Network-on-Chip in Digital Hearing Aids

M.Sc. thesis

Technical University of Denmark (DTU)
Informatics and Mathematical Modelling (IMM)
Computer Science and Engineering

31st of July 2006

IMM M.Sc. thesis nr.: 76
Supervisor: *Prof. Jens Sparsø*
External supervisor: *Thomas Troelsen (Widex A/S)*

Morten Sleth Rasmussen (s011295)

Abstract

This project investigates the costs of replacing the communication structure of a hearing aid system developed by *Widex A/S*. The existing design is based on an inflexible ad-hoc point-to-point structure, which is sought replaced by a flexible communication structure in future system designs.

Two NoC structures have been implemented and synthesized for area and power estimation. As power consumption and area cost are constrained design parameters in hearing aids, all design effort has been put into designing the cheapest solution that fulfill the system requirements.

The power dissipation and area estimates of the NoCs are compared to the current system design to get a realistic estimate of the costs involved in using a NoC. The comparison shows a low area and power overhead, which indicate that NoC are suitable for even small lower power DSP-systems.

This report documents the design, implementation and synthesis of the networks. Design choices involved in designing NoCs such as topology, application mapping, services, routing and interfacing are discussed. As the NoC solution is designed with a particular system in mind, an introduction to digital hearing aid systems is also given along with a short introduction to the NoC concept.

Acknowledgements

This Master of Science project has been carried out at the Technical University of Denmark in cooperation with *Widex A/S*. I would like to thank *Widex A/S* for hospitality, for letting me work at your facilities, for giving me inside information on your digital hearing aid system design, for giving access to your design flow and expertise. I would like thank Thomas Troelsen from *Widex A/S* for insightful discussions on future hearing aid systems designs. I am grateful to Lars Fomsgaard from *Widex A/S* for technical assistance on using the design flow.

I would in particular like to thank my supervisor Jens Sparsø. Your interest on network-on-chip has been a great inspiration to me.

Contents

Acknowledgements **iii**

Contents **iv**

1 Introduction **1**
1.1 Project description 2
1.2 Network-on-Chip 2
1.3 Previous work 3
1.4 Report structure 4

2 Network on Chip **5**
2.1 Introduction 5
2.2 Topologies 6
2.2.1 Regular topologies 8
2.2.2 Irregular topologies 9
2.3 Routing 10
2.4 Services 12
2.5 Programming model 13

3 Digital Hearing Aids **15**
3.1 Introduction to Hearing Aids 16
3.2 The back-end chip 16
3.2.1 System architecture 17
3.3 Internal back-end communication 18
3.3.1 Traffic types 18
3.3.2 Communication protocol 19
3.3.3 Traffic patterns 20
3.3.4 Programming 21
3.4 Low power design 23
3.5 Future hearing aid systems 23

4 Design **25**
4.1 Requirements 25
4.2 Network Topology 26

4.2.1 Tree topologies . . . 27
4.2.2 Grid topologies . . . 27
4.2.3 Custom topologies . . . 28
4.2.4 Design choices . . . 30
4.3 Application mapping . . . 31
4.3.1 Design choices . . . 32
4.4 Services and features . . . 34
4.4.1 Guaranteed service . . . 35
4.4.2 Best effort service . . . 35
4.4.3 Use-case switching . . . 36
4.4.4 Multicast . . . 36
4.4.5 Design choice . . . 36
4.5 Routing . . . 37
4.5.1 Switching . . . 38
4.5.2 Forwarding . . . 40
4.5.3 Buffering . . . 41
4.5.4 Protocol . . . 44
4.6 Programming model . . . 48
4.6.1 Distributed programming . . . 49
4.6.2 Centralized programming . . . 49
4.6.3 Programming method . . . 49
4.6.4 Design choices . . . 49
4.7 Interfacing . . . 50
4.7.1 Abstraction . . . 50
4.7.2 Interface . . . 50
4.7.3 Design choices . . . 51
4.8 The networks . . . 51
4.8.1 The routing node . . . 52
4.8.2 The network adapter . . . 53
4.8.3 The mesh . . . 54
4.8.4 The optimized mesh . . . 55

5 Implementation 57
5.1 Routing node . . . 57
5.1.1 NoC links and flits . . . 58
5.1.2 Flit handler . . . 58
5.1.3 Router . . . 59
5.2 Network adapter . . . 60
5.2.1 Unpack unit . . . 60
5.2.2 Pack unit . . . 60
5.3 Network configuration . . . 61
5.4 Traffic generators . . . 61
5.5 Synthesis . . . 61
5.5.1 Design flow . . . 61

