | 2 | application | 52 | 332.8 ns | 166.4 ns |
| 60 | 12 | 24/26 | (0,1,2,2,1,0) | 4 | application | 100 | 640.0 ns | 160.0 ns |
| 60 | 12 | 24/26 | (0,..,3,3,..,0) | 6 | application | 150 | 960.0 ns | 160.0 ns |
| 60 | 12 | 24/26 | (0,..,4,4,..,0) | 8 | application | 198 | 1267.2 ns | 158.4 ns |

Table 5.4: Measurement of the round-trip delays of isochronous connections for three different sets of framing parameters. All transmissions use the start slot 0. The measured values match exactly the calculated and the simulated values.

#### Single-Hop Delay

The last column of table 5.4 also lists the mean per-hop delays for each connection. The values show that the typical per-hop delay between the applications of two adjacent network nodes that transmit data within isochronous connections results to:

$$D_{typical} = 160 \text{ ns} \ldots 180 \text{ ns}. \tag{5.4}$$

It is worth to mention that 20 clock cycles (128 ns $\approx$ 75 %) of this value are caused by the MGTs of the FPGAs, even after the reduction of its internal delays. The delay of the remaining parts of the transport network (including the switches, the synchronization logic and the interfaces at both network nodes) has been reduced to only 5 to 7 clock cycles per hop!

### 5.3.2 Verification of Isochronous Transfers

A next step is to simulate the execution of a neural network experiment to verify the parallel transmissions of the isochronous data between all network modules. The part of the ANN controller is performed by the demonstrator application for isochronous connections. The test verifies the following features:

- The synchronous start of the transmissions.
- The correct timing of the arrival of the isochronous data.

- The data integrity.

Table 5.5 summarizes the configuration of the experiment. The network is first synchronized using the program `mgtsync`. After that, a neural netlist is generated that covers all 16 Nathan network modules. The network is mapped to the hardware and the routing tables of the switches are configured accordingly (the single broken link from node 13 to node 14 is left unused). The high-level control of the test is performed with the program `mgtroute`.

*test procedure*

The program first fills the transmit buffers of all local ports on all network modules with random data. It then initiates the synchronous start of the transmissions within the isochronous connections by raising a GSS at the network node 0. The data is transferred to the destination nodes and stored within the receive buffers at addresses according to the current local times (in terms of cycles). After the data has been transferred, the appearance of all data at the correct addresses is verified by the software. This checks the data reliability as well as the correct timing of the transmissions with the precision of a single clock cycle. Due to the storage according to the timing and the small size of the DPBRAMs, only about 40 data slots per connection are transferred within a single experiment. The described experiment is therefore repeated in 50,000 runs by the software.

| | |
|---|---|
| number of network nodes | 16 |
| framing parameters | 12 slots, no shift |
| mapped network | 240 connections, pseudo-random |
| mapping result | 81.6 % slot efficiency (MGT 13-14 left out) |
| experimental runs | 50,000 experiments, 12 h time |
| transmitted data | 488,750,000 data slots (1.8 GByte) |
| erroneous slots | 12 |
| error rate (32 bit) | $2.5 \cdot 10^{-8}$ |
| lost synchronization | 0 |

Table 5.5: Experimental results for transmissions within isochronous connections. 50,000 Experiments have been run. The synchronization has shown to be absolutely stable. The resulting error rate shows that far most of the transmissions have been performed with a cycle-accurate timing without data errors.

*successful operation*

As a result, all experiments have been successfully executed. The total amount of about 2 GByte of data has been transmitted. The execution time for the experiment was about 12 h. For the main part of the time, the network was not transferring data, but the software was clearing and verifying the contents of the receive buffers via the SlowControl. The synchronization remained stable and no re-synchronization has been necessary between single runs. The verification of the received data resulted in 12 data errors, each within different runs. The remaining 99.98 % of all runs have been executed without any error. This leads to the following conclusions:

- The timing of the isochronous transmissions has been verified to be cycle-accurate and deterministic.

- It has been shown that the synchronization is stable over 12 h with random data to be transferred in 50,000 runs.
- The synchronous start of transmissions using a GSS has been successfully executed 50,000 times without failure.

*calculation of error probability*

The number of the received errors further allows to calculate the mean error rate for the parallel transfer of data within isochronous connections with the used slot efficiency of about 80 %. The 12 errors result in a mean per-slot error probability of:

$$(2.5 \pm 0.7) \cdot 10^{-8} \quad \text{(mean slot error probability)} \tag{5.5}$$

This number can be seen as a mean value that corresponds to the transmissions over all links of the backplane (expect the erroneous one from slot 13 to slot 14). The calculated error value is in accordance with the error rates that are measured with the CRC function of the MGTs of $6.3 \cdot 10^{-7}$ to $6.3 \cdot 10^{-11}$ for the particular links. It has further been shown that the data errors affect certain links more often than others, which are completely error-free. This also corresponds to the measurements of section 5.1.1.

## 5.4 Discussion of the Neural Network Topologies

The previous sections verified the physical properties of the transport network and proved its ability to establish isochronous connections on the Stage 1 framework. This section now quantifies the provided services to give an intention of which neural network topologies (i.e. netlists) can be operated on the hardware. The possible networks are limited basically by three factors:

- The input count of the ANN chips limits the netlists that can be mapped onto the hardware without a loss of neurons during the neuron mapping process.
- The available physical bandwidth of the framework limits the amount of neuron data to be transported between the chips at runtime
- The efficiency of the connection mapping process limits the fraction of the physical bandwidth, which is available for neuron data.

*generated pseudo-random netlists*

The following discussion is reduced to netlists that can be mapped to the hardware without any loss of neurons. The netlists and its corresponding *neuron mapping* are created with the network generator script of section 4.10.4, which generates pseudo-random networks optimized for the hardware of the Stage 1 framework.

*connection mapping of the netlists*

In a next step, the created networks are taken as input for the implemented *connection mapping* software, which maps the bandwidth between the neurons to the physical bandwidth and calculates the contention-free slot assignment. The resulting assignment is analyzed to calculate the mean QoS values of throughput, delay and jitter. Each network topology can therefore be assigned a maximum physical mean rate for its neural data. This section therefore is both, a performance evaluation of the connection mapping process and also a discussion of the possible networks, which can be operated on the Stage 1 framework.

*difference between HAGEN and Spikey*

Concerning the `HAGEN` chip, the required amount of data to be transported is de-

terministic and constant and can be directly calculated out of the network topology. It is therefore possible to calculate a maximum frequency $f_{net}$ for the clocked operation of its network. Concerning the `Spikey` chip, the calculated available physical bandwidth corresponds to a physical mean spike frequency $\nu$ of its single neurons. In contrast to the `HAGEN` chip, the spike frequencies of the neurons of the `Spikey` chip differ in a wide range depending on a number of parameters like the configured synaptic timing parameters, the neuron timing parameters, the network topology or the network stimulation. The resulting values are therefore only an indication about the amount of expected event losses or for the possible speedup $\mu$ at which the ANN chip can be operated.

### 5.4.1 Characterization of Neural Network Topologies

*presumptions*

The characterization of different neural network topologies is made by defining a set of classifiers. The calculation of the described classifiers is implemented within the connection mapping software of section 4.10.2.

#### Classifier 1: Number of Isochronous Connections

*group inter-neuron connections*

The first step of the mapping software is the extraction of the inter-chip connections out of the synaptic inter-neuron connections that are defined by the netlist. An inter-chip connection $c_{AB} \in C$ is created for each combination of two chips A and B for which A has one or multiple neurons that transmit neural data to synaptic inputs of neurons on chip B. The connection $c_{AB}$ transports the aggregated network data of *all* of the corresponding neurons on A to B.

*improved bandwidth efficiency*

The grouping of multiple neurons reduces the administrative overhead for the transport network. Concerning the `Spikey` ANN, its neural data is generated statistically by the spike events of single neurons. It is expected that the grouping of multiple synaptic connections to a single inter-chip connections evens the required data rate by the law of large numbers and thus results in a more efficient bandwidth usage. The optimum bandwidth efficiency is achieved further if all inter-chip connections have to transport roughly the data of the same amounts of neurons.

*example*

The total number $\|C\|$ of inter-chip connections and the distance between the ANN chips to be connected are critical parameters for the connection mapping algorithm. This is because each connection requires at least a single slot to be reserved exclusively for it along the links of its route. As an example, the maximum number of inter-chip connections of a network with $n$ chips is required in the case that each chip receives neural data from at least a single neuron of all other chips.

$$\|C\|_{max} = n(n-1) = O(n^2) \tag{5.6}$$

This is the case for fully connected networks or for the typical pseudo-random networks, which are created by the generator script without further optimizations. To reduce the computational task of the connection mapping algorithm, the number of connections can be reduced by an optimized placing of given netlists or by generating the networks accordingly.

### Classifier 2: Hop Ratios

*definition*

The *hop ratios* are a notation to denote the distribution and the ratios of the number of network hops of the inter-chip connections. This is worth to be considered since the number of network hops of the connections affects the performance of the mapping algorithm and the resulting transmission delay and is also a general classifier of the connectivity of the network. To be more precise, the hop ratios are denoted as a set of numbers separated by colons that correspond to the relative amount of neuron data that has to be transported within connections of a number of hops that correspond to the position of the numbers within the notation. The first number corresponds to 0 hops, the second number to 1-hop connections etc. The fraction of neurons which is interconnected *on-chip* compared to the number that is connected *off-chip* (between the chips) equals the first value divided by the sum of the remaining ones.

*example*

As an example, the hop ratios

$$1:1:0:0:0 \tag{5.7}$$

correspond to a network of which 50 % of the neuron data is transferred on-chip (0 hops) and 50 % of the neuron data is transferred to adjacent nodes (1 hop). No connections with a larger number of hops exist (the last three numbers are all 0). The hop ratios

$$1:2:1:0:0 \tag{5.8}$$

corresponds to a network with 25 % of the connections on-chip, 50 % to adjacent nodes and 25 % of the neuron data is transferred over two hops.

The hop ratios are an indication about the required amount of bandwidth for a given (placed) network. The important aspect is that it is possible to generate networks with different topologies that all use 100 % of the neurons and the synapses, but which have different hop ratios and thus a different bandwidth requirement.

### Classifier 3: Network Load

*definition*

The bandwidth requirement of a netlist that has been mapped to the hardware can be measured by the *network load* $L$, which is defined as the number of neurons that transmit data from or via a physical instance:

*network load of a connection*

- The network load of a connection $c$ between the two ANN chips A and B is defined as the number $\|N_c\|$ of neurons within the set $N_c$ of source neurons of $c$ on A.

  $$L_c = \|N_c\| \tag{5.9}$$

  Since each connection is assigned a dedicated number of data slots by the connection mapping algorithm, the value of $L_c$ determines the mean bandwidth that is available for a single neuron and thus the event drop ratio depending on its mean physical spike frequency.

*network load of a physical link*

- The network load $L_e$ of a physical link $e$ is defined as the number of neurons that transmit data via that link:

  $$L_e = \sum_{c:e\in r_c} L_c, \tag{5.10}$$

where $r_c = \{e_1, e_2, ... : e_i \in E\}$ denotes the route of connection $c$. The particular values for the links $L_e$ can be compared to get a measure of how even the network has been mapped onto the physical hardware.

- Using these definitions, the *total network load* $L_{tot}$ of a configured neural network that has been mapped to the physical ANN hardware can be defined: *total network load*

$$L_{tot} = \sum_e L_e = \sum_c L_c \cdot h_c \tag{5.11}$$

where $h_c = \|r_c\|$ denotes the number of physical links on the global route of connections $c$ between its source ANN chip and its destination ANN chip. The total network load respects the number of network hops of the particular connections. It is thus a measure for how much physical bandwidth a given neural network topology requires and allows to compare the different topologies in terms of their total bandwidth usage. In the case that the neural network topology consists only of chip-internal feedbacks, $C = \emptyset$ and $L_{tot} = 0$. The equality can be seen by

$$\sum_e L_e = \sum_e \sum_{c:e \in r_c} L_c = \sum_c \sum_{e:e \in r_c} L_c = \sum_c L_c \cdot h_c \tag{5.12}$$

*pre-mapping values*

Care has to be taken in the case that the netlist has been placed to the hardware but the routing of the connections has not yet been performed by the connection mapping algorithm. In this case, the routes $r_c$ of the inter-chip connections $c \in C$ are undefined and such are the above definitions of $L_e$ and $L_{tot}$. To have a preliminary characterization of unrouted networks, the total network load can be provisionally defined as $L'_{tot}$ in analog to the above formula using the routes $r'_{AB}$ that equal the shortest path from A to B. Since the mapping algorithm may be forced to select longer routes during the routing process, $L_{tot} \geq L'_{tot}$. The preliminary mean load of the physical links $e \in E$ can then be estimated by

$$\overline{L}'_e = \frac{L'_{tot}}{\|E\|} \tag{5.13}$$

### 5.4.2 Calculations for Neural Networks on the Backplane

The network classifiers presented above allow to quantify different network topologies. The following paragraphs use these qualifiers on neural networks that are mapped onto the hardware of the backplane. The considerations are made for *highly interconnected networks* at which each ANN has at least a single neuron of any ANN as input. The number of inter-chip connections of such a network with $n$ chips is $\|C\| = n(n-1)$ as stated above. Note that this does not require a similar load $L_c$ for the particular connections.

*cubic topology*

It has been shown in section 2.2.3 that the topology of the backplane equals a 4-dimensional binary cube, which can be extended to higher dimensions $d > 4$ by using the additional links on top of the `Nathan` network modules. This knowledge and the above definitions allow pre-calculations to be made for highly interconnected networks without knowing the exact configuration of the neural network to be mapped, which is done next.

**The Hop Ratios for the Cubic Backplane Topology** A $d$-dimensional binary cube contains $n = 2^d$ nodes as well as $\|E\| = d \cdot n$ unidirectional links. Since not all nodes are physically connected, the shortest path between two nodes varies between $h = 1 \ldots d$. The number $n_h$ of network nodes in distance $h$ within the $d$-dimensional binary cubic topology equals:

$$n_h = \binom{d}{h} \quad \text{(number of chips in distance h)} \tag{5.14}$$

For highly interconnected networks, the number of connections with a certain amount of hops equals the number of ANN chips at this hop distance, which are described by equation 5.14. The numbers are therefore denoted as the *intrinsic* hop ratios of the binary cube in the following. As an example, the intrinsic hop ratios of a homogeneous 16-chip pseudo-random network are *1:4:6:4:1*. A highly connected network has far more connections of medium hop-distance than of close-distance or far distance. The fraction of inter-neuron connections that are routed between the chips is 93.8 %.

*total network load*

**The Network Loads for the Cubic Backplane Topology** The knowledge of the intrinsic hop ratios allows to estimate the total network load for a highly connected network by assuming a routing of the connections along the shortest path between its source node and its destination node. The mean load per connection is abbreviated as $\overline{L}_c = l$.

$$\begin{aligned} L'_{tot} &= \sum_c l \cdot h_c = ln \sum_{h=1}^{d} \binom{d}{h} h \\ &= ln \sum_{h=1}^{d} \frac{d!h}{h!(d-h)!} = lnd \sum_{h=1}^{d} \binom{d-1}{h-1} \\ &= ln \sum_{h'=0}^{d-1} \binom{d-1}{h'} = ln2^{d-1} = \frac{1}{2} ldn^2 = O(n^2) \end{aligned} \tag{5.15}$$

*mean link load*

The estimated mean load per link therefore equals

$$\overline{L}'_e = \frac{L'_{tot}}{dn} = \frac{1}{2} ln = O(n). \tag{5.16}$$

Since the available bandwidth of the links is limited, an increased number of network nodes therefore forces either to reduce the mean load per inter-chip connection according to equation 5.16 or to reduce the number of connections between the chips. To avoid a connection from a dedicated chip A to a chip B, no neuron on A is allowed to be connected to any neuron on chip B.

### 5.4.3 Homogeneous Pseudo-Random Networks

The above results are now used to discuss the values for pseudo-random networks that are mapped on the backplane, i.e. for which the physical locations of the modeled neurons have been assigned. The networks are generated with the network

generator script from section 4.10.4. The script generates map-able networks by dividing the available inputs of a particular network block in equally-sized groups among the inter-chip connections that have the network block as their destination. The considered topologies are binary sub-cubes of the dimension $d$ mapped on the backplane.

*definition and generation*

Without any modification, the script generates *homogeneous* pseudo-random networks. That is, the inter-neuron connections are generated with the same probability between all available chips. A homogeneous pseudo-random network of $n = 2^d$ chips therefore has $n(n-1)$ connections, each of a similar network load $L_c$. The inputs $I$ of each network block are evenly divided into $n$ groups of the same size, according to the $n$ chips. The fraction of $(n-1)/n$ of the synaptic inputs belongs to connections from distant chips, the fraction $1/n$ of the inputs is connected on-chip.

### Theoretical Values

*network loads for homogeneous networks*

The deterministic generation process allows to calculate theoretical values for the preliminary network qualifiers for the homogeneous networks of the script before the connections mapping (the routing) has been performed. Since each `Spikey` chip features two network blocks, the mean load per connections $\overline{L}_c = l$ equals

$$\overline{L}_c \leq 2I/n. \tag{5.17}$$

The given value is only a maximum value since the generation process may select the same neuron for the inputs of both networks blocks of the same chip, so that only the data of this single neuron has to be transported to the chip. The equations 5.15 and 5.16 result in

$$L'_{tot} \leq Idn, \tag{5.18}$$

$$\overline{L}'_e \leq I. \tag{5.19}$$

In particular, the mean number $\overline{L}'_e$ of neurons, whose data has to be transported via a single physical link converges against the number $I$ of inputs per network block and is independent of the number of chips $n$ or the number of neurons per chip.

*values for the mean neuron spike frequency*

The physical mean frequency $\overline{\nu}$ of the single neurons results out of the physical bandwidth $w$, which is divided by the number of neurons, which require 4 byte for its event encoding[2]:

$$\overline{\nu} \leq \frac{w}{\overline{L}'_e \cdot \frac{4 \text{ Byte}}{\text{neuron}}}. \tag{5.20}$$

Using $I = 256$ neurons and $w = 312.5$ MByte/s, the maximum mean event frequency for a single ANN neuron equals

$$\overline{\nu} \leq 305 \text{ kHz}. \tag{5.21}$$

This is the maximum physical mean frequency for single neurons of homogeneous pseudo-random networks, which the network can handle before dropping data. The

[2]The frame gap is ignored since the framing parameters can be set to large frame sizes. The reader may also refer table 4.7 of section 4.8.1 for the exact calculations for different numbers of neurons.

value is calculated for the case that the connection mapping algorithm succeeds to route all connections along the shortest path and that it succeeds to reserve 100 % of the physical bandwidth for the neural data.

*discussion of the value*

The important aspect is that the calculated number $\overline{\nu}$ is only a mean value, calculated for connections with comparably loads $L_c$. The allowed spike frequency of a particular physical neuron also depends on the actual spike frequencies of the other neurons of its connection. Since the transport network reserves the bandwidth exclusively for the connections, excess data cannot be transported and is dropped at the connection interface at the source node. This may require to even limit the physical mean bandwidth to lower rates to keep with bursts of single neurons or a correlated spike behavior of multiple neurons. Again, since the event rate of the neurons of the `Spikey` ANN chips depends on multiple factors, it is hardly possible to give assumptions for its allowable speedup $\mu$.

### Example Values for Generated Networks

*experimental values*

Table 5.6 shows the values of the preliminary classifiers for the generated pseudo-random networks. The networks all have a homogeneous $d$-dimensional binary cubic topology and include $2^d$ `Spikey` ANN chips each. All networks have 100 % hardware efficiency, thus all available neurons, synapse drivers and synapses of the chips are enabled. The number of inter-chip connections equals $n(n-1)$ as stated above. The values of the network loads $L'_{tot}$, $L_c$ and $L_e$ are preliminary and are assumed for a shortest-path routing between the source chips and the destination chips of the inter-chip connections.

| chip num | neuron num | synapse num | link num | intrinsic hop ratios | $\|C\|$ | $L'_{tot}$ | $\overline{L}_c$ | $\overline{L}'_e$ |
|---|---|---|---|---|---|---|---|---|
| 2 | 768 | 196608 | 2 | 1:1 | 2 | 430 | 215.0 | 215.0 |
| 4 | 1536 | 393216 | 8 | 1:2:1 | 12 | 1869 | 116.8 | 233.6 |
| 8 | 3072 | 786432 | 24 | 1:3:3:1 | 56 | 5933 | 61.7 | 247.2 |
| 16 | 6144 | 1572864 | 64 | 1:4:6:4:1 | 240 | 16046 | 31.3 | 250.7 |
| 32 | 12288 | 3145728 | 160 | 1:5:10:10:5:1 | 992 | 40498 | 15.8 | 253.1 |

Table 5.6: Properties of homogeneous pseudo-random neural networks to be mapped on `Spikey` ANN chips within the multi-dimensional binary cubic topology of the backplane. All hardware resources of the ANN chips are used. Synaptic connections between all ANN chips are assumed. The load-values $L'_{tot}$ and $\overline{L}'_e$ are preliminary and assume a shortest-path routing for all inter-chip connections.

*consequences*

As expected by equations 5.17 and 5.19, the mean load $\overline{L}_c$ of the connections decreases with an increasing number of chips, while the mean load $\overline{L}_e$ of the physical links converges against the value of $I = 256$ neurons. The value of $\overline{L}_e$ therefore limits the mean spike frequency of a single neuron according to equation 5.21.

## 5.4.4 Modified Pseudo-Random Networks

To reduce the values of $L'_{tot}$, $\overline{L}_c$ and $\overline{L}_e$ and to allow for a larger value of the neuron spike frequency $\overline{\nu}$, the parameters for the network generation process of homogeneous

networks (hop ratios, input count and synaptic efficiency) can be modified. It is shown in the following, how each modification affects the expected QoS results of the inter-chip connections during the connection mapping process of the transport network.

### Modification of the Hop Ratios

*main idea*

A modification of the hop ratios can be used to decrease the required network load and to improve the guaranteed QoS values for throughput, delay and jitter. This is since the hop ratios denote the relative number of neurons to be taken as synaptic inputs for a dedicated network block depending on its shortest-path hop distance. Networks with modified hop ratios still use all available inputs, neurons and synapses of the ANN chips, but the inter-chip connections have a limited number of hops. Note that a modification of the hop ratios does primarily affect the number of neurons $L_c$ of the connections and not the number of the connections itself. In fact, the number of connections is reduced only in the case that a particular hop ratio is set to zero.

*example hop ratios*

Table 5.7 shows the resulting network loads for various pseudo-random networks with modified hop ratios, whereas figure 5.4 exemplarily shows the neuron connectivity matrix for a 16-chip network with the modified hop ratios *10:4:6:0:0*. In this case, each chip takes about 50 % of its synaptic inputs from off-chip and 50 % is connected on-chip[3]. Due to the intrinsic hop ratios of *1:4:6:4:1* of the backplane topology, the ratios *10:4:6:0:0* result in similar values for the number $\overline{L}_c$ of neurons of the inter-chip connections.

| chips | hop ratios 0..4 hops | inputs from off-chip | inter-chip connect. | $L'_{tot}$ | $\overline{L}_c$ | $\overline{L}'_e$ |
|---|---|---|---|---|---|---|
| 16 | 1:4:6:4:1 | 93.8 % | 240 | 16055 | 31.3 ± 0.8 | 250.9 |
| 16 | 1:1:1:1:1 | 79.4 % | 240 | 15805 | 26.3 ± 20.0 | 247.0 |
| 16 | 15:4:6:4:1 | 48.4 % | 240 | 8358 | 16.3 ± 0.8 | 130.6 |
| 16 | 4:3:2:1:0 | 59.1 % | 224 | 7880 | 21.3 ± 10.8 | 123.1 |
| 16 | 10:4:6:0:0 | 48.4 % | 160 | 6233 | 24.3 ± 1.0 | 97.4 |
| 16 | 3:2:1:0:0 | 49.7 % | 160 | 5337 | 24.9 ± 13.5 | 83.4 |
| 16 | 1:1:0:0:0 | 50.0 % | 64 | 3936 | 61.5 ± 1.5 | 61.5 |

Table 5.7: Reduction of the total network load $L'_{tot}$ and the mean load per physical link $\overline{L}'_e$ by modifying the hop ratios for networks of a fully equipped backplane with 16 `Spikey` chips. All networks use 100 % of the available neurons and synapses.

*throughput and loss rate*

**QoS Effects** The reduction of $L'_{tot}$ and $\overline{L}'_e$ increases the available bandwidth per neuron and thus reduces the expected loss rate of spike events. As a drawback, the usage of hop ratios that differ from the intrinsic ratios of the $d$-dimensional cubic topology of equation 5.14 leads to significant deviations in the particular connections loads $\overline{L}_c$ from its mean values as it can be seen in figure 5.4.

*unbalanced connection load*

Concerning the connection mapping process, this requires to assign a different

[3]The value is not exactly 50 %, since the number of available synapse drivers cannot be equally divided among the connections.

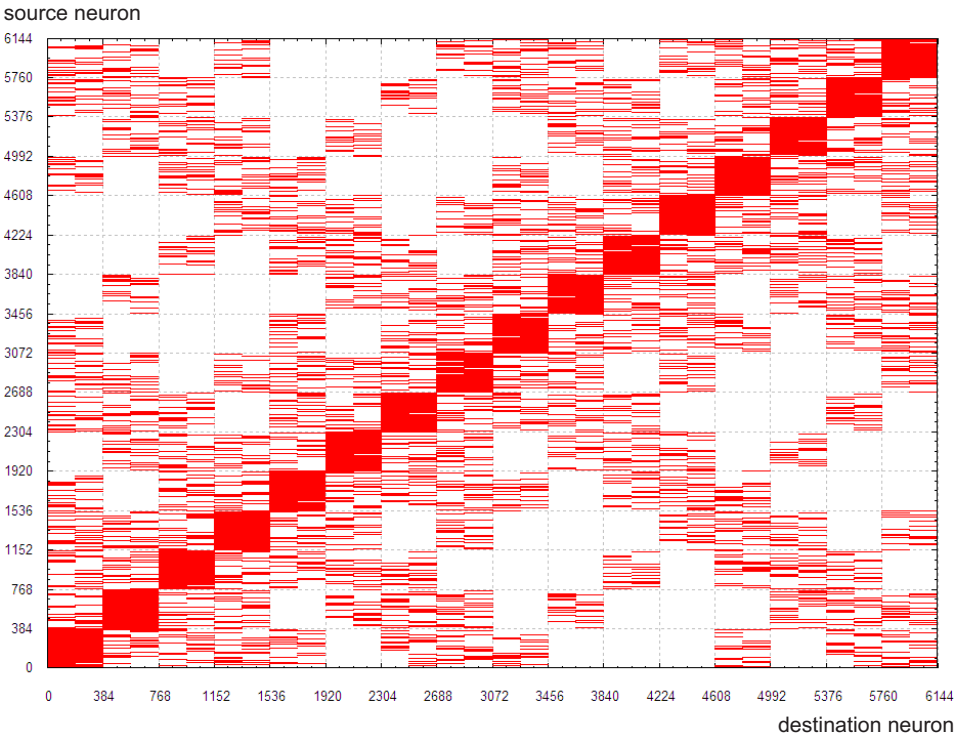



Figure 5.4: The inter-neuron connectivity matrix of a pseudo-random 16-chip network with the modified hop ratios of *10:4:6:0:0*. Although all available neurons and synapses of the ANN chips are used, the total network load is reduced significantly. About 50 % of the synaptic inputs are taken from off-chip.

number of time slots to the different connections according to the values of $L_c$. This may lead to an unbalanced usage of time slots and thus to an increased drop rate for particular connections. This effect is illustrated in the figures 5.5 and 5.6, which show the distribution of $L_c$ for the networks of table 5.7. It can be seen that the hop ratios that correspond to the intrinsic ratio of the cubic topology result in smaller deviations from the mean value of $L_c$.

*calculation example for 3:2:1:0:0*

Consider a 16-chip network with selected hop ratios of *3:2:1:0:0*. About 50 % of the synaptic inputs of the two network blocks are to be selected from off-chip (2 × 128 inputs). Two thirds of these inputs (171) are to be taken from neurons on chips at a single hop distance, resulting in 4 connections with an $L_c$ of about 43 neurons each. One third of the inputs (85) is to be taken from the 6 chips being two hops distant, resulting in 6 connectinos with an $L_c$ of about 14 neurons. The loads of the required inter-chip connections therefore differ by a factor of three, which requires three times the number of time slots to get a balanced link usage (cf. again figure 5.6).

*delay*

The delays of the connections mainly depend on its number of network hops (cf. e.g. table 5.4 of section 5.3.1). A modification of the ratios can therefore be used to limit the delays of the synaptic connections to be modeled. By setting hop numbers to zero, synaptic connections to the corresponding distances are not generated. As an example, the setting *1:1:0:0:0* of a 16-chip network generates only connections to adjacent chips (about 50 %), by still using all hardare resources.

*jitter*

The jitter of the connections depends on the assignment of time slots within the reservation periods of the physical links during the reservation process. Since the hop ratios can be used to reduce the number of connections by setting particular ratios to zero, this reduces the number of connections per physical link and thus increases the number of time slots for the remaining connection and results in a more dense assignment with better jitter results.

### Modification of the Input Count

*method*

The reduction of the input count $I$ results in a reduction of the required physical bandwidth of the neural network. The input count can be reduced by using only a fraction of the available inputs during the network generation process. Although the limited input count is a main limitation of the ANN chips, its reduction has the advantage that it increases the possible spike frequencies of the single neurons by keeping to the basic topology of the network in terms of its number of used chips or its hop ratios. The exact topology is indeed changed since the leaving inputs unused equals a reduced connectivity. As a drawback, this also decreases the general hardware efficiency since the synapses that correspond to the affected inputs cannot be used with other inputs.

*reduced bandwidth requirement*

**QoS Effects** Concerning the required bandwidth, equations 5.17 to 5.19 show that a reduction of the input count corresponds to a linear reduction of the number of neurons per connection $L_c$ and thus to a linear reduction of the total amount of neural network data $L_{tot}$ as well as to a reduction of the link loads $L_e$. The reduced $L_c$ allows higher spike frequencies for the single neurons and reduces the probability for event drops at the interface of the connections.

*unchanged delay, jitter*

Since the absolute amount of inter-chip connections as well as the hop ratios remain unchanged, the network topology and the routes of the connections are unchanged, too. The connection mapping algorithm therefore has the same result and there is no effect on the resulting QoS delay or jitter.

### Modification of the Synaptic Efficiency

The example networks presented so far feature a synaptic efficiency of 100 %, i.e. all synapses within the ANN chips are enabled for a high connectivity of the generated network. However, this leads to the effect that neuron destinations are correlated: each neuron that sends its spike events to a synapse driver on a certain network block has all its destination neurons in common with other neurons that send data to the same block.

*method*

To investigate the bandwidth requirement of sparsely connected networks, the synaptic efficiency can be reduced to a given probability. A very low probability may even result in the case that two synapse drivers on the same chip are connected to a disjoint subset of the local neurons. A modification of the synaptic efficiency therefore allows to trade the *diversity* or *randomness* of the generated network against its connectivity and hardware efficiency. Note that the total number of the used synapse drivers and neurons remains unchanged. Figure 5.7 exemplarily illustrates a subset of the connectivity matrix of a network with 10 % synapse efficiency.

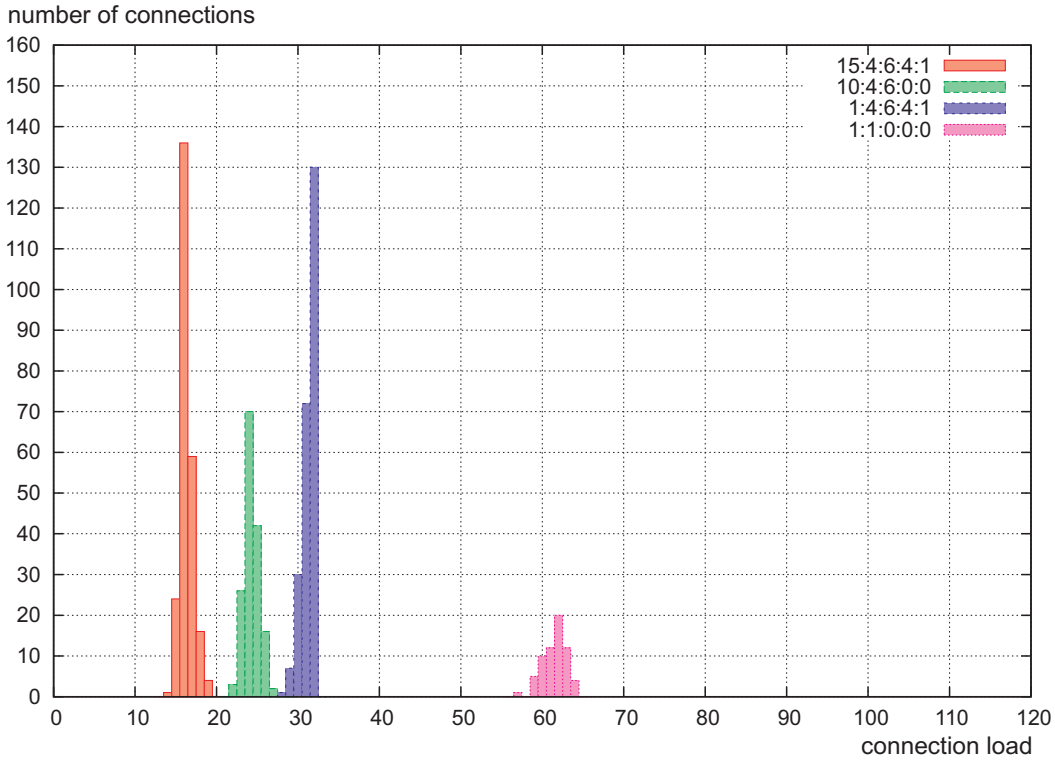



Figure 5.5: Distribution of the connection loads $L_c$ for hop ratios of a 16-chip network that correspond to the cubic topology. The low deviations of $L_c$ result in a balanced assignment of time slots to all connections.

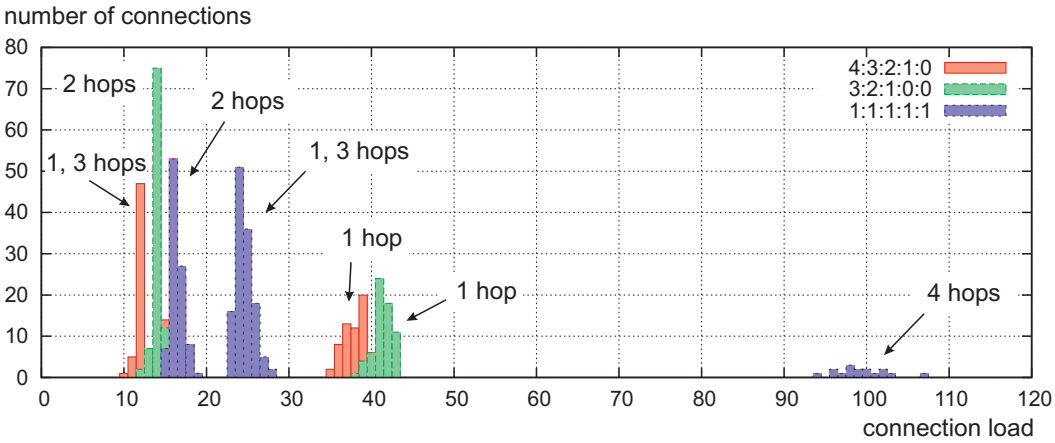



Figure 5.6: Distribution of the connection loads $L_c$ for a 16-chip network with hop ratios that are not oriented at the intrinsic cubic topology of the backplane. The resulting values for $L_c$ depend on the hop distances and require a different number of time slots to balance the link usages.

Figure 5.7: The neuron connectivity matrix of a 16-chip network with 10 % synaptic efficiency. The reduction of the probability for the synapses to be activated increases the diversity or randomness of the generated networks. The resulting network load remains unaffected except for very low probabilities.

*number of destination neurons*

The statistical activation of synapses also leads to a more diverse distribution of the *number* of destination neurons for a single source neuron. In networks with a synapse efficiency of 100 %, the number of destination neurons is a multiple of the 192 neurons of a network block. A sparse connectiviy distributes the number of destinations neurons, which can be interpreted as a higher diversity or randomness of the generated networks. The difference is plotted exemplarly for a network of 16 chips in figure 5.8 and figure 5.9 for synapse efficiencies of 100 % and 50 %, respectively.

*no QoS effects*

**QoS Effects** The activation or deactivation of synapses is a modification within the ANN chips. The number of used neurons or input synapse drivers remains unchanged. Since the amount of data to be transported between two ANN chips depends on the number of the neuron outputs on one chip, which has to be transported to input synapse drivers of the other chip, a reduction of the synapse efficiency cannot be used to reduce the network load. The reduction in the synapse efficiency has therefore no primary effect on the QoS loss rate, delay or jitter. To indeed reduce the network load, the synaptic efficiency has to be reduced to very low efficiencies in the range of few percents such that certain synapse drivers are deactivated at all. Table 5.8 shows example calculations for different efficiencies.

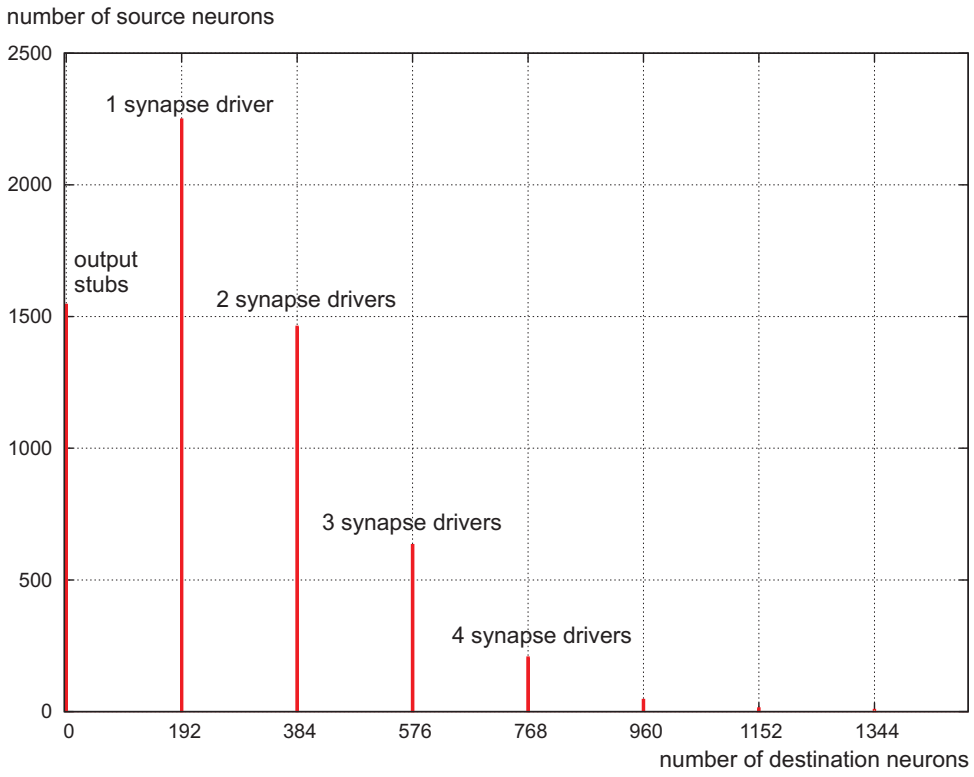



Figure 5.8: Distribution of the number of destination neurons of a pseudo-random network of 16 `Spikey` chips with 100 % of the available synapses used. Each inter-chip connection from a neuron output to a synapse driver results in 192 destination neurons.

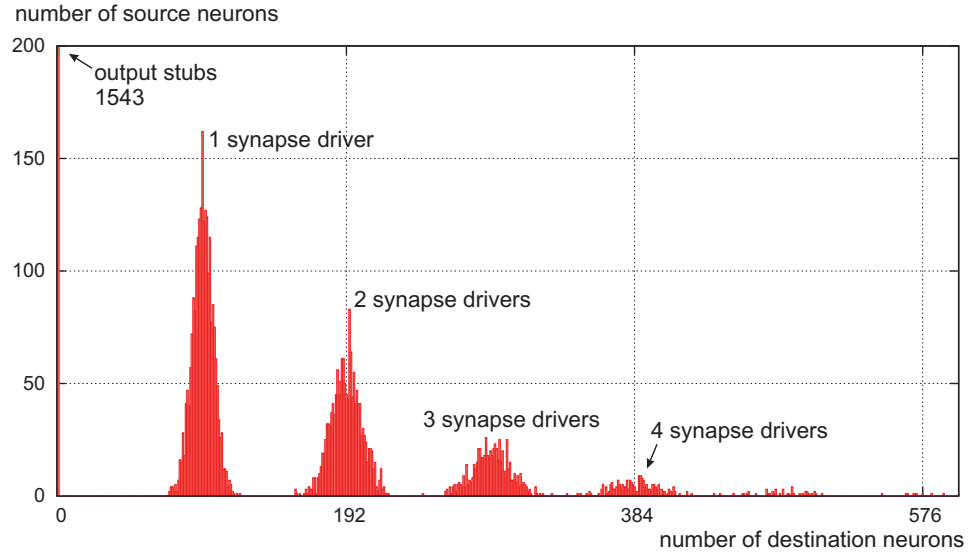



Figure 5.9: Distribution of the number of destination neurons of a pseudo-random network of 16 `Spikey` chips with 50 % synaptic efficiency. The chip-internal synapses are enabled statistically, which enhances the randomness of the network configuration. The number of output stubs of the network remains unchanged.

| chips | synapse num | synapse efficiency | conn. num | inputs from off-chip | $L'_{tot}$ | $\overline{L}_c$ | $\overline{L}_e$ |
|---|---|---|---|---|---|---|---|
| 16 | 1179461 | 75.0 % | 240 | 93.8 % | 16046 | 31.3 | 250.7 |
| 16 | 787245 | 50.1 % | 240 | 93.8 % | 16046 | 31.3 | 250.7 |
| 16 | 393721 | 25.0 % | 240 | 93.8 % | 16046 | 31.3 | 250.7 |
| 16 | 157435 | 10.0 % | 240 | 93.8 % | 16046 | 31.3 | 250.7 |
| 16 | 78809 | 5.0 % | 240 | 93.7 % | 16044 | 31.3 | 250.7 |
| 16 | 15761 | 1.0 % | 240 | 93.9 % | 13793 | 26.9 | 215.5 |

Table 5.8: Pseudo-random networks of 16 `Spikey` chips with different reduced synaptic efficiencies. A reduced efficiency does not lower the network load significantly.