5.5.2 Power estimation 62
5.6 Verification 64

6 Results **67**
6.1 NoC performance 67
6.1.1 NoC utilization 67
6.1.2 NoC latency 68
6.2 Network costs 70
6.2.1 Area 71
6.2.2 Power consumption 71

7 Discussion **73**
7.1 Result Elaboration 73
7.2 NoC in DSP-systems 75
7.3 Future work 76

8 Conclusion **77**

Bibliography **79**

A NoC costs **A.1**
A.1 Mesh NoC internal costs A.1
A.2 Optimized NoC internal costs A.1

B CD contents **B.1**

C Source code **C.1**
C.1 NoC components C.1
C.1.1 Flit handler C.1
C.1.2 BE router C.3
C.1.3 Data buffer C.7
C.1.4 VC buffer C.8
C.1.5 Address table C.9
C.1.6 Pack Unit C.10
C.1.7 Unpack Unit C.13
C.1.8 Programming Unit C.16
C.1.9 Network Adapter C.17
C.1.10 Routing Node C.21
C.1.11 Traffic generator C.35
C.2 Optimized NoC C.37
C.2.1 Optimized NoC C.37
C.2.2 Global settings C.49
C.2.3 Main testbench C.50
C.3 Mesh NoC C.63
C.3.1 Mesh NoC C.63

C.3.2 Global settings C.81
C.3.3 Main testbench C.81

Chapter 1

Introduction

The rapid development in the area of CMOS technology, allow more and more complex circuits to be integrated into a single chip. The idea of having a complete system in a single chip is today possible and is known as System-on-Chip (SoC).

In order to keep up with increasing system complexity, designers tend to use larger reusable blocks when designing systems. Designing everything from scratch is no longer feasible and larger blocks like microprocessors, DSP-units, memories and I/O-controllers are reused. These can be thought of as separate subsystems, designed independently and then put together. The major challenge of designing systems is then how to put these subsystems together. A general connection structure must be designed to support all the different subsystems and allow them to exchange data. Furthermore a standard interface must be employed to ensure cooperation between blocks and communication structure.

Point-to-point connections is an obvious solution, but will give a fully connected network structure, when all subsystems have to be able to communicate. This solution is requires a lot of wires and becomes infeasible for systems containing more than a few subsystems. Furthermore the wire utilization is low and thereby not very area efficient.

A shared bus gives better wire utilization, but becomes a bottleneck in the system. It does not allow independent data transfers to take place concurrently. Busses does not scale well because the capacitance increases dramatically, with increasing bus length and more subsystems connected.

A more suitable solution may be found somewhere in between, by having several shared busses. Taking a step further and introducing more intelligent bus sharing, will lead to a network-like structure, where links are shared and data is routed through these, from one point to another. Using this kind of communication structure also allows more flexible communication. Adding reconfigurability to the network will increase the system flexibility even more by supporting multiple configurations, which can be switched at run-time to meet communication requirements in different situations. This feature is useful in DSP systems, where data may be processed by different subsystems under different configurations.

1.1 Project description

In this project a flexible solution for communication in an existing special purpose DSP is designed. The goal is to replace the existing communication structure in the system, with a configurable structure and investigate the costs in terms of area and power.

The DSP is a configurable system for digital audio processing in digital hearing aids developed by *Widex A/S*. It consists of a number of audio processing blocks connected by point-to-point connections, which can be thought of as the subsystems of the hearing aid. In addition a shared bus is used for configuration purposes. Blocks are reconfigured regularly to match changing audio conditions, i.e. filter parameters are tuned in. Special modes can be selected by the user, for special situations, which reconfigures the audio blocks by loading in new instructions and parameters.

The current implementation is inflexible and leaves no room for changing algorithms, when the system has been designed. More and more advanced features require more communication between the subsystems, and make a more flexible communication structure more suitable for the system.

The current system consist of special purpose blocks, these become more and more advanced and new blocks are introduced into the system, as new audio processing algorithms are developed. Future systems may include general purpose DSP-blocks that can be used for various tasks under different configurations. It is of great interest to investigate new and more flexible communication structures for future generations of digital hearing aid systems.

1.2 Network-on-Chip

Network-on-Chip (NoC) is gaining more and more attention at universities and conferences[1], as it is believed to be a solution to existing and future design problems.