### 5.4.5 Summary of the Evaluated Network Topologies

*limited physical spike frequencies*

Two types of network topologies have been investigated: homogeneous pseudo-random networks and modified pseudo-random networks. It has been shown that the estimated required link load $L_e$ of homogeneously connected networks converges against the input count $I$ of the network blocks and limits the maximum mean spike frequencies for the single neurons to about 305 kHz.

*possible enhancements*

The possible spike frequencies can be increased by a reduction of the link load. Three modifications of the homogeneous network topology are conceivable:

1. A modification of the hop ratios changes the number of neuron outputs that are transported via connections of a certain number of hops. This does not only allow to control the on-chip to off-chip ratio of the inter-neuron connections, but also to avoid connections with a large physical delay completely. Hop ratios not oriented on the intrinsic topology of the backplane result in a deviation within the connection load $L_c$ and thus in a different number of slots per connection and may cause unbalanced link usages.
2. A reduction of the input count results in a linear reduction of the required bandwidth. The drawback is the reduced connectivity and hardware efficiency.
3. A reduction of the synapse efficiency (while keeping the input count and the neuron number per chip) does not reduce the bandwidth requirement, but enhances the diversity and the randomness of the network.

*preliminary values*

The listed values for $L_{tot}$ and $L_e$ are preliminary mean values and have been calculated according to the distribution of the inputs during the generation process. To use the networks for experiments, the connections have to be quantized, routed and the collision-free reservation pattern has to be assigned. This is done by the connection mapping process, which is evaluated in the next section.

## 5.5 Evaluation of the Connection Mapping Algorithm

*purpose*

The connection mapping algorithm is executed during the network initialization phase of the transport network. Its purpose is to route the inter-chip connections of the placed networks and to calculate the contention-free reservation pattern of the

physical links, which are stored into the routing tables of the switches. The software implementation of the algorithm has been described in section 4.10.2. The following paragraphs evaluate the performance of the connection mapping algorithm. The discussion focuses on neural networks using the `Spikey` chip, but the results can be transferred comparably to the `HAGEN` chip with deterministic bandwidth requirements.

### 5.5.1 Evaluated Qualifiers

*presentation of the results*

The performance of the algorithm has been evaluated by calculating the network qualifiers of the previous section of the routed networks. Since not all qualifiers depend on the route of the connections, but on the netlist and the neuron placement, the values of the number of used chips, block inputs $I$ (synapse drivers), neurons per chip or the connection loads $L_c$ remain unchanged. The former preliminary values for the total network load $L_{tot}$ and the loads of the physical links $L_e$ become fixed during the calculation of the routes. The following tables also list the slot usage (occupancy) of the network. This denotes the fraction of time slots that have been reserved for neural connections and thus the remaining bandwidth that is available for best-effort packet transfers. The calculation of the routes and thus the routing tables determines the final QoS guarantees of the isochronous inter-chip connections of the transport network:

*throughput*

1. The throughput of the isochronous connections depends on the reserved bandwidth in terms of time slots $m_c$. The grouping of inter-neuron connections to inter-chip connections then determines the available average bandwidth $\overline{w}_n$ for a single neuron of the `Spikey` chip. The value $\overline{w}_n$ is calculated over all connections.

*delay*

2. The end-to-end connection delay $D_c$ mainly depends on the number of network hops of the selected routes. In the case that the mapping algorithm succeeds to route all connections along their shortest paths, the distribution of the delays corresponds to the hop ratios $H$ of the mapped networks. The resulting delays for various hop distances have already been discussed in section 5.3.1.

*jitter*

3. The end-to-end connection jitter $J_c$ is calculated out of the interface wait time and the transmission delays according to equation 3.24. It mainly depends on the number $m_c$ of time slots per connection out of the available number $m$ of time slots per reservation period. According to equation 4.8, the jitter is bounded by $J_c \leq (2m+1) \cdot 6.4\,\text{ns}$ for all connections. The maximum value calculates for $m_c = 1$. The minimum possible jitter is $J_c = 19.2\,\text{ns}$.

#### Test Cases

*parameter sets*

The connection mapping algorithm has been evaluated with different sets of homogeneous and modified pseudo-random networks consisting of 2 to 32 ANN chips. The networks and their neuron placements have been generated with the network generator script. The evaluation has been done for different sets of the following parameters:

1. The number $p{=}N_p$ of local switch ports for connection-oriented traffic. Large values allow a high off-chip event rate and simplify the assignment problem of the time slots, but result in an increased consumption of programmable logic.
2. The number $m$ of time slots per reservation period. It determines the granularity of the division of the physical bandwidth into time slots and is the key parameter for the algorithm. Low values of $m$ lead to low jitter, but may hinder a successful mapping.
3. The number $f$ of time slots per frame. The value of $f$ must be an integer multiple of $m$. Large values reduce the bandwidth waste caused by the frame gap.
4. The maximum number $l$ of neurons, whose data are be transported within the same reserved time slot of a connection. Low values lead to a higher number of reserved slots per connection and reduce the drop rates as well as the jitter, but require a higher granularity and significantly complicate the global assignment problem.

*presentation*

The mapping results are given in tabular form arranged in three parts. The first set of columns denote the network to be mapped according to its input count $I$ and the hop ratios $H$. The second set of columns denote the mapping parameters $p$, $m$, $f$ and $l$ if used. The last set lists the QoS results in terms of jitter, the provided bandwidth and the possible physical mean frequencies of the neurons. The mapping parameters have been selected such that the lowest possible number of mapping slots $m$ has been used for a given number of ports $p$ such that a successful mapping could be achieved. This equals the trade-off of the available amount of logic within the FPGA against the network performance.

### Interpretation of the Results

*interpretation of QoS results*

The interpretation of the resulting values for the available throughput, the delay and the jitter has to be made with respect to the neuron model. The required values `Spikey` ANN to the transport network have been discussed in section 2.4.2. Since the `Spikey` chip is only the implementation of the leaky integrate-and-fire model in hardware, these values are no strict requirements, but merely a degree of the quality of the modeling. In particular, the speedup $\mu$ of the chip denotes the configuration of the timing parameters of the membrane potential of the neuron.

*interpretation of delay and jitter*

The delay of the isochronous connections should therefore be in accordance with the delay of the corresponding biological axonal connections with respect to the selected speedup. This basically enforces the delay to be as small as possible. The same can be stated for the jitter. Although a jitter different from zero can be eliminated using a jitter buffer at the destination of the connection, this clearly increases the delay and should be reduced.

*interpretation of throughput*

The interpretation of the available throughput has to consider that the number of spikes to be generated by the `Spikey` chip at runtime depends on a number of conditions (like the input stimuli) and can hardly be predicted from the network topology or from the configured speedup of the chip. The following evaluation therefore denotes only the maximum physical mean frequency $\overline{\nu}$ of a single neuron, which

is calculated over all connections out of their mean bandwidth $w_c$ and their connection load $L_c$. The value $\overline{\nu}$ denotes the average frequency of the single neurons which the network can handle without dropping events at the interface. It can be used to compare the bandwidth requirements of different network topologies or to compare the mapping results of different parameter sets.

### 5.5.2 Mapping Results of Pseudo-Random Networks

*selected networks*

The pseudo-random networks have been mapped on backplane topologies of 2, 4, 8 and 16 nodes, as well as on two backplanes with 32 nodes by using an additional MGT connection between the network modules at the same same positions on the two backplanes. For each number of chips, the connection mapping has first been executed for the unmodified homogeneous pseudo-random networks of table 5.6. Further mappings have been made for an increased number of ports, a reduced input count and for modified hop ratios to increase the available mean neuron frequencies $\overline{\nu}$.

*results*

The mapping results are shown in the tables 5.9 to 5.13. It can be seen that the mapping algorithm succeeds to occupy 100 % of all available slots in most cases if the parameter $p$ of the used local ports is large enough to provide sufficient bandwidth. Furthermore, the mappings with large frame sizes result in the mean frequency for a single neuron of 0.31 MHz as calculated in equation 5.21 (for the two-chip network, the value is slightly higher due to the increased probability, that the outputs of two neurons have to be fed to both network blocks on the same chip, which saves routing resources). In addition, a reduction of the input count does not improve the jitter as expected (the number of connections is the same), but indeed results in a nearly linearly increased mean bandwidth per neuron. To improve the jitter and to further increase $\overline{w}_n$, the hop ratios can be modified to leave the farthest connections with the largest delays out.

**2-Chip Networks** Table 5.9 lists example parameters for two-chip networks. The mapping simply uses a single connection in each direction and a single local port. Since all time slots can be used, the throughput equals the usable bandwidth of the physical links. The jitter is minimal.

**4-Chip Networks** The selected mapping results of four-chip network are shown in table 5.10. Two local ports and two time slots per reservation period are needed to provide the required off-chip data rate. The modification of the hop ratios simplifies the mapping task such that only connections between adjacent chips remain, which only a single required time slot and lower jitter.

**8-Chip Networks** A pseudo-random network of eight chips of table 5.11 requires three local ports and four slots per reservation period to occupy 100 % of the network's resources. A slight modification of the hop ratios that avoids the eight farthest connections allows to reduce the port number and the slot number both to three for a reduced jitter by still using the full network bandwidth.

**16-Chip Networks** The 16-chip networks of table 5.12 occupy a full backplane and require four global ports. A granularity of eight slots per reservation period is required to achieve 100 % bandwidth usage. Since this requires eight local ports and thus a large amount of programmable logic, a 16 chip network has to be implemented either non-homogeneous or with less slot occupancy. The last table entries list example mapping parameters for modified hop ratios that us only adjacent connections or also two-hop connections and require only few local ports.

**32-Chip Networks** Table 5.13 lists the results for homogeneous networks with 32 chips. The high number of 992 different connections requires a fine bandwidth granularity and thus large reservation periods and a large jitter. It is notable that the mapping algorithm succeeds even for networks of this complexity and reaches a slot occupancy of 80.1 %. Since large reservation periods result in a high jitter, table 5.13 also lists networks with reduced hop ratios to only up to two hops distance.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | slot usage | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1:1 | 256 | 2 | 1 | 1 | 12 | 19.2 | 100 % | 215.0 | 1.34 ± 0.02 | 0.34 |
| 1:1 | 256 | 2 | 1 | 1 | 60 | 19.2 | 100 % | 215.0 | 1.43 ± 0.02 | 0.36 |
| 1:1 | 128 | 2 | 1 | 1 | 60 | 19.2 | 100 % | 118.5 | 2.59 ± 0.01 | 0.65 |
| 2:1 | 256 | 2 | 1 | 1 | 60 | 19.2 | 100 % | 151.0 | 2.04 ± 0.06 | 0.51 |

Table 5.9: Mapping results of two-chip networks. The connections use one link in each direction and occupy all time slots.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | slot usage | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1:2:1 | 256 | 12 | 1 | 3 | 12 | 44.8 | 66.7 % | 233.6 | 0.82 ± 0.02 | 0.21 |
| 1:2:1 | 256 | 12 | 2 | 2 | 12 | 32.0 | 100.0 % | 233.6 | 1.23 ± 0.03 | 0.31 |
| 1:2:1 | 256 | 12 | 2 | 2 | 60 | 32.0 | 100.0 % | 233.6 | 1.32 ± 0.03 | 0.33 |
| 1:2:1 | 128 | 12 | 2 | 2 | 60 | 32.0 | 100.0 % | 123.3 | 2.50 ± 0.06 | 0.63 |
| 2:2:0 | 256 | 8 | 2 | 1 | 60 | 19.2 | 100.0 % | 116.3 | 2.65 ± 0.06 | 0.66 |

Table 5.10: Mapping results of 4-chip networks. 100 % capacity of the physical links can be used with minimum jitter.

### 5.5.3 Mapping Results of Networks with non-Intrinsic Hop Ratios

*deviation in bandwidth requirements*

Networks with non-intrinsic hop ratios can feature a significantly different connection load as described in section 5.4.4. Without further control, this effect causes different $w_n$ values and thus different maximum frequencies between the neurons of highly loaded and lightly loaded connections (cf. figure 5.6). To limit the overall drop rate, the network is then forced to consider the connection with the *lowest* bandwidth as the reference. The effect can be handled with the mapping parameter $l$ to enhance the number of time slots for highly booked connections.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | reserved slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1:3:3:1 | 256 | 56 | 1 | 8 | 8 | 108.8 | 50.0 % | 247.2 | 0.56 ± 0.01 | 0.14 |
| 1:3:3:1 | 256 | 56 | 2 | 5 | 5 | 70.4 | 80.0 % | 247.2 | 0.84 ± 0.02 | 0.21 |
| 1:3:3:1 | 256 | 56 | 3 | 4 | 4 | 57.6 | 100.0 % | 247.2 | 1.01 ± 0.02 | 0.25 |
| 1:3:3:1 | 256 | 56 | 3 | 4 | 60 | 57.6 | 100.0 % | 247.2 | 1.25 ± 0.03 | 0.31 |
| 1:3:3:1 | 128 | 56 | 2 | 5 | 60 | 70.4 | 80.0 % | 125.7 | 1.96 ± 0.04 | 0.49 |
| 1:3:3:1 | 128 | 56 | 3 | 4 | 60 | 57.6 | 100.0 % | 125.7 | 2.45 ± 0.06 | 0.61 |
| 4:3:3:0 | 256 | 48 | 2 | 4 | 60 | 57.6 | 75.0 % | 148.3 | 1.56 ± 0.04 | 0.39 |
| 4:3:3:0 | 256 | 48 | 3 | 3 | 60 | 44.8 | 100.0 % | 148.3 | 2.07 ± 0.06 | 0.52 |

Table 5.11: Mapping results of eight-chip networks. By modifying the hop ratios to avoid the eight farthest connections, 100 % of the network's resources can be used with three local ports and three slots per reservation period.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | used slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1:4:6:4:1 | 256 | 240 | 1 | 17 | 51 | 224.0 | 47.1 % | 250.7 | 0.58 ± 0.01 | 0.15 |
| 1:4:6:4:1 | 256 | 240 | 2 | 11 | 55 | 147.2 | 72.7 % | 250.7 | 0.89 ± 0.02 | 0.22 |
| 1:4:6:4:1 | 256 | 240 | 3 | 10 | 60 | 134.4 | 80.0 % | 250.7 | 0.98 ± 0.02 | 0.25 |
| 1:4:6:4:1 | 256 | 240 | 4 | 9 | 63 | 121.6 | 88.9 % | 250.7 | 1.09 ± 0.03 | 0.27 |
| 1:4:6:4:1 | 256 | 240 | 8 | 8 | 64 | 108.8 | 100.0 % | 250.7 | 1.23 ± 0.03 | 0.31 |
| 1:4:6:4:1 | 192 | 240 | 2 | 11 | 55 | 147.2 | 72.7 % | 188.9 | 1.18 ± 0.03 | 0.30 |
| 1:4:6:4:1 | 192 | 240 | 3 | 10 | 60 | 134.4 | 80.0 % | 188.9 | 1.30 ± 0.03 | 0.33 |
| 1:4:6:4:1 | 192 | 240 | 4 | 9 | 63 | 121.6 | 88.9 % | 188.9 | 1.45 ± 0.04 | 0.36 |
| 1:4:6:4:1 | 128 | 240 | 2 | 11 | 55 | 147.2 | 72.7 % | 126.8 | 1.76 ± 0.04 | 0.44 |
| 1:4:6:4:1 | 128 | 240 | 3 | 10 | 60 | 134.4 | 80.0 % | 126.8 | 1.94 ± 0.05 | 0.49 |
| 1:4:6:4:1 | 128 | 240 | 4 | 9 | 63 | 121.6 | 88.9 % | 126.8 | 2.16 ± 0.05 | 0.54 |
| 10:4:6:0:0 | 256 | 160 | 2 | 6 | 60 | 83.2 | 66.7 % | 97.5 | 2.10 ± 0.09 | 0.53 |
| 10:4:6:0:0 | 256 | 160 | 3 | 5 | 60 | 70.4 | 80.0 % | 97.5 | 2.52 ± 0.10 | 0.63 |
| 1:1:0:0:0 | 256 | 64 | 1 | 4 | 60 | 57.6 | 25.0 % | 61.5 | 1.25 ± 0.03 | 0.31 |
| 1:1:0:0:0 | 256 | 64 | 2 | 2 | 60 | 32.0 | 50.0 % | 61.5 | 2.50 ± 0.06 | 0.63 |
| 1:1:0:0:0 | 256 | 64 | 3 | 2 | 60 | 32.0 | 50.0 % | 61.5 | 2.50 ± 0.06 | 0.63 |
| 1:1:0:0:0 | 256 | 64 | 4 | 1 | 60 | 19.2 | 100.0 % | 61.5 | 5.00 ± 0.12 | 1.25 |

Table 5.12: Mapping results of 16-chip networks. The large number of 240 inter-chip connections requires to omit long-range connections or to reduce the used input-count of the chips to achieve a high possible neuron spike frequency.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | used slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1:5:10:10:5:1 | 256 | 992 | 2 | 23 | 69 | 300.8 | 69.6 % | 253.3 | 0.85 ± 0.02 | 0.21 |
| 1:5:10:10:5:1 | 256 | 992 | 3 | 20 | 60 | 262.4 | 80.1 % | 253.3 | 0.97 ± 0.03 | 0.24 |
| 1:5:10:0:0:0 | 256 | 480 | 2 | 10 | 60 | 134.4 | 50.0 % | 156.5 | 0.98 ± 0.02 | 0.25 |
| 1:5:10:0:0:0 | 256 | 480 | 3 | 8 | 64 | 108.8 | 62.5 % | 156.5 | 1.23 ± 0.03 | 0.31 |
| 1:5:10:0:0:0 | 256 | 480 | 4 | 7 | 63 | 96.0 | 71.4 % | 156.5 | 1.40 ± 0.03 | 0.35 |
| 1:5:10:0:0:0 | 192 | 480 | 2 | 10 | 60 | 134.4 | 50.0 % | 118.4 | 1.30 ± 0.03 | 0.33 |
| 1:5:10:0:0:0 | 192 | 480 | 3 | 8 | 64 | 108.8 | 62.5 % | 118.4 | 1.62 ± 0.04 | 0.41 |
| 1:5:10:0:0:0 | 192 | 480 | 4 | 7 | 63 | 96.0 | 71.4 % | 118.4 | 1.86 ± 0.04 | 0.46 |

Table 5.13: Mapping results of 32-chip networks distributed over two backplanes. The setup operates 12288 neurons and 3.1 million synapses.

*examples*

Table 5.14 lists example networks with non-intrinsic hop-ratios. The jitter is denoted as the mean jitter $\overline{J}_c$ of all connections. It can be seen that a limitation of the slot load requires multiple time slots to be reserved for the high-bandwidth connections and thus evens the available bandwidth per neuron. Although this indeed requires a larger reservation period in some cases, it also evens the maximum average frequency of all neurons. Figure 5.10 illustrates the case for the 16-chip network with the hop ratios *3:2:1:0:0*.

| H | I | $\|C\|$ | $p$ | $m$ | $f$ | $l$ | $\overline{J}_c$ [ns] | used slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1:1:1:1:1 | 256 | 64 | 3 | 10 | 60 | - | 134.4 | 80.0 % | 247.2 | 1.44 ± 0.43 | 0.36 |
| 1:1:1:1:1 | 256 | 64 | 3 | 10 | 60 | 40 | 137.0 | 84.1 % | 250.2 | 1.23 ± 0.28 | 0.31 |
| 4:3:2:1:0 | 256 | 224 | 3 | 9 | 63 | - | 121.6 | 77.8 % | 122.6 | 1.98 ± 0.75 | 0.50 |
| 4:3:2:1:0 | 256 | 224 | 3 | 11 | 66 | 30 | 137.6 | 72.7 % | 122.6 | 1.83 ± 0.34 | 0.46 |
| 3:2:1:0:0 | 256 | 160 | 3 | 5 | 60 | - | 70.4 | 80.0 % | 83.0 | 3.24 ± 1.44 | 0.81 |
| 3:2:1:0:0 | 256 | 160 | 3 | 7 | 63 | 30 | 87.0 | 71.4 % | 83.0 | 2.75 ± 0.51 | 0.69 |
| 3:2:1:0:0 | 256 | 160 | 3 | 9 | 63 | 18 | 103.0 | 66.7 % | 83.0 | 2.47 ± 0.10 | 0.62 |

Table 5.14: Mapping results of 16-chip networks with non-intrinsic hop ratios. The number of time slots for highly booked connections can be increased to even the available mean bandwidth for the affected neurons.

### 5.5.4 Mapping Results of Alternative Hardware Topologies

*adding and removing backplane links*

To demonstrate the flexibility of the mapping algorithm, the algorithm has been tested on hardware topologies that differ from the cubic regular topology of the backplane. The hardware topologies have been selected out of the possible extensions that can be achieved with the four additional MGT connectors on top of each `Nathan` module (cf. section 2.2.3) and also by leaving existing physical connections unused. As an example, the following physical topologies are conceivable:

*examples*

- A feed-forward topology, i.e. a sequential arrangement.

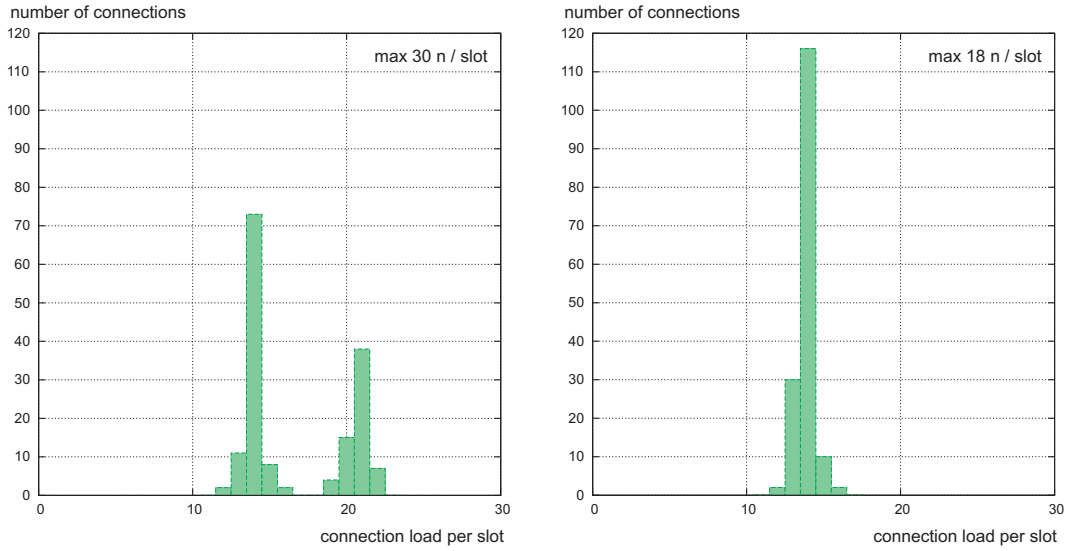



Figure 5.10: Resulting distribution of neurons per slot of the connections of the 16-chip pseudo-random network with the hop ratios 3:2:1:0:0. Connections with a higher load get multiple slots to even the usage of the physical bandwidth. Left: using a maximum neuron number of 30 per slot. Right: using a maximum number of 18 neurons per slot. (cf. figure 5.6).

- Two network nodes with 5 times the standard link capacity.
- A fully connected network of four nodes.
- A fully, doubly connected network of four nodes.
- A fully connected networks of eight nodes.
- A doubly connected backplane of 16 nodes.
- Modifications of the cubic topology: the folded and twisted hypercubes.
- Broken physical links.

The mapping program has been informed about the physical topology via its hardware configuration file. The appropriate netlists and mappings have been created with the network generator script with a modified configuration file. All networks use 100 % of the ANN resources, i.e. all available input drivers, neurons and synapses. The next paragraphs briefly present mapping results for four examples.

*simple mapping task, good results*

**Feed-Forward Networks** Each network node within a feed-forward network has only one successor. Recursions within the neural network to be modeled can be implemented only on-chip. Clearly, the mapping task is simple since only a single connection is routed from each chip to its successor and all time slots on the corresponding physical link can be used for it. Only two global ports and a single local port are required to use all neurons and synapses of the backplane. Table 5.15 shows mapping results for feed-forward networks of four and 16 chips with and without on-chip loops. The possible physical mean frequencies for the single neurons can be tuned by further reducing the fraction of neurons that transmit data off-chip.

| n | H | I | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | used slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0:1:0 | 256 | 4 | 1 | 1 | 60 | 19.2 | 100 % | 341.5 | 0.90 ± 0.01 | 0.23 |
| 4 | 1:1:0 | 256 | 4 | 1 | 1 | 60 | 19.2 | 100 % | 213.3 | 1.44 ± 0.04 | 0.36 |
| 16 | 0:1:0:0:0 | 256 | 16 | 1 | 1 | 60 | 19.2 | 100 % | 341.3 | 0.90 ± 0.02 | 0.23 |
| 16 | 1:1:0:0:0 | 256 | 16 | 1 | 1 | 60 | 19.2 | 100 % | 213.4 | 1.44 ± 0.03 | 0.36 |

Table 5.15: Mapping results of four-chip and 16-chip feed-forward networks. Only the physical links to the successor chip have been available for the mapping algorithm.

*enhanced bandwidth*

**4-Node Fully Connected Topology** The upper part of table 5.16 shows mapping results of homogeneous pseudo-random four-chip networks. It can be seen that an extension of the cubic topology to a fully connected topology enhances the external bandwidth, which can be used by increasing the internal ports from two to three. As a result of this, the available bandwidth per single neuron doubles.

| n | hardware topology | H | $\|C\|$ | $p$ | $m$ | $f$ | $J_c$ [ns] | reserved slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | cubic | 1:2:1 | 12 | 2 | 2 | 60 | 32.0 | 100.0 % | 233.6 | 1.32 ± 0.03 | 0.33 |
| 4 | fully | 1:2:1 | 12 | 2 | 2 | 60 | 32.0 | 50.0 % | 116.8 | 1.32 ± 0.03 | 0.33 |
| 4 | fully | 1:2:1 | 12 | 3 | 1 | 60 | 19.2 | 100.0 % | 116.8 | 2.63 ± 0.05 | 0.66 |
| 8 | cubic | 1:3:3:1 | 56 | 3 | 4 | 60 | 57.6 | 100.0 % | 247.2 | 1.24 ± 0.03 | 0.31 |
| 8 | twisted | 1:3:3:1 | 56 | 3 | 4 | 60 | 57.6 | 95.8 % | 236.8 | 1.24 ± 0.03 | 0.31 |
| 8 | folded | 1:3:3:1 | 56 | 2 | 4 | 60 | 57.6 | 62.5 % | 154.3 | 1.24 ± 0.03 | 0.31 |

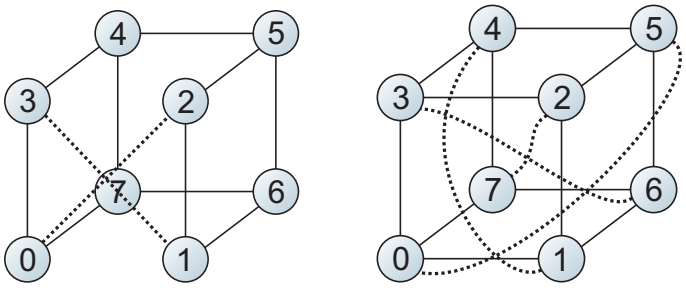
Table 5.16: Mapping results of homogeneous pseudo-random networks on the hardware topologies of the binary cube and its modifications. The additional connections can be achieved with the four top-connectors of the network modules.

*modifications of the cubic topology*

**The Folded and the Twisted Hypercube** The last three mappings of table 5.16 concern mapping results of 8-chip homogeneous pseudo-random networks on modified cubic topologies, namely the *twisted hypercube (THC)* and the *folded hypercube (FHC)* (for an overview see e.g. [106]). The THC features the same number of links, but has selected edges exchanged and features a lower diameter. The FHC adds an additional link between the farthest nodes of the cubic topology. Figure 5.11 illustrates the two topologies. Table 5.17 compares the hop distribution of the modifications against the regular cube.

*results*

Compared to the cubic topology, the THC does not allow a fewer number of time slots per reservation period and does not improve the connection jitter or the average neuron frequency. However, it requires the same amount of physical links and keeps a few slots free for best-effort traffic. The mapping on the FHC achieves the same QoS results by using only two third of the bandwidth and with a local port less, but indeed requires an additional global port at each node.




(a) the twisted hypercube (b) the folded hypercube

Figure 5.11: Modifications of the 8-node cubic topology, illustrated as dotted lines.

| n | connect. | global ports | internal | 1 hop | 2 hops | 3 hops |
|---|---|---|---|---|---|---|
| 4 | cubic | 2 | 4 | 8 | 4 | 0 |
| 4 | fully connected | 3 | 4 | 12 | 0 | 0 |
| 8 | cubic | 3 | 8 | 24 | 24 | 8 |
| 8 | twisted | 3 | 8 | 24 | 28 | 4 |
| 8 | folded | 4 | 8 | 32 | 24 | 0 |

Table 5.17: Comparison of the hop distribution of regular, twisted and folded hypercubes. The values denote the number of inter-chip connections with different hop distances.

*simulation of link failures*

**Mapping Robustness against Single Link Failures** The last topic concerns the robustness of the mapping algorithm against single broken links or against network asymmetries in general. The mapping tests have been made with the homogeneous pseudo-random 16-chip network with all inputs, neurons and synapses used. The selected mapping parameters leave a remaining physical bandwidth of 20 % at an intact cubic topology. The network has been mapped to the backplane topology, at which single links in different cube-dimensions have successively been deactivated.

*results*

The results are shown in table 5.18 and proof that the implemented mapping algorithm is robust against multiple link failures. Even in the case that three links have been removed completely, the mapping results are not affected significantly compared to the original mapping on the intact cubic topology.

| n | hardware topology | H | $p$ | $m$ | $f$ | $J_c$ [ns] | reserved slots | $\overline{L}_e$ | $\overline{w}_n$ [MB/s] | $\overline{\nu}$ [MHz] |
|---|---|---|---|---|---|---|---|---|---|---|
| 16 | cubic | 1:4:6:4:1 | 3 | 10 | 10 | 134.4 | 80.0 % | 250.7 | 0.91 ± 0.02 | 0.23 |
| 16 | 1 fail | 1:4:6:4:1 | 3 | 10 | 10 | 134.4 | 81.6 % | 255.7 | 0.91 ± 0.02 | 0.23 |
| 16 | 2 fail | 1:4:6:4:1 | 3 | 10 | 10 | 134.4 | 83.2 % | 260.7 | 0.91 ± 0.02 | 0.23 |
| 16 | 3 fail | 1:4:6:4:1 | 3 | 10 | 10 | 134.4 | 84.9 % | 266.0 | 0.91 ± 0.02 | 0.23 |

Table 5.18: Demonstration of the robustness of the mapping algorithm. Even after three links have been removed, the mapping algorithm finds a valid mapping with nearly the same QoS results compared to the intact regular cubic topology.

### 5.5.5 Summary

*main features of the mapping algorithm*

The investigation of the performance of the connection mapping algorithm for different neural networks (netlists) and different hardware topologies can be concluded to the following main results:

- The mapping algorithm is able to map arbitrary netlists on arbitrary hardware topologies.
- The mapping algorithm achieves 100 % slot usage and thus 100 % bandwidth efficiency for all homogeneous pseudo-random networks up to 16 chips if mapped on $d$-dimensional cubic topologies. 32-chip networks can be mapped with 80 % efficiency.
- The mapping algorithm can by widely parameterized. An example is to trade the number of local ports (the logic consumption) against the bandwidth efficiency.
- The mapping algorithm is stable and robust to map networks that differ from the regular cubic topology of the backplane. This has been demonstrated with mappings on feed-forward topologies, extended backplane topologies, modified cubic topologies and also cubic topologies with up to three missing links.

*resulting neuron frequencies*

Concerning the resulting mean frequencies $\overline{\nu}$ of the single neurons, it has been shown that the mapping algorithm is able to achieve the calculated theoretical maximum of 305 kHz of section 5.4.3 for the networks for which it occupies all slots. The algorithm therefore succeeds to route all connections along the shortest possible path through the network. Higher values for $\overline{\nu}$ require to reduce the input count $I$ or to modify the hop ratios of the several networks.

## 5.6 High-Level Simulation of the Neural Data Transport

The previous sections discussed the performance of the connection mapping process for pseudo-random networks and their modifications. The performance of the algorithm has been measured according to the mean bandwidth $\overline{w}_n$ that is available for single neurons and its resulting possible mean spike frequencies $\overline{\nu}$. The values for $\overline{w}_n$ and $\overline{\nu}$ can be calculated due to the deterministic behavior of the transport network that guarantees the throughput as long as the slot admission policy at the source node of the isochronous connections is observed.

*effect of ANN controller*

This section verifies the calculations of the previous sections with a high-level simulation of the transport network that further includes the application functionality, namely a simulation model of the ANN chip `Spikey` and of its controller with an interface to the transport network. Drops of neural spike events can occur at both controller interfaces: to the ANN chip and also to the transport network. The possible frequencies of the single neurons therefore depend on the bandwidth of both interfaces and on the implemented queuing and scheduling policies within the controller (cf. figure 5.12). The effect of the ANN controller is discussed here exemplarily with a single simulation. The results can be transferred to other networks and hardware topologies as well.

*cycle-accurate high-level simulation*

The simulation is performed with the C++ software from [47]. The software

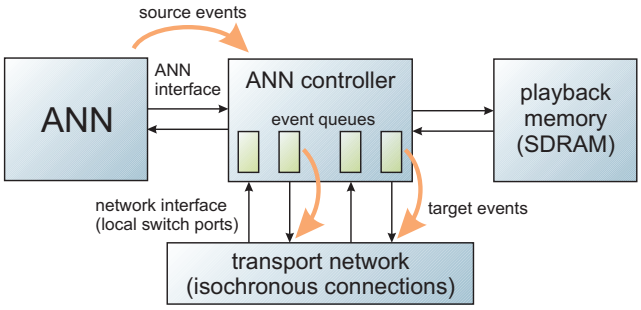



Figure 5.12: Data path of the neural spike events. Drops can occur at both controller interfaces.

| | |
|---|---|
| hardware topology | full backplane, 16 chips, 4-dim. binary cube |
| neural netlist | 16-chip, pseudo-random |
| neural elements | `Spikey` chip, 6144 neurons, 1572864 synapses |
| input count per block | 256 |
| hop ratios | 10:4:6:0:0, ca. 50 % off-chip connections |
| mapping parameters | 3 local ports, 5 slots/rsv. period, 60 slots/frame |
| slot usage | 80 % |
| mean bandwidth/neuron | $\overline{w}_n = 2.52 \pm 0.10$ MByte/s |
| max. mean neuron frequency | $\overline{\nu} = 0.63 \pm 0.03$ MHz |

Table 5.19: Parameters of the simulated neural network experiment.

generates spike events for the modeled neurons independently for each neuron with a certain probability $P$ within each simulated clock cycle. The simulation is cycle-accurate and uses the routing tables that have been generated by the connection mapping algorithm software and which are also used to configure the switches of the transport network during the network initialization phase (cf. section 4.10.2). The parameters of the used hardware topology, netlist and mapping are listed in table 5.19

*description of the simulation*

The simulation sweeps the parameter of the spike probability $P$ for single neurons per chip cycle from 0 to 1 %. For each value of $P$, the simulation executes 10000 cycles in which the generated spike evens are transferred via the transport network between the chips. Since the *source* events of some neurons have to be transported to multiple destinations within multiple connections, the ANN controller at each chip generates a separate *target* spike event for each connection and transmits it via the network.

*calculation of expected spike rates*

The spike probability per chip cycle at which the reserved bandwidth is exceeded can be calculated as follows: Table 5.19 shows that the reserved slots are fully occupied with a mean spike frequency of 0.63 MHz. Higher frequencies cause drops at the network interface at the source nodes of the connections. The value is calculated by the mapping program as an average over all connections. The event generation of the `Spikey` chip uses a 312.5 MHz clock (the 400 MHz clock from [47]). The event probability per neuron within a chip cycle therefore calculates to:

$$P = \frac{\overline{\nu}}{312.5\ \text{MHz}} = 0.0020 = 0.2\,\%. \tag{5.22}$$

Note that this value is calculated initially by considering the load of all connections. The effect that the controller generates multiple target events for single source events from particular neurons is already included in the number. This also means that the effective number of spikes that is transferred via the ANN interface to the controller is smaller. Since roughly $50\,\% = 192$ of the neurons transmit data off-chip, the interface has to transfer

$$N \ll P \cdot 192 = 0.384 \ \frac{\text{events}}{\text{chip cycle}}. \tag{5.23}$$

This is less than the available physical transfer rate of the ANN interface of 1.5 events per chip cycle at maximum [47]. The event rate of the single neurons of this setup is therefore limited by the physical capacity of the transport network and not by the ANN interface of the controller.

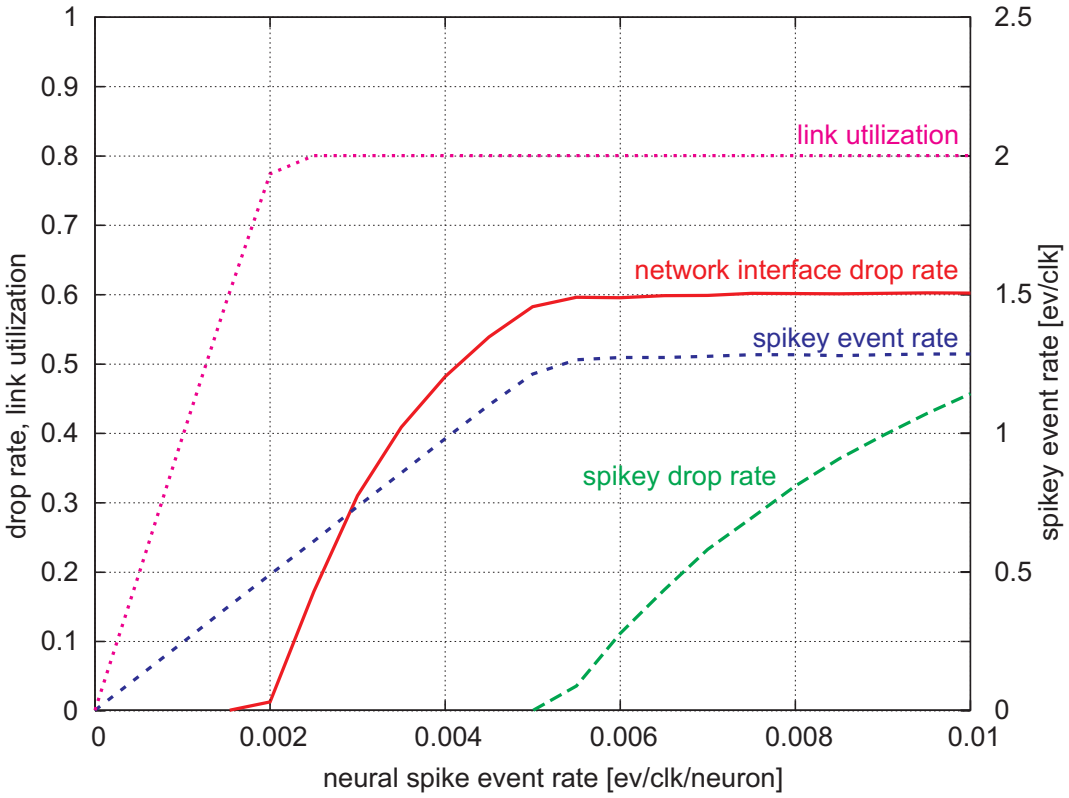



Figure 5.13: Simulation results of the event drop rates at both interfaces. The reserved bandwidth of the transport network is reached at the neuron spike probability of about 0.002 per chip cycle.

*results*

The result of the simulation is illustrated in figure 5.13. It can be seen that an increase of the spike probability $P$ per neuron first results in a linearly increased spike rate and an increased bandwidth usage as expected. The reserved bandwidth is reached at a probability of 0.002 events per chip cycle, which exactly matches the above calculation. Further spikes result in drops at the network interface. The slot usage of the network does not exceed the reserved value of 80 %. The drops at the ANN interface start at higher probabilities. The exact value depends on the

probability that the ANN can put three spikes into a single 64 bit packet. This issue is not further discussed here. The iterested reader may refer to [47].

*bursting and synchronized behavior*

It is finally noted that the neurons of a real experiment do not spike with fixed probabilities, but the networks experience a bursting of single neurons and also a synchronized behavior of multiple ones [14]. The values calculated above are only mean values calculated over all connections each with multiple neurons. The required bandwidth depends on the experimental setup and the network stimuli at runtime. The upcoming implementation of the ANN controller in programmable logic should monitor the drop rates to give feedback to the experimenter. Further experiments using the simulation software including bursts can be found in [47, 111].