NoC is a different way of designing systems that has great influence on SoC systems during design, implementation and on the functionality of the final system. The idea behind the NoC concept is to replace point-to-point wires and global shared busses, with a generic communication medium that can support all types of communication as illustrated in Figure 1.1. NoC addresses issues like:

- Electrical wire scaling in very large systems by wire sharing in a regular wiring structure avoiding long wires.
- System synchronization. Decoupling communication from computation loosens the global synchronization requirements and global clock distribution may be avoided.
- Design productivity. NoCs provide an excellent platform for modular design and reuse.

[1] At Design, Automation and Test in Europe Conference 2006 a new symposium was proposed under the name "IEEE International Symposium on Networks on Chip (NoC)"

A NoC consists of routing nodes connected in some topology using point-to-point links. The network can be expanded by adding more routing nodes, which scales well as no additional electrical capacitance or other parasitic factors has greater effect in larger networks. Links and nodes may be pipelined to increase the bandwidth. One may see the network as an advanced pipeline, where data is send through a link every clock cycle. It is determined at each routing node where to send data next, which requires logic control of the dataflow that implements the routing protocol. It may be necessary to stall some links, while others continue to solve conflicts. Just like stalling an ordinary pipeline to avoid data hazards.

By sharing the links between routing nodes, the number of wires is reduced and there are no very long wires with large capacitances. Long wires are segmented by the network pipeline, which reduce or eliminate the need for buffers and reduce the cost overhead a network interconnect.

The subsystems (blocks) are connected to the network and are then able to communicate with other subsystems connected to the network. Data is routed through the network from source to destination, which may not be directly connected. The blocks send and receive data through a standard interface and the network handles data transport. This interface marks the clear separation and decoupling of communication and computation in NoC systems.

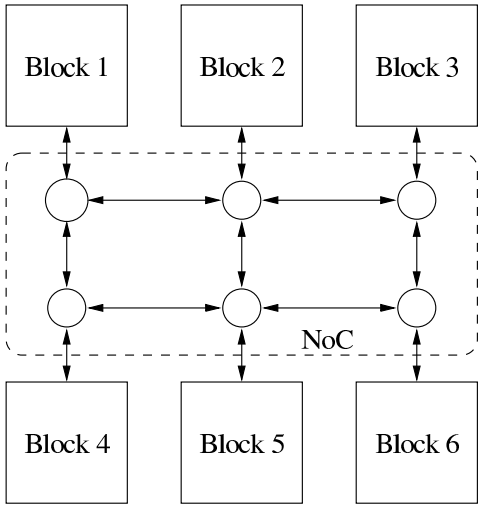


Figure 1.1: NoC-based SoC.

1.3 Previous work

Currently Network-on-Chip is a research topic on several universities. Different approaches are investigated which focus on the advantages of using NoC. While some focus on creating general network platforms, suitable for a wide range of systems, others focus on more specialized networks for a narrow range of applications. Both

clock-less and clocked networks exist. Asynchronous networks try to avoid the clock distribution problem in very large systems.

These networks are in general large and expensive, but offer many features. Most of them use standard block interfaces like OCP, creating a general network that can be used for a wide range of applications.

MANGO[1] is an asynchronous network supporting guaranteed services and best effort traffic. CHAIN[2] is a simpler asynchronous network, which has been demonstrated in a multiplexer/demultiplexer tree-like topology for a small system. SPIN[3] investigates a network based on the butterfly fat-tree topology. QNoC[4] is a customized Quality of Service NoC architecture derived by modifying generic topologies. Xpipes[5] is library of network components that can be configured in various network architectures, which can be used for both regular general networks and application specific networks.

The networks mentioned above are all developed for academic purposes. To my knowledge AEthereal[6] is the only network developed and used for industrial purposes. It is synchronous and supports both guaranteed services and best effort communication. The cost of using AEthereal in a commercial SoC is investigated in [7].

1.4 Report structure

This report will give a short introduction to the most essential topics in the area of Network-on-Chip in Chapter 2, where references to sources of more information on NoC can be found too. Chapter 3 will introduce digital hearing aid systems. The focus will mainly be on digital hearing aids from a system point of view, where communication is important. Chapter 4 will discus how to design a more flexible communication structure for digital hearing aid systems based on a black box view of the existing system. Chapter 5 will give some details on how the NoC is implemented and the design flow used for the implementation. Chapter 6 will present a communication analysis, based on a model of the actual communication in the existing system. The cost of introducing a new communication structure is presented here as well. Chapter 7 will discus the costs of more flexible interconnects. Trade offs between cost and added features in the system will be discussed along with future hearing aid system designs based on NoC. Chapter 8 concludes the thesis.

Chapter 2

Network on Chip

This chapter will give an introduction NoCs and terms used to in the context of intra-chip communication. A more thorough introduction can be found in the referenced papers and articles, especially the survey found in [1]. This chapter is safe to skip for readers already familiar to the essential NoC principles.

2.1 Introduction

The increasing complexity and number of blocks in digital systems causes more communication within the chip. The limiting resource in future technologies is expected to shift from transistors to wires. NoC-based systems try to shift the design focus to give communication more attention and deal with it in a structured way.

In existing systems the blocks are tied together using point-to-point links and buses to form a working SoC. In [8] the idea of using networks instead of ad-hoc solutions using point-to-point connections and buses is introduced. With this approach subsystems communicate by sending packets to one another over a network connecting all subsystems. A network is a more general communication medium, which both has a more well-defined structure and adds more flexibility to the system. Wire sharing is an inherent feature of communication networks and scales better than ad-hoc bus designs.

Network-on-Chip is broad term which simply indicates that some kind of communication network is implemented on the chip. Many design choices and trade offs must be made when designing the network, to try to find an optimal solution to the communication demands of the particular system. No single network design is optimal in all designs and for all applications. Systems have different design requirements, like power consumption, physical size, performance and work load, which will cause some designs more suitable than others.

Replacing the wire structures on a chip with a network comes at a cost. Where communication structure was made up of wires and buffers before, a network also has to include control logic, to switch and route data through the network. Area overhead

and increased communication latency must be expected, when using a network. But it adds more features to the system in terms of flexibility.

Furthermore, modular design is an inherent feature of the structured design approach of NoCs. Modular design and more reuse is becoming more and more important to the design productivity to keep up with the increasing amounts of available chip resources. Blocks designed using a standard interface can easily be used in different NoC systems. One could imagine SoC designs made up of a NoC where subsystems are just plugged in.

2.2 Topologies

A network is a common communication structure with communicating blocks attached. In general a network can be classified as a *direct network* or an *indirect network* [1].

In a *direct network* blocks communicate directly with each other and has the ability to route data. A block does not require a direct connection to communicate. Data is routed through intermediate blocks to reach the destination. *Indirect networks* have separated the block and network routing into distinct components. The network is made up of routing components connected with each other using uni- or bidirectional links. Blocks are attached to these routing components, but the network may have routing components which does nothing else but routes data. Both types of networks are illustrated in Figure 2.1.

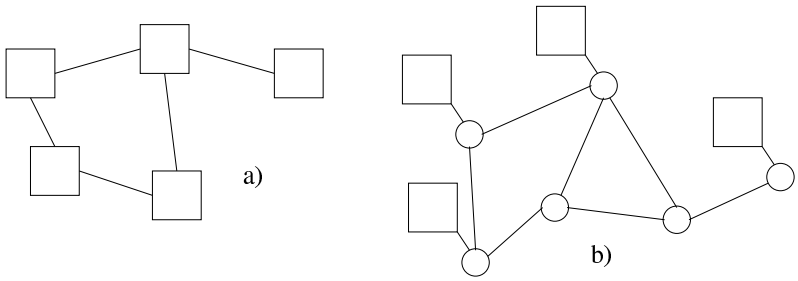


Figure 2.1: a) Direct network. b) Indirect network.

From a simplified perspective a network consists of the following fundamental components:

- *Routing Nodes* route the data through the network according to the chosen protocol. The routing nodes handle switching of data between the links and implement the routing strategy.
- *Links* connects the routing nodes to allow the routing nodes to communicate and exchange data. Links can be uni- or bidirectional and provides the raw bandwidth in the network. They may contain one or more physical or logical channels. They may be pipelined to achieve higher throughput and their only task is to transfer data between routing nodes.

- *Network Adapters* (NA) implements the interfaces presented to the blocks. The network adapter translates transactions on this interface into network communication. The advantage of using a network adapter is hiding the implementation details of the communication structure, the network, to the block. The blocks do not need any knowledge about the network to communicate. The computational units are decoupled from communication, which allow independent design and implementation and easier reuse.

The routing nodes and links are very similar to network nodes and edges known from general network theory and the network adapters introduce higher level of abstraction to the connected subsystems. These essential concepts are illustrated in Figure 2.2 along with an example topology.