## 5.7 Performance of the Best-Effort Schedulers

*motivation*

The implemented transport network serves packet-based data as best-effort data in a non-deterministic process. In contrast to connection-based data, which is forwarded within reserved time slots, the packets are first stored in the VOQs of each intermediate switch. The forwarding decision which packet to be served is made at runtime by the central best-effort scheduler that calculates the matching for the central crossbar of the switch (cf. section 4.6). The development of the transport network required to select a certain queuing policy, scheduler type and further parameters like the packet sizing for the implementation of the transport network within programmable logic.

*packet switch simulator*

For that reason, the simulator software `switchtest` has been developed. Is has already described in section 4.10.5. This section addresses only the simulation feature of the software. The purpose of the software is to simulate an input-queued packet switch with VOQs and to investigate the performance of different schedulers under various traffic conditions. The bypass-switch of the transport network is a sub-type of the supported switch architectures. The following sections discuss the simulation results to motivate the designs decision for the implemented schedulers. The investigated schedulers are all crossbar schedulers, i.e. they calculate a bipartite matching out of the $N \times N$ requests from the queues and select up to $N$ of them:

*simulated schedulers*

1. Parallel matching schedulers: PIM, RRM and iSLIP with one and multiple iterations.
2. 2-dimensional schedulers: the ripple-carry, RPA and DPA.
3. The MSM scheduler as the reference with almost optimal performance.

The schedulers that have been implemented in programmable logic are iSLIP, RPA and DPA due to its comparably good performance, the fair scheduling and the limited online complexity (cf. section 4.6).

### General Performance of the Schedulers

*comparison to literature*

The first simulation has been made to compare the general performance between the different schedulers and to verify the results with the literature. The simulation uses a slotted timing with a single packet per time slot and no reserved traffic. The queue-size has been set to 1000 packets to eliminate packet drops. Figure 5.14 shows the resulting average packet delays and the overall throughput depending on the

traffic load for a 16-port switch with Bernoulli i.i.d. arrivals with 100000 simulated time slots (the number of iterations of the iSLIP scheduler are noted behind the term *iSLIP* in the figure). The resulting values match the values from the literature, e.g. as published in [84].

*average delays*

It can be seen in the figure that the values for the average packet delay increase significantly with loads close to 100 %. Since the iSLIP scheduler is an iterative matching scheduler, its performance clearly depends on the number of iterations. Using two iterations, the average delay is significantly lower for higher loads by approximately a factor of 10 compared to the single iteration. Using four iterations, the performance is comparable to that of RPA and DPA, which both perform well. The MSM scheduler performs best, but has the far highest complexity and can hardly be implemented in programmable logic. In contrast to the average delays, the overall throughput is near the optimum of 100 % for loads up to 100 % for all schedulers.

*three schedulers available in hardware*

The following discussion considers only the three schedulers that are available within the programmable logic: this is iSLIP with a single iteration, RPA and DPA. Multiple iterations for iSLIP are unavailable in the present design due to the small duration of a time slot $S$ of only two clock cycles. The ripple-carry arbiter cannot be used due to its unfairness and MSM is to complex to be implemented in hardware.

### Performance within the Transport Network

*parameter set for the transport network*

The bypass-switch without reserved traffic is effectively an input-queued switch with VOQs. The above simulations have been repeated with typical values within the transport network for the number of ports, the queue size and the packet length. The number of ports has been set to five to simulate four global ports and a single local port of the switch. The queue sizes have been set to four, which allows to use a single DPBRAM element at each VOQ (cf. section 4.5.3). The packet size has been set to 11 slots per packet according to the present DSM implementation. Figure 5.15 shows the simulation results.

*results*

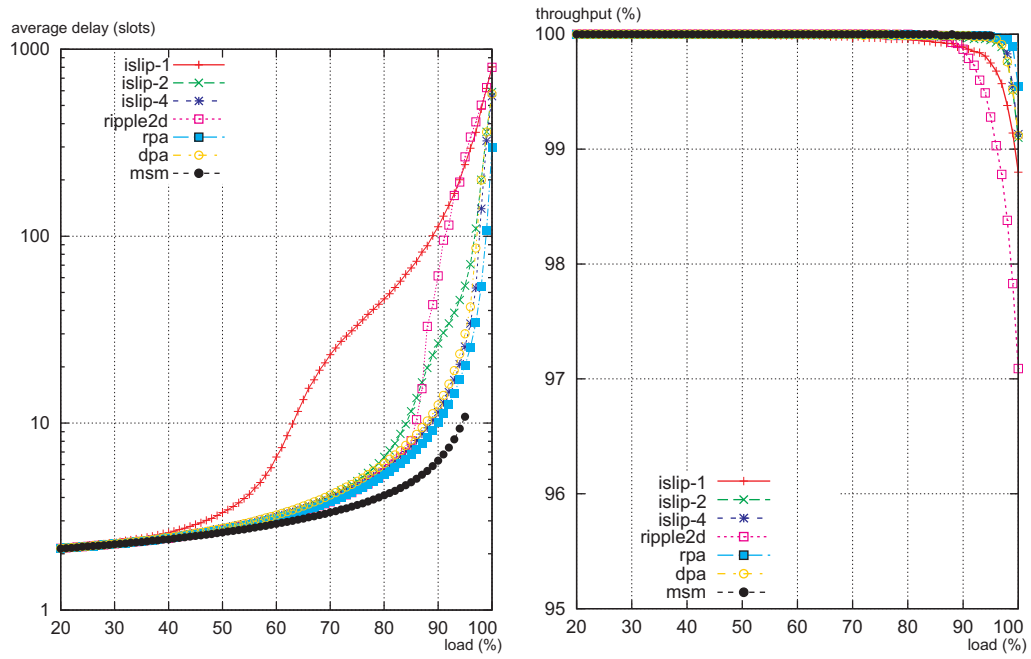
It can be seen from the figure that the schedulers perform similar to each other, but worse than in the first simulation. The reason is that the switch architecture does not merge packets from different inputs sequentially to the same output. As long as a packet is transferred via the crossbar, the corresponding input as well as the output are blocked for the scheduling for other ports within each time slot. In the case that many ports are blocked, the complexity of the scheduling task is reduced such that the remaining ports are scheduled similar by the different schedulers. The average delays are at least 11 slots according to the selected packet size.

### Influence of the Queue Sizes

The switch design allows to adjust the number of DPBRAM to be used for a single VOQ. Since the used FPGA provides only 44 of the memory blocks, a trade-off has to be made between the performance of the best-effort packet transfers and the logic consumption. Figure 5.16 shows simulation results for the average delays and the throughput for queue sizes of 16 packets per each input/output port combination of the VOQs. The values have to be compared to figure 5.15.

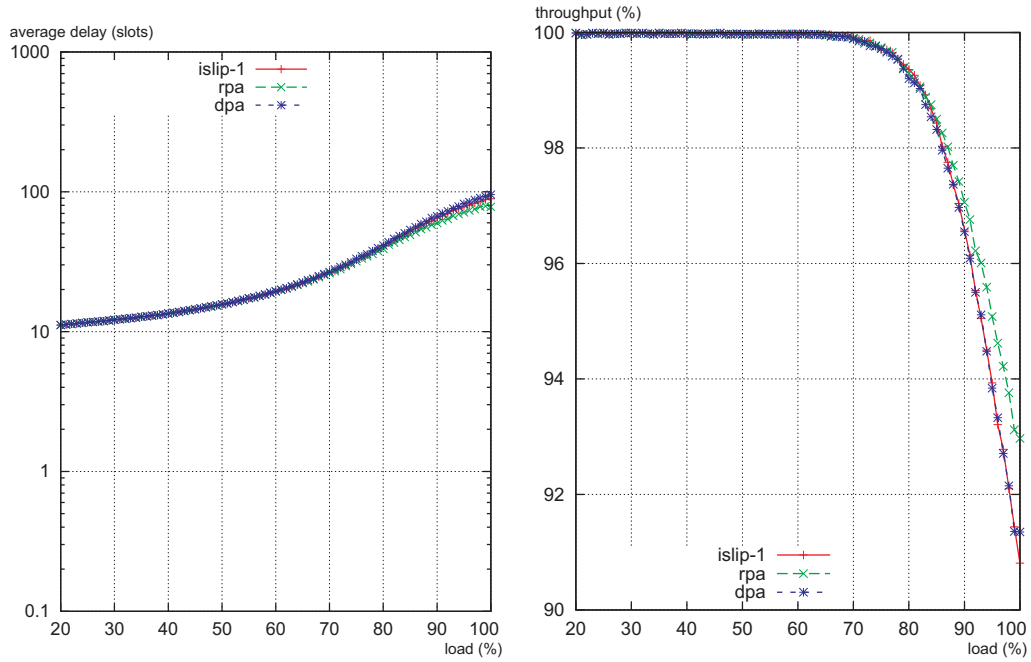
*trade delay against loss rate*

It can be seen that for high loads, the average delay is much better for smaller




(a) average delay (b) throughput

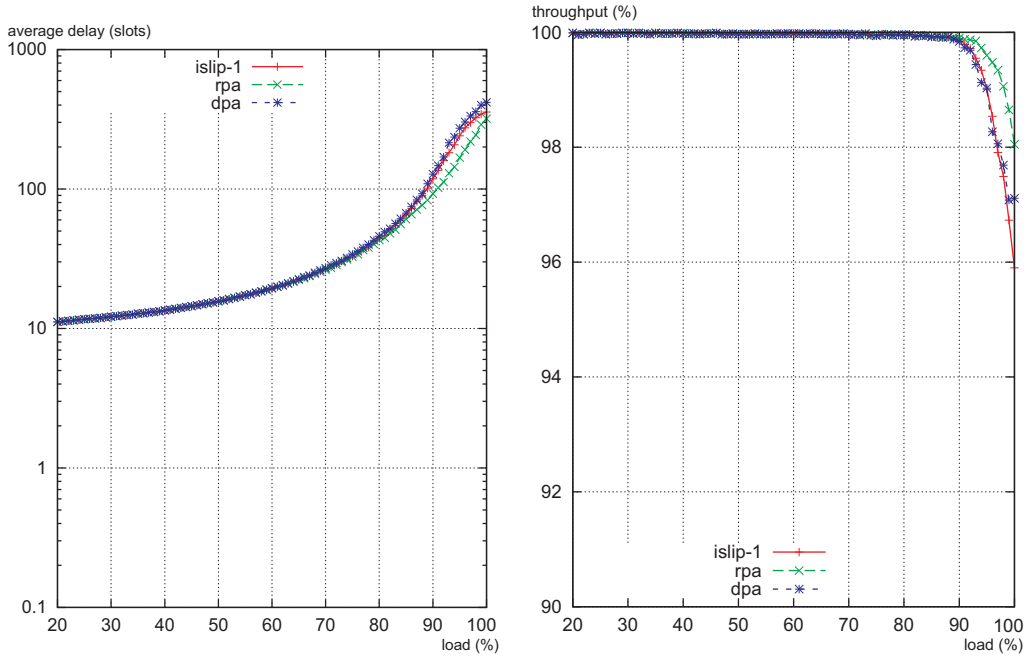
Figure 5.14: Simulations of the average delay and the overall throughput for different schedulers for a 16-port switch with uniform i.i.d. Bernoulli arrivals. The simulation comprises 100000 time slots and a queue size of 1000 packets. A single packet is transmitted per slot.




(a) average delay (b) throughput

Figure 5.15: Simulation of the schedulers with typical parameters of the transport network. The simulation uses 5 ports per switch, 4 packets per queue and 11 slots per packet.

queue sizes in contrast to large queues. This is because packets that are stored in large queues may have already been be dropped by smaller queues and thus do not increase the average delay. Smaller queues therefore feature a higher drop rate and thus a reduced throughput for higher data rates. Since the avoidance of drops leads to delays in the order of several hundred of slots, it can be stated that large queues to not necessarily lead to an improved performance, which is in accordance with the observations from [98]. Furthermore, this shows that the ARQ algorithm within the transport layer has to cope with delays in the range of hundred and more time slots under heavy traffic for switches at the given parameters (cf. section 4.9.6).




(a) average delay (b) throughput

Figure 5.16: Simulation of the schedulers with typical parameters of the reference implementation, but with increased buffer sizes of 16 packets per queue.

### Interaction with reserved Traffic

*40 % background reservation*

In the case that guaranteed traffic as well as reserved traffic is used, the arrival of the packet data is periodically modulated with the reservation patterns. To study this effect, the simulation has been performed with 40 % of the time slots periodically occupied by reserved traffic. The previous parameters have been kept to five ports, four packets per queue and 11 slots per packet. This means that two input ports and two output ports of the switch are not available for best-effort traffic at each time slot. The results are shown in figure 5.18

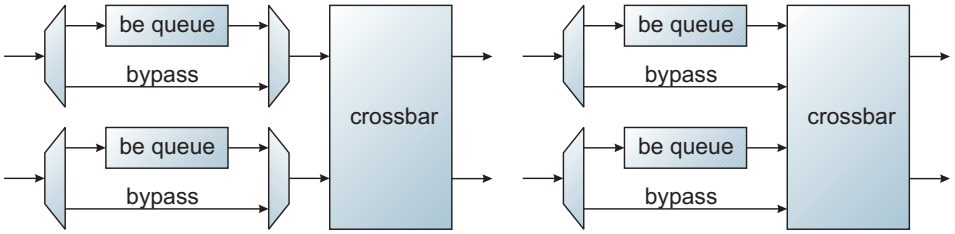
It can be seen that the performance of the different schedulers converges to the same values for the average delay and the throughput. This is since the traffic constraints introduced by the unreserved slots are such strict that the scheduling task for the remaining slots is simple and is handled similarly by each of the schedulers.

Consequently, a switch design with a high amount of reserved slots does not require a complicated scheduler. A low-complex scheduler is also feasible as long as the fairness is preserved.

### Effect of the Bypass and the Crossbar Inputs

*separate crossbar inputs for both traffic classes*

The last topic to be investigated is the influence of the bypass in conjunction with the doubled number of inputs of the crossbar, which is one of the key ideas of the novel switch design. Since reserved traffic as well as best-effort traffic use a separate set of inputs to the crossbar, contention at the crossbar inputs between the two classes are completely removed. This feature is different compared to other network architectures that also service combined QoS classes with slotted bandwidth, but use a conventional $N \times N$ crossbar as e.g. [45]. The difference between both architectures is again illustrated in figure 5.17.




(a) conventional $N \times N$ crossbar (b) $2N \times N$ crossbar of the bypass switch

Figure 5.17: The bypass-switch avoids queuing of priority traffic and features separated inputs to the crossbar for each traffic class. The example shows a two-port switch.

*significant improvement*

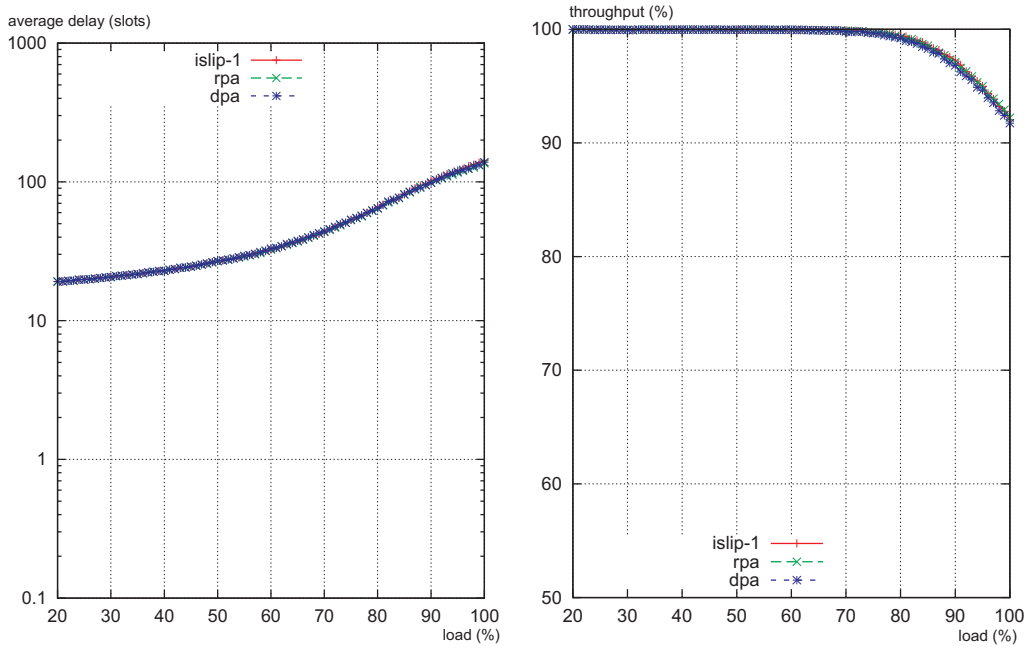
Figure 5.19 shows the results. It can be seen that the performance of the bypass-switch with separate crossbar inputs significantly outperforms the conventional design with multiplexers before the crossbar inputs. This holds true for both, the average delay as well as the throughput. Note that a doubling of the number of inputs does only increase the size of the crossbar by the factor of 2 and not quadratically. Furthermore, the performance improvement for best-effort traffic does not reduce the guarantees made for priority traffic.

## 5.8 Summary

In this chapter, the implementation of the transport network for the FACETS Stage 1 framework has been evaluated. The discussions showed that the implemented network is suitable for the execution of experiments with hardware neural networks.

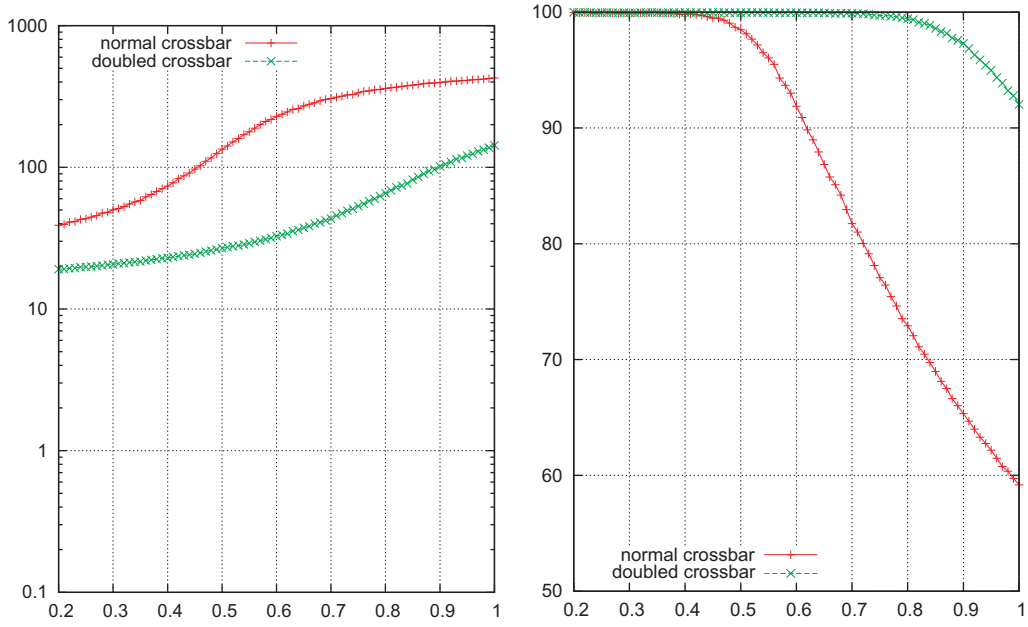
*isochronous connections*

Since neither experimental networks nor a network-compatible ANN controller has been available, the suitability of the network for the desired experiments has been discussed according to the mapping results for different generated neural network topologies. The mapping of the test-networks to the hardware proved the efficiency of the proposed bandwidth reservation algorithm. The evaluation of dedicated qualifiers for the generated network topologies allowed to predict maximum physical mean frequencies for the single physical neurons that can be handled by the




(a) average delay (b) throughput

Figure 5.18: Average delay and throughput for 40 % of the bandwidth reserved for priority traffic (5 ports, 4 packets per queue, 11 slots per packet)




(a) average delay (b) throughput

Figure 5.19: Comparison of the scheduling results for switch designs with and without separate crossbar inputs for the service classes. It can be seen that the novel bypass-switch architecture outperforms the standard design.

transport network. The calculated values have been verified with a cycle-accurate software simulation of spike transmissions via isochronous connections within a 16-chip network. The transport network is able to provide a stable framework-wide synchronization of all network nodes. The deterministic transport of isochronous test-data has been verified by means of several hundred millions of cycle-accurate transfers.

*best-effort transfers*

The last section discussed the bypass switch and the performance of the implemented scheduling algorithms. A software simulation of the switch showed that a low-complex scheduler and small buffers are not necessarily a drawback in the given environment, which allows to save programmable logic. It has further been shown, that the usage of separate crossbar inputs for the two service classes as done within the bypass-switch outperforms conventional switch architectures that also provide different service classes.

# Conclusion and Outlook

This thesis addressed the design and implementation of the novel MCGN network architecture that manages the provision of two fundamentally different levels of QoS within an embedded environment: isochronous connections and best-effort packet transfers. The network has been implemented on the FACETS Stage 1 framework as the transport network for hardware ANNs. The main challenges of the development have been: *main challenges* *main challenges*

1. The end-to-end delay of the isochronous connections should not only be nearly constant, but even more important, also the *absolute value* of the delay should be as low as possible to keep to the timing requirements of the used neuron models and the speedup of the ANN chips.
2. Requests for on-demand transfers of high-level data during the experiments require additional best-effort packet transfers. Isochronous connections cannot be used since connections cannot be set up at runtime due to their complexity and the latency of the setup.
3. The online part of the network has to be implemented within the limited space of the programmable logic of the FPGAs on the network modules.

The proposed switching architecture fulfills these requirements with basically four techniques: a slotted TDM framing scheme of the bandwidth, a global cycle-accurate synchronization, an efficient bandwidth reservation algorithm and the novel bypass-switch architecture for the multiplexing and demultiplexing of the two traffic classes. The main features of MCGN are: *main features*

1. The provision of services for two different traffic classes: isochronous connections and best-effort packets multiplexed on the same physical medium.
2. Isochronous data is served in-order with guaranteed throughput, bounded delay and bounded jitter.
3. The avoidance of queuing of isochronous data results in a minimum end-to-end delay of only a few clock cycles plus the physical layer delay per network node.
4. The bypass-switch has a hierarchical and modular design to select the best-effort packet queuing discipline and the crossbar scheduler best adopted to the problem.
5. Both traffic classes support multiple protocols as payload within the same network.

6. The global synchronization strategy allows arbitrary frame sizes and thus bandwidth granularities even on distributed setups with different physical link delays.
7. The bandwidth reservation algorithm achieves up to 100 % slot usage even for several hundreds of isochronous connections.
8. The low online complexity of $O(1)$ for isochronous data supports tiny slot sizes for a fine-grained bandwidth division.
9. The design is scalable in terms of line speed (no internal speedup) and the number of network hops (bounded delay). The scalability in the number of ports solely depends on the complexity of the selected best-effort scheduler.
10. The compact design is suitable for an implementation within the limited space of programmable logic.

*reference implementation in hardware and software*

A reference implementation of the proposed architecture has been presented within the programmable logic of the Stage 1 framework. The algorithmic parts of the network initialization phase are implemented in software. The implemented VHDL design of the switch is parameterizable for different packet sizes, port numbers, best-effort schedulers and also slot sizes.

### QoS Results for Isochronous Connections

*deterministic forwarding*

Since isochronous connections use reserved bandwidth, the throughput is strictly guaranteed as long as the reserved bandwidth is not exceeded at the source node of a connection. It has been shown that delay and jitter can be calculated in advance due to the deterministic forwarding process.

*minimum end-to-end delay*

The end-to-end delay is bounded and has been reduced to the near minimum by totally avoiding the queuing of isochronous data. The problem of contention between the switch ports has been solved without introducing additional delays. The proposed solution uses a contention-free reservation strategy during the network initialization phase. The resulting delay is only a few clock cycles due to the synchronization adjustment. Concerning the reference application, it has been shown that the main part (about 80 %) of the delay is caused by the already delay-reduced MGTs of the FPGAs. A further delay reduction would therefore require a different physical layer. The calculated delay values have been exactly reproduced by measurements of the reference implementation.

*bounded and small end-to-end jitter*

The connection jitter is bounded by at maximum a single frame time. The jitter mainly depends on the waiting process for reserved slots at the source interface. It is possibly slightly enlarged by intermediate network nodes by the size of the frame gap. It has been shown that for regular network topologies with comparable transmission times and a single slot per connection, the end-to-end jitter is *independent* of the number of network nodes. The best possible jitter that has been achieved for various neural network topologies is only 19.2 ns.

*demonstrator application*

Since the network interface of the ANN controller has currently not been finished, the isochronous transfers have been verified using a demonstrator application in programmable logic. The calculated delay values for the isochronous connections have been reproduced exactly. Long-term transfer tests showed a mean 32 bit error rate of $2.5 \cdot 10^{-8}$ due to sub-optimal electrical properties of the physical transmissions.

### Global Synchronization

*synchronization precision*

The forwarding scheme of MCGN relies on the global synchronization of all network nodes. The developed technique is suitable for synchronizing a distributed setup of network nodes to the precision of a single clock cycle. A synchronization service has been developed that allows applications to communicate via globally synchronized signals with this precision.

*arbitrary frame sizes possible*

Since queuing stages are avoided, synchronization would usually constrain the selectable frame sizes according to the physical network topology. This problem has been solved by the development of the *shifted framing* technique, which allows arbitrary frame sizes independent of physical transmission times. This is of importance since it allows a fine-grained bandwidth division to be selected that is best adopted to the application (e.g. the neural network topology to be investigated) and thus leads to optimal bandwidth usage.

*stability*

The global synchronization of all network nodes has been successfully established within the Stage 1 framework down to the precision of a single clock cycle of 6.4 ns. The synchronization state has been measured as being stable during experiments with a duration of at least 12 h. This is sufficient for the application since single neural network experiments are expected to run only fractions of seconds.

### Bandwidth Reservation

*transformation to vertex color problem*

The developed MCGN architecture reserves bandwidth resources by means of a connection mapping algorithm. It is executed in software during the network initialization phase prior to the network operation. This reduces the online complexity of the forwarding of the isochronous data to a simple table look-up. The main algorithmic problem to be solved is the contention at the switch outputs during the forwarding process. It has been shown that this problem equals the common *vertex color* problem from graph theory, which is well studied.

*100 % slot occupancy in many cases*

The performance of the implemented connection mapping algorithm has been evaluated on a number of reference topologies. It has been demonstrated that the algorithm achieves 100 % slot usage for all tested network topologies for up to 16 chips, which equals a near-optimum usage of the available bandwidth (except the frame gap). The algorithm even scales with higher chip numbers: a pseudo-random network of 32 chips and 992 isochronous connections has been successfully reserved with 80.1 % slot occupancy. The developed solution can therefore be expected to outperform other approaches that use a greedy algorithm as e.g. proposed in [117] for the run-time slot allocation. It has further been shown that the developed algorithm does not rely on symmetric topologies. Tests with up to three missing links or twisted links showed that performance deceases only slightly.

### Best-Effort Packet Transfers

*development of a novel switch architecture*

The proposed network architecture provides packet-based transports as best-effort services. Packets occupy unreserved or unused slots. The multiplexing of reserved traffic from isochronous connections and best-effort traffic is performed by the novel bypass-switch architecture. The switch allows a separate processing of isochronous data and best-effort packets in a hierarchical design. Contentions at the crossbar

inputs between both classes are removed by using separate inputs. Simulations showed that this significantly improves the performance of best-effort transfers in the presence of reserved traffic.

*development of a shared memory system*

The packet-based part of the bypass-switch equals an input-queued switch with VOQs and a centralized crossbar scheduler. Due to its modularity, it is possible to trade the logic consumption against the switch performance in terms of buffering size or the scheduler complexity. The suitability of different schedulers has been investigated by a slot-based simulation of the proposed switch architecture. The scheduling algorithms iSLIP, RPA and DPA have exemplarily been implemented in hardware. The simulation results have been used to verify the hardware implementation for correctness.

*simulation and verification of the packet schedulers*

The packet-based transfers are the basis for a global DSM subsystem. Although the DSM currently lacks a cache coherency protocol, it provides a convenient method for the global on-demand data exchange usable by all clients within the programmable logic. The implemented routing logic exploits the 4-dimensional cubic topology of the backplane and performs a simple and efficient dimensional routing algorithm.

### Analysis of Applicable Neural Network Topologies

*defined qualifiers on generate topologies*

The performance of the mapping algorithm has been evaluated with customized generated neural network topologies, which are mappable to the network blocks of the `Spikey` ANN chips without losses of synapses or neurons. The evaluation has been done with a set of defined network qualifiers to analyze the resource consumption and connectivity of the generated neural network topologies.

*calculation of mean link load*

The investigated networks feature a pseudo-random topology with the inputs of the ANN network blocks evenly connected to neurons from all chips. As a remarkable result, it has been calculated that for homogeneous pseudo-random networks and for a larger number of chips, the mean number of neurons $L_e$, whose data have to be transported via a single physical link converges against the input count $I$ of the ANN chips and is particularly independent of the total number of neurons or the number of chips used.

*calculation of max. mean neuron frequency*

It has further been calculated that the mean physical spike frequency for a single neuron, which can be handled by the network without dropping data equals $\overline{\nu}$=0.31 MHz in the case of optimal mapping. Although this value does not necessarily allow to predict a possible speedup to be adjusted for the `Spikey` chips, it can be used as a reference when planning large-scale experiments with multiple chips on the Stage 1 framework.

*values for modified topologies*

If has finally been demonstrated how a change in the input count or a change in the network hop ratios affects the neural network topologies and the finally possible physical mean frequencies $\overline{\nu}$ of the single neurons. The avoidance of inter-chip connections with a large number of hops allows networks of 32 chips to be mapped on the Stage 1 framework by using all available neurons and synapses with a mean frequency for a single neuron of $\overline{\nu}$=0.35 MHz.

### Future Work and Outlook

The presented calculations, measurements and simulations demonstrated the suitability of the MCGN architecture to serve as a transport network for the application of large-scale neural networks on the FACETS Stage 1 framework. The developed network supports multiple backplanes of ANN chips to experiment with thousands of neurons and millions of synapses. Future work on the implemented design within the programmable logic addresses basically two items:

*future work*

- The connections of the existing ANN controller to the transport network. This requires a re-design of the controller with a buffering strategy that is scalable for a large number of neuron connections with low online complexity. An initial approach has been published in [47].
- The improvement of the ARQ algorithm within the transport layer of the DSM subsystem. The present implementation is a first step. Although fully operational, it provides only limited performance.

A high-performance DSM system further allows the individual network modules of the framework to be interconnected to the control PC of the Stage 1 setup at higher speeds. One option is to transmit DSM packets via a gigabit Ethernet adapter, which is accessed by the FPGA on the backplane [67]. Further work on the MCGN specification has been suggested in section 3.10.1.

*application fields*

The proposed solution is not limited to ANN research, but is rather a general network architecture to provide different classes of QoS in an embedded environment. Customization for different applications is supported by the modular VHDL design and its number of parameters like slot duration, packet size etc. Possible applications that require isochronous connections are embedded systems with real-time requirements or integrated multi-media devices.

*next steps in the FACETS project*

In particular, the network architecture is also suitable for the upcoming Stage 2 hardware of the FACETS project, which uses wafer-scale integration of neural networks [121]. A single 20 cm wafer then comprises about 60 million synapses. The Stage 2 framework also requires a transport network for the interconnection of the wafers to be implemented within FPGAs. Its demands on the FPGA network are comparable to the first stage of the project. Due to the low online complexity of the presented network architecture, and due to its scalability in terms of line speed and the number of network hops, the proposed network architecture can be scaled to this application without major changes and is best suited to interconnect a wafer-scale setup of neural networks. This system will be a further step towards the final goal of understanding the functionalities of the human brain.

# List of Acronyms

| | |
|---|---|
| **ACK** | acknowledgment |
| **ANN** | artificial neural network |
| **ARP** | address resolution protocol |
| **ARQ** | automatic repeat request |
| **ASIC** | application specific integrated circuit |
| **ATA** | advanced technology attachment |
| **ATM** | asynchronous transfer mode |
| **ATX** | advanced technology extended |
| **BER** | bit error rate |
| **CBR** | constant bit rate |
| **CIOQ** | combined input-output queuing |
| **CMCR** | centralized multicast contention resolution |
| **CMOS** | complementary metal oxide semiconductor |
| **CPU** | central processing unit |
| **CRC** | cyclic redundancy check |
| **CSS** | connection synchronous signals |
| **DAC** | digital-to-analog converter |
| **DCM** | digital clock manager |
| **DCR** | device control registers |
| **DDR** | double data rate |
| **DRRM** | dual round-robin matching |
| **DLL** | delay-locked loop |
| **DPA** | diagonal propagation arbiter |
| **DPBRAM** | dual-port block RAM |
| **DRR** | deficit round-robin |
| **DSM** | distributed shared memory |
| **DWDM** | dense wavelength division multiplexing |
| **EDF** | earliest deadline first |
| **FCFS** | first-come first-served |
| **FHC** | folded hypercube |
| **FIFO** | first-in first-out |
| **FPGA** | field programmable gate array |
| **GPS** | generalized processor sharing |
| **GSS** | global synchronous signals |
| **HAGEN** | Heidelberg AnaloG Evolvable Neural network |
| **HOL** | head-of-line |
| **HRR** | hierarchical round robin |

| | |
|---|---|
| **HTTP** | hypertext transfer protocol |
| **IEEE** | institute of electrical and electronics engineers |
| **i.i.d.** | independent, identically distributed |
| **IP** | internet protocol |
| **IPP** | input packet processor |
| **ISDN** | integrated services digital network |
| **LAN** | local area network |
| **LFSR** | linear feedback shift register |
| **LPF** | longest port first |
| **LQF** | longest queue first |
| **LUT** | look-up table |
| **LUT4** | 4-input LUT |
| **LVDS** | low voltage differential signaling |
| **MCGN** | multi-class gigabit network |
| **MGT** | multi-gigabit transceiver |
| **MPLS** | multiprotocol label switching |
| **MSM** | maximum size matching |
| **MWM** | maximum weight matching |
| **NoC** | networks-on-chip |
| **MAC** | media access control |
| **NTP** | network time protocol |
| **OCF** | oldest cell fist |
| **OPP** | output packet processor |
| **OSI** | open systems interconnection |
| **OSPF** | open shortest path first |
| **PAL** | packet adaptation layer |
| **PAR** | positive acknowledgment with retransmission |
| **PCB** | printed circuit board |
| **PCS** | physical coding sublayer |
| **PCI** | Peripheral Component Interconnect |
| **PIM** | parallel iterative match |
| **PLB** | processor local bus |
| **PMA** | physical media attachment |
| **POP3** | post office protocol version 3 |
| **PPE** | programmable priority encoder |
| **PWWFA** | parallel wrapped wave front arbiter |
| **QoS** | quality of service |
| **RAM** | random access memory |
| **RPA** | rectilinear propagation arbiter |
| **RRM** | round robin matching |
| **RSVP** | resource reservation protocol |
| **RTT** | round-trip time |
| **SATA** | serial advanced technology attachment |
| **SCSI** | small computer system interface |
| **SDRAM** | synchronous dynamic RAM |
| **SERDES** | serializer deserializer |
| **SMT** | surface mount technology |

| | |
|---|---|
| **SoC** | system-on-chip |
| **SPS** | synchronous protocol stack |
| **SRAM** | static random access memory (RAM) |
| **SRR** | static round robin |
| **STDP** | spike time dependent plasticity |
| **SOF** | start-of-frame |
| **SOP** | start-of-packet |
| **TCP/IP** | transmission control protocol/internet protocol |
| **TDM** | time division multiplexing |
| **TFTP** | trivial file transfer protocol |
| **THC** | twisted hypercube |
| **TTA** | tiny tree arbiter |
| **UDP** | user datagram protocol |
| **VC** | virtual circuit |
| **VHDL** | VHSIC hardware description language |
| **VHSIC** | very high speed integrated circuit |
| **VLSI** | very large scale integration |
| **VOQ** | virtual output queue |
| **WAN** | wide area network |
| **WFA** | wave front arbiter |
| **WFQ** | weighted fair queuing |
| **WRR** | weighted round-robin |
| **WWFA** | wrapped wave front arbiter |
| **ZBT** | zero bus turnaround |

# List of Symbols

### Framing

| | | |
|---|---|---|
| $T$ | s | time frame duration |
| $S$ | s | time slot duration |
| $G$ | s | frame gap duration |
| $f$ | integer | time slots per frame |
| $n$ | integer | reservation periods per frame |
| $m$ | integer | time slots per reservation period |
| $w$ | bit/s | physical bandwidth of the links |
| $w_c$ | bit/s | bandwidth for connection $c$ |
| $w_s$ | bit/s | bandwidth per time slot |

### Synchronization, Timing

| | | |
|---|---|---|
| $D$ | s | transmission delay between inputs of adjacent switches |
| $D_<$ | s | transmission delay of the front part of the frame |
| $D_>$ | s | transmission delay of the back part of the frame |
| $D_e$ | s | transmission delay of link $e$ |
| $D_c$ | s | transmission delay of connection $c$ |
| $J_c$ | s | jitter of connection $c$ |
| $s$ | integer | logical slot shift between adjacent switches, $s \geq 0$ |
| $s_e$ | integer | logical slot shift at link $e$ |
| $\epsilon$ | s | value of the adjustable delay element, $\epsilon \geq 0$ |

### Mapping Parameters

| | | |
|---|---|---|
| $f$ | integer | time slots per frame |
| $p$ | integer | number of local ports |
| $m$ | integer | time slots per reservation period |
| $l$ | integer | maximum number of neurons per time slot of connection |

### Neural Network Topologies

| | | |
|---|---|---|
| $L$ | integer | network load, number of neuron outputs via instance |
| $L_e$ | integer | network load of physical link $e$ |
| $L_c$ | integer | network load of connection $c$ |
| $L_{tot}$ | integer | total network load |

### Spikey Chip

| | | |
|---|---|---|
| $I$ | integer | number of used inputs, $0 < \mathrm{I} \leq 256$ |
| $\nu$ | integer | event rate of a single physical neuron |
| $\nu_f$ | integer | maximum event rate at each physical link |

# List of Figures


1.1 Example network of 16 hosts and four subnets of different topologies 6
1.2 Illustration of the layered network architecture . . . . . 6
1.3 The seven-layer OSI reference model . . . . . 7
1.4 The hybrid network model used in this thesis . . . . . 10
1.5 Example of an Ethernet data frame that contains a TCP/IP packet . 11
1.6 Basic switch architectures . . . . . 17
1.7 Simplified schematic of an input-queued and an output-queued switch 18
1.8 Illustration of the head-of-line locking problem . . . . . 19
1.9 Operation of a round robin scheduler . . . . . 22
1.10 Illustration of the stop-and-go queuing discipline . . . . . 23
1.11 Illustration of the bipartite graph matching problem . . . . . 24
1.12 Crossbar schedulers with a parallel matching . . . . . 25

2.1 Photograph of the HAGEN chip . . . . . 31
2.2 Creation of multi-layered feed-forward networks using local feedbacks 33
2.3 Input sources of a HAGEN network block . . . . . 34
2.4 Photograph of Spikey die . . . . . 35
2.5 Schematic of the FACETS Stage 1 framework . . . . . 40
2.6 Photograph of the Nathan network module . . . . . 41
2.7 Photograph of the backplane . . . . . 43
2.8 Backplane topology . . . . . 44
2.9 Neural network experimental setup . . . . . 47
2.10 Large-scale interconnection of multiple ANN chips . . . . . 49
2.11 Pre-Experimental mapping process . . . . . 50
2.12 Clustering of netlists into network blocks . . . . . 51
2.13 Simplified schematic of the transport network . . . . . 55

3.1 MCGN protocol stack . . . . . 69
3.2 Network topology model . . . . . 71
3.3 Schematic of a general MCGN network node . . . . . 71
3.4 Illustration of the framing strategy of MCGN . . . . . 72
3.5 Illustration of the network initialization phase . . . . . 74
3.6 Schematic of the isochronous switch. . . . . 76
3.7 Contention resolution for reserved traffic . . . . . 77
3.8 Schematic of the global synchronization . . . . . 79
3.9 Upper-layer interface for connection-based traffic. . . . . 82

3.10 Synchronization condition at the switch inputs . . . . . 85
3.11 Illustration of a network node with time counter and delay elements 86
3.12 Switch timing scheme . . . . . 87
3.13 Synchronization . . . . . 89
3.14 Synchronization Constraints . . . . . 91
3.15 Synchronization with fixed framing . . . . . 93
3.16 Synchronization with fixed framing maximum frame size . . . . . 94
3.17 Shifted framing schematic . . . . . 94
3.18 Synchronization with shifted framing (unsynchronized local clocks) . 95
3.19 Synchronization with shifted framing (synchronized local clocks) . . 96
3.20 Synchronization with shifted framing at switch outputs . . . . . 97
3.21 Global synchronous signal example . . . . . 100
3.22 Illustration of the proposed GSS interface . . . . . 101
3.23 Connection mapping process flow . . . . . 102
3.24 Connection mapping example . . . . . 107
3.25 Slot assignment with fixed framing, connection collision graph . . . . 107
3.26 Slot assignment with shifted framing, illustration of collision . . . . . 109
3.27 Slot assignment with shifted framing, connection collision graph . . . 110
3.28 Calculation of connection jitter at the interface . . . . . 116
3.29 Influence of multiple reservation periods on the jitter calculation . . 117
3.30 Jitter calculation for a multi-slot reservation pattern . . . . . 118
3.31 Embedding a best-effort packet . . . . . 121
3.32 Best-effort packet format . . . . . 122
3.33 The bypass switch . . . . . 124
3.34 A virtual output queue . . . . . 125
3.35 Local connection-based interface of the bypass switch . . . . . 126
3.36 Upper-layer interface for best-effort packets . . . . . 127

4.1 Schematic overview of the reference implementation . . . . . 137
4.2 Frame Format . . . . . 139
4.3 Best effort packet format . . . . . 140
4.4 Global clock distribution . . . . . 143
4.5 Block diagram of the MGT . . . . . 144
4.6 Block diagram of the physical layer implementation . . . . . 145
4.7 Schematic of the MGT receive elastic buffer . . . . . 146
4.8 Local time distribution . . . . . 149
4.9 Synchronization sublayer: receive data path . . . . . 150
4.10 Synchronization sublayer: transmit data path . . . . . 150
4.11 Timing simulation of the shifted framing implementation . . . . . 152
4.12 Timing scheme of fixed framing synchronization . . . . . 154
4.13 Timing scheme of shifted framing synchronization . . . . . 154
4.14 Bit-meanings of the GSS header field . . . . . 156
4.15 Schematic of the GSS implementation . . . . . 157
4.16 Top-level block schematic of the implemented bypass-switch . . . . . 158
4.17 Implementation of the virtual-output queue . . . . . 161
4.18 Schematic of the switch core . . . . . 162
4.19 Schematic of the crossbar . . . . . 163

4.20 Timing diagram of the switch interface for isochronous traffic . . . . 166
4.21 Schematic of the implemented iSLIP scheduler . . . . . 168
4.22 Truth table of the TTA Element . . . . . 169
4.23 Implementation of the round-robin TTA scheduler . . . . . 170
4.24 Schematic of the 2-Dimensional Ripple Arbiter Cell . . . . . 172
4.25 Schematic of the Simple 2-Dimensional Ripple Carry Arbiter . . . . 173
4.26 Schematic of the 2-Dimensional RPA and DPA Arbiters. . . . . 174
4.27 Schematic of the demonstrator application for isochronous transfers . 182
4.28 Overview of the distributed shared memory subsystem . . . . . 184
4.29 Data format of a shared memory packet. . . . . 186
4.30 Schematic of the DSM client process . . . . . 188
4.31 Timing diagram of the user interface to the DSM client process . . . 188
4.32 Schematic of the DSM server process . . . . . 189
4.33 Block schematic of the DSM packet adaptation layer. . . . . 190
4.34 Illustration of the the stop-and-wait algorithm . . . . . 191
4.35 Illustration of the the stop-and-wait algorithm with timeout . . . . . 192
4.36 Schematic of the implementation of the stop-and-wait algorithm. . . 193
4.37 Synchronization order of adjacent network nodes . . . . . 200
4.38 Data flow of the mapping software . . . . . 202
4.39 Illustration of the generation of pseudo-random networks . . . . . 205
4.40 Inter-connectivity matrix of a 16-chip pseudo-random network . . . . 206
4.41 Block-level schematic of the high-level packet switch simulator . . . . 207

5.1 MGT error rates, sorted by parameters . . . . . 212
5.2 MGT error rates, sorted by physical units . . . . . 213
5.3 Measurement of the delays of isochronous connections . . . . . 217
5.4 Connectivity matrix of a 16-chip network with modifies hop ratios . 228
5.5 Distribution of connection loads for networks w. cubic hop ratios . . 230
5.6 Distribution of connection loads for networks w. non-cubic hop ratios 230
5.7 Connectivity matrix of a 16-chip network w. 10 % synaptic efficiency 231
5.8 Distribution of the number of dest. neurons, 100 % synapse efficiency 232
5.9 Distribution of the number of dest. neurons, 50 % synapse efficiency 232
5.10 Distribution of neurons/slot for different quantization limits . . . . . 240
5.11 Illustration of the twisted hypercube and the folded hypercube . . . 242
5.12 Data path of the neural spike events within the high-level simulation 244
5.13 High-level simulation results of event drop rates . . . . . 245
5.14 Delay and throughput for different best-effort schedulers . . . . . 248
5.15 Delay and throughput for the schedulers within the transport network 248
5.16 Delay and throughput for the schedulers with increased buffer size . 249
5.17 The bypass-switch and switches with conventional crossbar inputs . . 250
5.18 Scheduler results for 40 % bandwidth occupied for priority traffic . . 251
5.19 Scheduling results for switches with single or separate crossbar inputs 251

# List of Tables


1.1 QoS requirements for example applications . . . . . 13

2.1 Nominal specifications of the HAGEN chip . . . . . 30
2.2 Nominal specifications of the Spikey chip . . . . . 36
2.3 Comparison of the chips HAGEN and Spikey . . . . . 56
2.4 Data rates and network speed of the HAGEN . . . . . 57
2.5 Expected data rates and delay requirements for the Spikey chip . . . 58

4.1 Notation of the traffic class of the data slots within frames . . . . . 141
4.2 Configuration parameters of the MGT . . . . . 145
4.3 Theoretical values for the synchronization parameters, fixed framing 153
4.4 Theoretical values for the synchronization parameters, shifted framing 155
4.5 Logic consumption of different multiplexer implementations . . . . . 164
4.6 Implemented routing table for dimensional routing . . . . . 176
4.7 Dependency of neuron data per link and mean spike frequencies . . . 180
4.8 Description of the fields of the shared memory packet header. . . . . 187
4.9 Typical delay values of DSM transfers . . . . . 194
4.10 Latencies for DSM read requests . . . . . 195
4.11 Shared memory delay-bandwidth products for different hop counts . 196
4.12 Theoretical transfer rates of an optimal ARQ algorithm . . . . . 196

5.1 Measurement of the transmission delays $D_0$ before synchronization . 214
5.2 Measurement of the transmission delays after global synchronization 215
5.3 Measurement of the transmission delays $D$ after synchronization . . 216
5.4 Measurement of the delays of isochronous connections . . . . . 218
5.5 Experimental results for transmissions within isochronous connections 219
5.6 Properties of homogeneous pseudo-random networks on the backplane 226
5.7 Pseudo-random networks with modified hop ratios . . . . . 227
5.8 Pseudo-random networks with modified synaptic efficiency . . . . . 233
5.9 Mapping results of two-chip networks . . . . . 237
5.10 Mapping results of four-chip networks . . . . . 237
5.11 Mapping results of eight-chip networks . . . . . 238
5.12 Mapping results of 16-chip networks . . . . . 238
5.13 Mapping results of 32-chip networks . . . . . 239
5.14 Mapping results of non-intrinsic hop ratios with slot-load limitation . 239
5.15 Mapping results of feed-forward networks . . . . . 241

5.16 Mapping results for extended hardware topologies . . . . . 241
5.17 Hop distribution of regular, twisted and folded hypercubes . . . . . 242
5.18 Robustness of the mapping algorithm . . . . . 242
5.19 Network parameters of the high-level simulation . . . . . 244

A.1 Resource consumption of the Bypass-Switch . . . . . 271
A.2 Resource consumption of the synchronization sublayer . . . . . 272
A.3 Resource consumption of the shared memory subsystem . . . . . 272
A.4 Resource consumption of the demonstrator application . . . . . 272

# Appendix 1

# Resource Consumption of FPGA Implementation

The results have been achieved by using the synthesis tools Xilinx XST 6.3.3 and MAP 6.3.3 [158], which have been optimized for small area and targeted to the Virtex-II Pro [154] technology of the FPGA of the Stage 1 framework.

### Resource Consumption of the Bypass-Switch

The resource consumption of the switch is determined mainly by the number of global ports ($g$), the number of local ports for priority traffic ($p$) as well as the scheduling algorithm used. The number of best-effort ports has been set to 1 since there is only the DSM subsystem using this packet class.

| g | p | sched. | registers | | LUT4 elements | | slices | | DPBRAMs | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | iSLIP | 702 | 7 % | 1,169 | 11 % | 677 | 13 % | 3 | 6 % |
| 2 | 4 | DPA | 687 | 6 % | 1,167 | 11 % | 672 | 13 % | 3 | 6 % |
| 2 | 4 | RPA | 717 | 7 % | 1,224 | 12 % | 701 | 14 % | 3 | 6 % |
| 4 | 1 | iSLIP | 965 | 9 % | 1,739 | 17 % | 954 | 19 % | 5 | 11 % |
| 4 | 1 | DPA | 922 | 9 % | 1,711 | 17 % | 932 | 18 % | 5 | 11 % |
| 4 | 1 | RPA | 1,012 | 10 % | 1,910 | 19 % | 1,032 | 20 % | 5 | 11 % |
| 4 | 4 | iSLIP | 1,254 | 12 % | 2,488 | 25 % | 1,346 | 27 % | 5 | 11 % |
| 4 | 4 | DPA | 1,211 | 12 % | 2,461 | 24 % | 1,333 | 27 % | 5 | 11 % |
| 4 | 4 | RPA | 1,301 | 13 % | 2,659 | 26 % | 1,434 | 29 % | 5 | 11 % |
| 5 | 4 | iSLIP | 1,579 | 16 % | 3,747 | 38 % | 1,983 | 40 % | 6 | 13 % |
| 5 | 4 | DPA | 1,516 | 15 % | 3,708 | 37 % | 1,962 | 39 % | 6 | 13 % |
| 5 | 4 | RPA | 1,648 | 16 % | 4,007 | 40 % | 2,112 | 42 % | 6 | 13 % |

Table A.1: Resource consumption of the Bypass-Switch.

### Resource Consumption of the Synchronization Sublayer

The resource consumption of the synchronization sublayer is determined mainly by the number of global ports ($g$) that determine the number of interfaced MGTs, as

well as by the used CRC quality, since the checksum has to be calculated for each port within both the receive and the transmit data paths.

| g | CRC bits | registers | | LUT4 elements | | slices | |
|---|---|---|---|---|---|---|---|
| 4 | 4 | 287 | 2 % | 404 | 4 % | 381 | 7 % |
| 4 | 8 | 302 | 3 % | 477 | 4 % | 425 | 8 % |
| 6 | 4 | 381 | 3 % | 558 | 5 % | 524 | 10 % |
| 6 | 8 | 404 | 4 % | 672 | 6 % | 587 | 11 % |

Table A.2: Resource consumption of the synchronization sublayer.

### Resource Consumption of the Shared Memory Subsystem

The implemented DSM contains the DSM PAL, the routing logic, a single server that interfaces the SDRAM memory as well as a configurable number of clients.

| clients | registers | | LUT4 elements | | slices | | DPBRAMs | |
|---|---|---|---|---|---|---|---|---|
| 1 | 693 | 7 % | 863 | 8 % | 759 | 15 % | 4 | 9 % |
| 2 | 967 | 9 % | 1,352 | 13 % | 1,112 | 22 % | 5 | 11 % |

Table A.3: Resource consumption of the shared memory subsystem

### Resource Consumption of the Demonstrator Application

The demonstrator application for isochronous transfers is mainly using buffers (DPBRAM elements) to store data received or to be transmitted. The SDRAM access can be disabled to save logic.

| ports | SDRAM access | registers | | LUT4 elements | | slices | | DPBRAMs | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | - | 316 | 3 % | 444 | 4 % | 280 | 5 % | 4 | 9 % |
| 4 | - | 462 | 4 % | 767 | 7 % | 473 | 9 % | 8 | 18 % |
| 2 | + | 345 | 3 % | 459 | 4 % | 307 | 6 % | 4 | 9 % |
| 4 | + | 491 | 4 % | 782 | 7 % | 492 | 9 % | 8 | 18 % |

Table A.4: Resource consumption of the demonstrator application

# Bibliography


[1] Baddeley, R., L. Abbott, M. Booth, F. Sengpiel, T. Freeman, E. Wakeman and E. Rolls. "Responses of neurons in primary and inferior temporal visual cortices to natural scenes". In: *Proceedings of the Royal Society B: Biological Sciences*, volume 264(1389), page 1775–1783. dec 1997.

[2] Banovic, D. and I. Radusinov. "VOQ Simulator -Software Tool for Performance Analysis of VOQ Switches". In: *International Conference on Internet and Web Applications and Services, Advanced International Conference on Telecommunications*, page 71. February 2006.

[3] Becker, J. *Ein FPGA-basiertes Testsystem für gemischt analog/digitale ASICs*. Diploma thesis, Universität Heidelberg, Germany, 2001.

[4] Bhatti, S. N. and J. Crowcroft. "QoS-Sensitive Flows: Issues in IP Packet Handling". In: *IEEE Internet Computing*, 4(4):48–57, 2000.

[5] Bi, G.-Q. and M.-M. Poo. "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type". In: *Journal of Neuroscience*, 18(24):10464–10472, December 1998.

[6] Birkhoff, G. D. "Tres observaciones sobre el algebra lineal". In: *Universidad Nacional de Tucumán Revista*, A(5):147–151, 1946.

[7] Blake, S., D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss. "An Architecture for Differentiated Services". *RFC 2475*, December 1998. Http://www.ietf.org/.

[8] Borrelli, C. *Xilinx Application Note 209: IEEE 802.3 Cyclic Redundancy Check*. Xilinx, Inc., www.xilinx.com, 2001.

[9] Brad Dunsmore and Toby Skandier. *Telecommunications Technologies Reference*. Cisco Press, sep 2002.

[10] Braden, R. "Requirements for Internet Hosts - Application and Support". *RFC 1123*, October 1989. Http://www.ietf.org/.

[11] Braden, R. "Requirements for Internet Hosts - Communication Layers". *RFC 1122*, October 1989. Http://www.ietf.org/.

[12] Braden, R., L. Zhang, S. Berson, S. Herzog and S. Jamin. "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification". *RFC 2205*, September 1997. Http://www.ietf.org/.

[13] Brélaz, D. "New methods to color the vertices of a graph". In: *Commun ACM*, 22(4):251–256, 1979.

[14] Brüderle, D., A. Grübl, K. Meier, E. Mueller and J. Schemmel. "A Software Framework for Tuning the Dynamics of Neuromorphic Silicon Towards Biology". In: *Proceedings of the 9th International Work-Conference on Artificial Neural Networks (IWANN)*, volume 4507, pages 479–486. San Sebastián, Spain, September 2007.

[15] Brunel, N. "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons". In: *Journal of Computational Neuroscience*, 8(3):183–208, May 2000.

[16] Busson, P., L. Dobrzynski, A. Karar, T. Romanteau, J. de Papp and J. Macé. "A solution to reduce the latency of a multi gigabit transceiver (Virtex-II Pro). Effect of clock jitter on the bit error rate". In: *9th Workshop on Electronics for LHC Experiments*, 2003.

[17] C. Clos. "A study of non-blocking switching networks", 1953. Bell System Technical Journal' 32 (5): 406–424.

[18] Chang, C.-S., W.-J. Chen and H.-Y. Huang. "On service guarantees for input-buffered crossbar switches: acapacity decomposition approach by Birkhoff and von Neumann". In: *Seventh International Workshop on Quality of Service 1999*, pages 79–86. 1999.

[19] Chang, C.-S., W.-J. Chen and H.-Y. Huang. "Birkhoff-von Neumann Input Buffered Crossbar Switches". In: *Proceedings of the IEEE INFOCOM 2000*, pages 1614–1623. Tel Aviv, Israel, March 2000.

[20] Chang, C.-S., D.-S. Lee and Y.-S. Jou. "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering". In: *Computer Communications*, 25(6):611–622, April 2002.

[21] Chang, C.-S., D.-S. Lee and C.-M. Lien. "Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering". In: *Computer Communications*, 25(6):623–634, April 2002.

[22] Chang, C.-S., D.-S. Lee and Y.-J. Shih. "Mailbox switch: a scalable two-stage switch architecture for conflict resolution of ordered packets". In: *Proceedings of IEEE INFOCOM 2004*, volume 3, pages 1995–2006. April 2004.

[23] Chang, C.-S., D.-S. Lee and C.-Y. Yue. "Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches". In: *Proceedings of IEEE INFOCOM 2003*, volume 3, pages 1622–1632. April 2003.

[24] Chao, H. J. and J.-S. Park. "Centralized contention resolution schemes for a large-capacityoptical ATM switch". In: *Proceedings of IEEE ATM Workshop 1998*, pages 11–16. Fairfax, VA, May 1998.

[25] Chuang, S.-T., A. Goel, N. McKeown and B. Prabhakar. "Matching Output Queueing with a Combined Input Output Queued Switch". In: *Proceedings of the IEEE INFOCOM '99*, pages 1169–1178. New York, NY, March 1999.

[26] Curd, D. R. *Xilinx Application Note 660: Dynamic Reconfiguration of RocketIO MGT Attributes.* Xilinx, Inc., www.xilinx.com, 2004.

[27] Davie, B., A. Charny, J. C. R. Bennet, K. Benson, J. Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu and D. Stiliadis. "An Expedited Forwarding PHB (Per-Hop Behavior)". *RFC 3246*, March 2002. Http://www.ietf.org/.

[28] Dayan, P. and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* Cambridge, Massachusetts, MIT Press, 2001.

[29] Delgado-Frias, J. and G. B. Ratanpal. "A VLSI crossbar switch with wrapped wave front arbitration". In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50(1):135–141, January 2003.

[30] Demers, A., S. Keshav and S. Shenker. "Analysis and simulation of a fair queueing algorithm". In: *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, pages 1–12. Austin, Texas, 1989.

[31] Demichelis, C. "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)". *RFC 3393*, November 2002. Http://www.ietf.org/.

[32] Design Automation Standards Committee of the IEEE Computer Society, New York. *VHDL Language Reference Manual, IEEE Std. 1076.1*, 1997.

[33] Dipaolo, M. and L. Lewis. *Xilinx Application Note 763: Local Clocking for MGT RXRECCLK in Virtex-II Pro Devices.* Xilinx, Inc., www.xilinx.com, 2004.

[34] Douence, V., A. Laflaquiere, S. L. Masson, T. Bal and G. L. Masson. "Analog Electronic System for Simulating Biological Neurons". In: *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks (IWANN) 1999*, pages 188–197. 1999.

[35] Felicijan, T. *Quality-of-Service (QoS) for Asynchronous On-Chip Networks.* Ph.D. thesis, University of Manchester, Dept. of Computer Science, 2004.

[36] Ferrari, D. and D. C. Verma. "A Scheme for Real-Time Channel Establishment in Wide-Area Networks". In: *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[37] Fibre Channel Specification. "Fibre Channel Industry Association", 2008. www.fibrechannel.org.

[38] Fieres, J. *A Method for Image Classification Using Low-Precision Analog Computing Arrays.* Ph.D. thesis, Universität Heidelberg, 2005.

[39] Fieres, J., A. Grübl, S. Philipp, K. Meier, J. Schemmel and F. Schürmann. "A Platform for Parallel Operation of VLSI Neural Networks". In: L. Smith, A. Hussain and I. Aleksander, editors, *Proceedings of the Brain Inspired Cognitive Systems (BICS)*, 2004.

[40] Florissi, D. *Isochronets: a high-speed network switching architecture*. Ph.D. thesis, Columbia University, January 1995.

[41] Florissi, D. and Y. Yemini. "The Gigabit per Second Isochronet Switch". Technical report, Distributed Computing and Communications (DCC) Lab, Columbia University, June 1995.

[42] Ganjali, Y., A. Keshavarzian and D. Shah. "Cell switching versus packet switching in input-queued switches". In: *IEEE/ACM Transactions on Networking*, 13(4):782–789, August 2005.

[43] Gerstner, W. and W. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[44] Golestani, S. J. "A Stop-and-Go Queuing Framework for Congestion Management". In: *Proceedings of ACM SIGCOMM Computer Communications Review*, volume 20, pages 8–18. September 1990.

[45] Goossens, K., J. Dielissen and A. Radulescu. "Æthereal Network on Chip:Concepts, Architectures, and Implementations". In: *IEEE Design and Test of Computers*, 22(5):414–421, 2005. ISSN 0740-7475.

[46] Grübl, A. *Eine FPGA-basierte Plattform für neuronale Netze*. Diploma thesis, Universität Heidelberg, Germany, 2003.

[47] Grübl, A. *VLSI Implementation of a Spiking Neural Network*. Ph.D. thesis, Universität Heidelberg, 2007.

[48] Gupta, P. and N. McKeown. "Designing and Implementing a Fast Crossbar Scheduler". In: *IEEE Micro*, 19(1):20–28, 1999.

[49] Gutmann, C. *Implementation einer Gigabit-Ethernet-Schnittstelle zum Betrieb eines Künstlichen Neuronalen Netzwerkes*. Diploma thesis, Universität Heidelberg, Germany, 2007.

[50] Häflinger, P., M. Mahowald and L. Watts. "A Spike Based Learning Neuron in Analog VLSI". In: *Advances in neural information processing systems*, 9, 1996.

[51] Heinanen, J., F. Baker, W. Weiss and J. Wroclawski. "Assured Forwarding PHB Group". *RFC 2597*, June 1999. Http://www.ietf.org/.

[52] Hodgkin, A. L. and A. F. Huxley. "A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve". In: *Journal of Physiology*, 117(4):500–544, 1952.

[53] Hohmann, S. *Stepwise Evolutionary Training Strategies for Hardware Neural Networks*. Ph.D. thesis, Universität Heidelberg, 2005.

[54] Hohmann, S., J. Fieres, K. Meier, J. Schemmel, T. Schmitz and F. Schürmann. "Training Fast Mixed-Signal Neural Networks for Data Classification". In: *Proceedings of the 2004 International Joint Conference on Neural Networks IJCNN'04*, volume 4, pages 2647–2652. IEEE Press, Budapest, Hungray, July 2004.

[55] Hohmann, S. G., J. Schemmel, F. Schürmann and K. Meier. "Predicting Protein Cellular Localization Sites with a Hardware Analog Neural Network". In: *Proceedings of the Int. Joint Conf. on Neural Networks*, pages 381–386. IEEE Press, Portland, Oregon, jul 2003.

[56] Hopcroft, J. E. and R. M. Karp. "An $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs". In: *Society for Industrial and Applied Mathematics, Journal of Computing*, 2(4):225–231, December 1973.

[57] HTX Board. "a universal HTX test platform", 2008. http://www.ra.informatik.uni-mannheim.de/.

[58] Hung, A., G. Kesidis and N. McKeown. "ATM input-buffered switches with the guaranteed-rate property". In: *Proceedings of the IEEE: Third IEEE Symposium on Computers and Communications, ISCC '98*, pages 331–335. July 1998.

[59] Hurt, J., A. May, X. Zhu and B. Lin. "Design and implementation of high-speed symmetric crossbar schedulers". In: *IEEE International Conference on Communications*, volume 3, pages 1478–1483. June 1999.

[60] HyperTransport Technology Consortium. *HyperTransport I/O Link Specification*, revision 3.0a edition, November 2006. Document No. HTC20051222-0046-0017.

[61] IBM corp., www.ibm.com. *64-Bit Processor Local Bus, Architecture Specifications Version 3.5*, may 2001. SA-14-2534-01.

[62] Infiniband Specification. "Infiniband Trade Association", 2008. http://www.infinibandta.org.

[63] Integrated Device Technology, Inc., www.idt.com. *IDT71V2556 Datasheet: 128K x 36, 256K x 18, 3.3V, Synchronous ZBT SRAMs, 2.5V I/O, Burst Counter, Pipelined Outputs*, 2004.

[64] Iyer, S. and N. McKeown. "Analysis of the Parallel Packet Switch Architecture". In: *IEEE/ACM Transactions on Networking*, 11(2):314–324, April 2003.

[65] Jayarajan, N. *Xilinx Application Note 562: Configurable LocalLink CRC Reference Design*. Xilinx, Inc., www.xilinx.com, 2007.

[66] Jiang, Y. and M. Hamdi. "A fully desynchronized round-robin matching scheduler for a VOQpacket switch architecture". In: *IEEE Workshop on High Performance Switching and Routing*, pages 407–411. Dallas, TX, May 2001.

[67] Olivier Jolly, PhD Thesis, in preparation, Heidelberg, 2008.

[68] Kalmanek, C., H. Kanakia and S. Keshav. "Rate controlled servers for very high-speed networks". In: *Proceedings of the IEEE GLOBECOM '90*, volume 1, pages 12–20. San Diego, CA, USA, December 1990.

[69] Karol, M. J., M. G. Hluchyj and S. P. Morgan. "Input Versus Output Queueing on a Space-Division Packet Switch". In: *IEEE Transactions on Communications*, 35:1347–1356, December 1987.

[70] Karp, R. M. "Reducibility among combinatorial problems". In: R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[71] Kermani, P. and L. Kleinrock. "Dynamic Flow Control in Store-and-Forward Computer Networks". In: *IEEE Transactions on Communications*, 28:263–271, February 1980.

[72] Keslassy, I., C.-S. Chang, N. McKeown and D.-S. Lee. "Optimal load-balancing". In: *Proceedings of the IEEE INFOCOM 2005*, volume 3, pages 1712–1722. Miami, FL, March 2005.

[73] Keslassy, I., M. Kodialam, T. V. Lakshman and D. Stiliadis. "On guaranteed smooth scheduling for input-queued switches". In: *IEEE/ACM Transactions on Networking*, 13:1364–1375, December 2005.

[74] Keslassy, I. and N. McKeown. "Maintaining Packet Order in Two-Stage Switches". In: *Proceedings of the IEEE INFOCOM 2002*, volume 2, pages 1032–1041. New York, NY, June 2002.

[75] Keslassy, I., R. Zhang-Shen and N. McKeown. "Maximum size matching is unstable for any packet switch". In: *IEEE Communications Letters*, 7(10):496–498, October 2003.

[76] Keyvani, M. *VHDL Implementation of a High-Speed Symmetric Crossbar Switch*. Ph.D. thesis, University of Tehran, August 2001.

[77] Kowalczyk, J. *Xilinx Application Note 670: Minimizing Receiver Elastic Buffer Delay in the Virtex-II Pro RocketIO Transceiver*. Xilinx, Inc., www.xilinx.com, 2003.

[78] Li, J. and N. Ansari. "QoS guaranteed input queued scheduling algorithms with low delay". In: *Proceeding of the IEEE Workshop on High Performance Switching and Routing*, pages 412–414. May 2001.

[79] Linux. "Linux Online", 2008. www.linux.org.

[80] Markram, H. "The Blue Brain Project". In: *Nature Reviews Neuroscience*, 7(2):153–160, February 2006.

[81] Marsan, M. A., A. Bianco, P. Giaccone, E. Leonardi and E. Neri. "Packet Scheduling in Input-Queued Cell-Based Switches". In: *Proceedings of the IEEE INFOCOM 2001*, volume 2, pages 1085–1094. Anchorage, AK, USA, April 2001.

[82] Maxim Integrated Products, www.maxim.com. *MAX5253 Datasheet: +3V, Quad, 12-Bit Voltage-Output DAC with Serial Interface*, 2002.

[83] McCulloch, W. S. and W. H. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[84] McKeown, N. "The iSLIP scheduling algorithm for input-queued switches". In: *IEEE/ACM Transactions on Networking*, 7(2):188–201, 1999.

[85] McKeown, N. and T. E. Anderson. "A quantitative comparison of iterative scheduling algorithms for input-queued switches". In: *Computer Networks and ISDN Systems*, 30(24):2309–2326, 1998.

[86] McKeown, N., M. Izzard, A. Mekkittikul, W. Ellersick and M. Horowitz. "Tiny Tera: A Packet Switch Core — Using new scheduling algorithms to build a 1-terabits packet switch with a central hub no larger than a can of soda". In: *IEEE Micro*, 17(1):26–33, February 1997.

[87] McKeown, N., A. Mekkittikul, V. Anantharam and J. Walrand. "Achieving 100 % throughput in an input-queued switch". In: *IEEE Transactions on Communications*, 47:1260–1267, August 1999.

[88] Mehrotra, A. and M. A. Trick. "A column generation approach for graph coloring". In: *INFORMS Journal on Computing*, 8:344–354, 1996.

[89] Meier et al., K. "The FACETS Project: Fast Analog Computing with Emerging Transient States", 2005. EU FP6-2004-IST-FETPI, contract no. 15879.

[90] Mekkittikul, A. and N. McKeown. "A Starvation-free Algorithm for Achieving 100 % Throughput in an Input-Queued Switch". In: *Proceedings of the International Conference on Computer Communications and Networks (ICCCN'96)*, pages 226–231. Rockville, October 1996.

[91] Mekkittikul, A. and N. McKeown. "A practical scheduling algorithm to achieve 100 % throughput in input-queued switches". In: *Proceedings of the IEEE INFOCOM '98*, volume 2, pages 792–799. April 1998.

[92] Metcalfe, R. M. and D. R. Boggs. "Ethernet: distributed packet switching for local computer networks". In: *Commun ACM*, 19(7):395–404, jul 1976.

[93] Mills, D. L. "Network Time Protocol (NTP)". *RFC 958*, September 1985. Http://www.ietf.org/.

[94] ModelSim Simulator. "a comprehensive simulation and debug environment for complex ASIC and FPGA designs", 2008. http://www.model.com/.

[95] Monatvista Linux. "Embedded Linux Software and development tools for intelligent devices and embedded systems", 2008. http://www.mvista.com/.

[96] Moy, J. "OSPF Version 2". *RFC 2328*, April 1998. Http://www.ietf.org/.

[97] Mueller, E. *Simulation of High-Conductance States in Cortical Neural Networks*. Master's thesis, Universität Heidelberg, Germany, 2003.

[98] Nagle, J. "On packet switches with infinite storage". *RFC 970*, December 1985. Http://www.ietf.org/.

[99] von Neumann, J. "A certain zero-sum two-person game equivalent to the optimal assignment problem". In: H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games, Vol. II*, pages 5–12. Prinston University Press, 1953.

[100] Ni, L. M. and P. McKinley. "A survey of wormhole routing techniques in direct networks". In: *IEEE Computer*, 26(2):62–76, February 1993.

[101] Nichols, K., S. Blake, F. Baker and D. Black. "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers". *RFC 2474*, December 1998. Http://www.ietf.org/.

[102] NS2. "The Network Simulator", 2008. http://www.isi.edu/nsnam/ns/.

[103] Olesinski, W., H. Eberle and N. Gura. "PWWFA: The Parallel Wrapped Wave Front Arbiter for Large Switches". In: *Workshop on High Performance Switching and Routing Conference (HPSR'07)*, pages 1–6. Brooklyn, New York, May 2007.

[104] Parekh, A. K. and R. G. Gallager. "A generalized processor sharing approach to flow control in integrated services networks: the single-node case". In: *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[105] Parekh, A. K. and R. G. Gallager. "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case". In: *IEEE/ACM Transactions on Networking*, 2(2):137–150, April 1994.

[106] Park, J. S. *The Folded Hypercube ATM Switches*. Ph.D. thesis, Virginia Polytechnic Institute and State University, September 2001.

[107] Patterson, D. and J. L. Hennessy. *Computer Organization and Design*. Morgan Kaufmann, third edition, August 2007. ISBN 0123706068.

[108] PCI Express. "PCI Express Specification", 2008. www.pcisig.com.

[109] Peter Alfke. "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", 1996. Xilinx application note, www.xilinx.com.

[110] Peterson, L. L. and B. S. Davie. *Computer Networks, A Systems Approach*. Morgan Kaufmann, 3th edition, May 2003.

[111] Philipp, S., A. Grübl, K. Meier and J. Schemmel. "Interconnecting VLSI Spiking Neural Networks Using Isochronous Connections". In: *Proceedings of the 9th International Work-Conference on Artificial Neural Networks (IWANN'2007)*, volume 4507, pages 471–478. San Sebastián, Spain, September 2007.

[112] Plummer, D. C. "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware". *RFC 826*, November 1982. Http://www.ietf.org/.

[113] Postel, J. "User Datagram Protocol". *RFC 768*, August 1980. Http://www.ietf.org/.

[114] Postel, J. "Internet Protocol, DARPA Internet Program Protocol Specification". *RFC 791*, September 1981. Http://www.ietf.org/.

[115] PyNN. "a Python package for simulator-independent specification of neuronal network models", 2008. www.neuralensemble.org.

[116] Python. "The Python Programming Language - Official Website", 2008. www.python.org.

[117] Rijpkema, E., K. Goossens, A. adulescu, J. van Meerbergen, P. Wielage and E. Waterlander. "Trade offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip". In: *In Proceedings of the conference on Design, Automation and Test in Europe*, March 2003.

[118] Rosen, E., A. Viswanathan and R. Callon. "Multiprotocol Label Switching Architecture". *RFC 3031*, January 2001. Http://www.ietf.org/.

[119] Rosenblatt, F. "The perception: a probabilistic model for information storage and organization in the brain". In: *Psychological Review*, 65(6):386–408, May 1958.

[120] SATA. "Serial ATA International Organization ( SATA-IO )", 2008. www.sata-io.org.

[121] Schemmel, J., J. Fieres and K. Meier. "Wafer-Scale Integration of Analog Neural Networks". In: *submitted for publication to the 2008 International Joint Conference on Neural Networks IJCNN'08*, April 2008.

[122] Schemmel, J., A. Grübl, K. Meier and E. Mueller. "Implementing Synaptic Plasticity in a VLSI Spiking Neural Network Model". In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, IEEE Press (2006).

[123] Schemmel, J., S. Hohmann, K. Meier and F. Schürmann. "A Mixed-Mode Analog Neural Network Using Current-Steering Synapses". In: *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, 2004.

[124] Schemmel, J., K. Meier and E. Mueller. "A new VLSI model of neural microcircuits including spike time dependent plasticity". In: *IEEE International Joint Conference on Neural Networks*, volume 3, pages 1711–1716. July 2004.

[125] Schmitz, T. *Evolution in Hardware - Eine Experimentalplattform zum parallelen Training analoger neuronaler Netzwerke*. Ph.D. thesis, Universität Heidelberg, 2005.

[126] Schoenen, R., G. Post and G. Sander. "Weighted Arbitration Algorithms with Priorities for Input-Queued Switches with 100 % Throughput". In: *Proceedings of the IEEE International Workshop on Broadband Switching Systems*, 1999.

[127] Schürmann, F. *Exploring Liquid Computing in a Hardware Adaptation: Construction and Operation of a Neural Network Experiment*. Ph.D. thesis, Universität Heidelberg, 2005.

[128] "SenseMaker: A Multi-sensory, Task-specific, Adaptable perception System". IST-2001-34712.

[129] Serpanos, D. N. and P. Antoniadis. "FIRM: A Class of Distributed Scheduling Algorithms for High-Speed ATM Switches with Multiple Input Queues". In: *Proceedings of the IEEE INFOCOM 2000*, volume 2, pages 548–555. 2000.

[130] Shizhao, L. and N. Ansari. "Input-queued switching with QoS guarantees". In: *Proceedings of the IEEE INFOCOM '99*, volume 3, pages 1152–1159. March 1999.

[131] Shreedhar, M. and G. Varghese. "Efficient fair queueing using deficit round robin". In: *ACM SIGCOMM Comput Commun Rev*, 25(4):231–242, October 1995.

[132] SIM. "A Fixed Length Packet Simulator", 2008. http://klamath.stanford.edu/tools/SIM/.

[133] Sivaraman, V., F. M. Chiussi and M. Gerla. "End-to-End Statistical Delay Service under GPS and EDF Scheduling: A Comparison Study". In: *Proceedings of the IEEE INFOCOM 2001*, volume 2, pages 1113–1122. Anchorage, AK, USA, April 2001.

[134] Sollins, K. R. "The TFTP protocol (revision 2)". *RFC 783*, June 1981. Http://www.ietf.org/.

[135] Sollins, K. R. "The TFTP protocol (revision 2)". *RFC 1350*, July 1992. Http://www.ietf.org/.

[136] Song, S., K. D. Miller and L. F. Abbott. "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity". In: *Nature Neuroscience*, 3(9):919–926, 2000.

[137] Spratt, M. "The architecture of IEEE 802.6 MANs". In: *IEE Colloquium on Fast Packet Switching*, 28(4):4/1–1/5, January 1991.

[138] Stoica, I. and H. Zhang. "Exact emulation of an output queueing switch by a combined inputoutput queueing switch". In: *Sixth International Workshop on Quality of Service (IWQoS '98)*, pages 218–224. Napa, CA, May 1998.

[139] Stroustrup, B. *The C++ Programming Language*, 1997.

[140] SVN. "Subversion - Version Control System", 2008. subversion.tigris.org.

[141] Tamir, Y. and H. C. Chi. "Symmetric Crossbar Arbiters for VLSI Communication Switches". In: *IEEE Transactions on Parallel and Distributed Systems*, 04(1):13–27, 1993.

[142] Tamir, Y. and G. L. Frazier. "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches". In: *IEEE Transactions on Computers*, 41(6):725–737, June 1992.

[143] Tannenbaum, A. S. *Computer Networks*. Pearson Education, 4th edition, 2004.

[144] Telecommunications Industry Association. *TIA TIA/EIA-644-A: Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits*, February 2001.

[145] "Token ring access method and Physical Layer specifications, ANSI/IEEE Std 802.5-1998E(R2003), standards.ieee.org, 2008".

[146] Tomazic, A. *Graphenfärbung mit Hilfe linearer Programmierung*. Diploma thesis, Universität Augsburg, Germany, 2005.

[147] Trynosky, S. *Xilinx Application Note 648: Serial Backplane Interface to a Shared Memory*. Xilinx, Inc., www.xilinx.com, 2004.

[148] Turner, J. and N. Yamanaka. "Architectural Choices in Large Scale ATM Switches". In: *IEICE Transactions on Communications*, 81(2):120–137, 1998.

[149] Wendt, K. *Mapping- und Konfigurationstool fü das FACETS-Design-Framework*. Diploma thesis, TU Dresden, Germany, May 2007.

[150] Widmer, A. X. and P. A. Franaszek. "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code". In: *IBM Journal of Research and Development*, 27(5):440–451, 1983.

[151] Wroclawski, J. "The Use of RSVP with IETF Integrated Services". *RFC 2210*, September 1997. Http://www.ietf.org/.

[152] Xilinx, www.xilinx.com. *PPC405 Processor Block Reference Guide V2.2*, June 2007.

[153] "Virtex-5 Multi-Platform FPGA". *www.xilinx.com*, 2007.

[154] Xilinx, Inc., www.xilinx.com. *Virtex-II Pro Platform FPGA Handbook*, 2002.

[155] Xilinx, Inc., www.xilinx.com. *Xilinx Answer Record 13962: What are the latencies from TXDATA to TXN/TXP and from RXN/RXP to RXDATA?*, 2002.

[156] Xilinx, Inc., www.xilinx.com. *Xilinx Answer Record 14669: Why are TX_BUFFER_USE and RX_BUFFER_USE always set to TRUE? Can I change them?*, 2002.

[157] Xilinx, Inc., www.xilinx.com. *RocketIO(TM) Transceiver User Guide*, 2004.

[158] Xilinx, Inc., www.xilinx.com. *Xilinx ISE 6 Software Manuals and Help - PDF Collection*, 2004.

[159] Yemini, Y. and D. Florissi. "Isochronets: a high-speed network switching architecture". In: *Proceedings of the IEEE INFOCOM '93*, volume 2, pages 740–747. March 1993.

[160] Yu, C.-L., C.-S. Chang and D.-S. Lee. "CR Switch: A Load-Balanced Switch with Contention and Reservation". In: *Proceedings of the IEEE INFOCOM 2007*, pages 1361–1369. Anchorage, AK, May 2007.

[161] Zhang, H. "Service disciplines for guaranteed performance service inpacket-switching networks". In: *Proceedings of IEEE*, 83(10):1374–1396, October 1995.

[162] Zheng, S. Q. and M. Yang. "Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters". In: *IEEE Transactions on Parallel and Distributed Systems*, 18(1):84–95, January 2007.

[163] Zimmermann, H. "OSI Reference model - The ISO model of architecture for open systems intercommunications". In: *IEEE Transactions on Communications*, 28(4):425–432, April 1980.

# Danksagung (Acknowledgements)

Mein herzlicher Dank gilt allen, die zum Gelingen dieser Arbeit beigetragen haben. Insbesondere möchte ich mich an dieser Stelle bedanken bei:

- Herrn Prof. Dr. Karlheinz Meier für die freundliche Aufnahme in die Electronic Vision(s) Arbeitsgruppe und die Möglichkeit, in einem so interessanten Projekt mitarbeiten zu können.
- Herrn Prof. Dr. Ulrich Rückert, der sich freundlicherweise bereit erklärt hat das Zweitgutachten zu übernehmen.
- Dr. Johannes Schemmel, der immer ein kompetenter und scharfsinniger Gesprächspartner war, für die vielen interessanten Diskussionen über das Projekt und für seine bestätigende wie auch kritische Art, die mir in vielen fachlichen Diskussionen weitergeholfen hat.
- Meinen Kollegen vom "Hardwarezimmer", insbesondere: Andi, Tillman und Dan. Für die schöne gemeinsame Zeit, für die fachlichen Diskussionen über Hardware-Design und dafür, dass eigentlich immer gute Stimmung war.
- Meinen Diplomanden Christian Gutmann und Alexander Sinsel, deren Arbeiten das Projekt mit voran gebracht haben. Christian ein ganz besonderer Dank für die Hilfe bei den Messungen in der Schlussphase der Arbeit.
- Allen aktuellen und ehemaligen Freunden und Kollegen aus der Electronic Vision(s) Gruppe. Es sind zu viele sie hier alle aufzuzählen. Danke für die angenehme Atmosphäre.
- Nochmals einen ganz herzlichen Dank an Dr. Andreas Grübl und Dr. Martin Philipp für das unermüdliche Korrekturlesen des Manuskripts und die vielen nützlichen Hinweise, Lob und Kritik.
- Meinem langjährigen Freund Paul Starzetz, der mein Interesse an Netzwerken mit geweckt hat und der immer für interessante Diskussionen in diesem Bereich gut ist.

- Meinen lieben Eltern für ihre moralische und nicht zuletzt auch für ihre finanzielle Unterstützung, ohne die mein Studium wohl so nicht möglich gewesen wäre.
- Meiner lieben Mahsa für die vielen schönen Momente, die wir zusammen verbringen können.