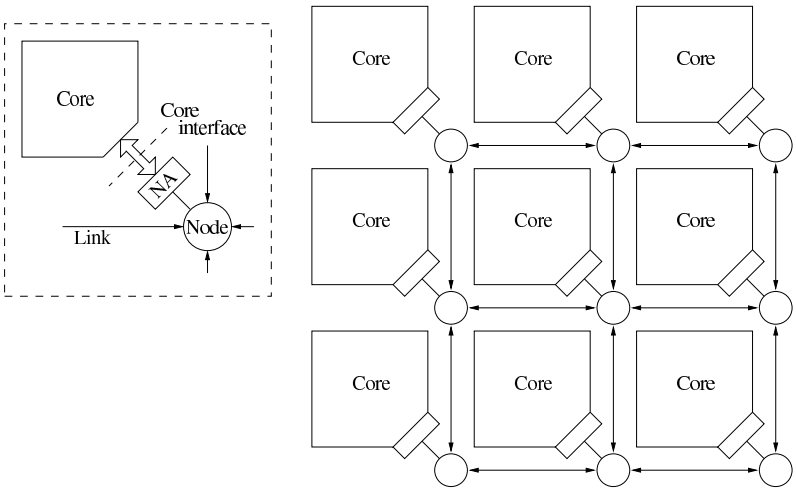


Figure 2.2: NoC components.

Routing nodes are spread across the chip and connected by links to form a network. How these are arranged is known as the network *topology* and the *protocol* specifies how these nodes and links are used. A network is defined mainly by its topology and the protocol it implements. The task of the network is to deliver messages from source to destination. The network provides hardware support for basic communication primitives, which can be used to implement higher level protocols. The task of the network adapter is to translate any command on its block interface into network communication in terms of the communication primitives provided by the network. The network adapter at the destination block converts received network request into the original command set up by the source block and presents the command at the destination block interface as illustrated in Figure 2.3. A network should essentially appear as virtual wires between the connected blocks.

The topology of the network is a very important design choice, which influences the support for traffic locality and the distance between blocks. Depending on the system, there may be many or few blocks to be connected to the network. If there

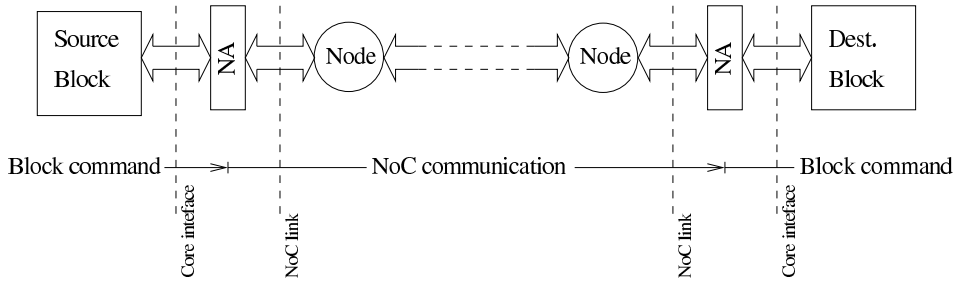


Figure 2.3: Hiding the NoC to the blocks.

is no prior knowledge on the what kind of communication to expect a more general topology should be chosen, but will imply a larger overhead in terms of unused network capacity. *Clustering* deals with localization in the network, both in physical and logical. Logical clustering can be very useful for programming applications using the network and be implemented as virtual wires. Physical clustering is physically arranging the blocks in a way to minimize global communication based on prior knowledge on the traffic. Additionally the topology has influence on the *reconfigurability* of the network. Configuration deals with how network resources are allocated. A topology with more connections and more bandwidth overhead is more flexible, than an optimized network with few links with high utilization.

2.2.1 Regular topologies

Most NoCs implement regular topologies that can be laid on a chip surface. This means that the topology has to map well into a 2-dimensional plane. This makes a grid based topologies a popular choice. Regular networks are homogeneous structures where power dissipation and area scales predictably for increasing size. They are based on one or just a few different routing nodes, and routing is simple.

A grid (k-ary n-cube) is an example of a regular network. Depending on the nature of the links, different grid topologies can be constructed. A mesh network has routing nodes placed in a rectangular structure and bidirectional links connect them to their four neighbors. A close related topology is the torus which is essentially a mesh with connected edges. The torus topology is demonstrated in [9] as interconnect for a multiprocessor chip is an example of a unidirectional grid topology. Most networks presented in NoC research are based the mesh topology using bidirectional links. Grid based networks are usually direct networks.

Other examples regular topologies explored in NoCs are k-ary trees and k-ary n-dimensional fat trees. In tree topologies local traffic is completely separated if the links are bidirectional, as it takes place within a single branch of the tree. A simpler tree based topology is a folded tree or mux/demux topology as known from CHAIN[2]. It has the advantage of very simple routing nodes and routing scheme. Fat tree topologies are more complex by having multiple roots. In tree based topologies, blocks are usually attached the leaf nodes creating indirect networks.

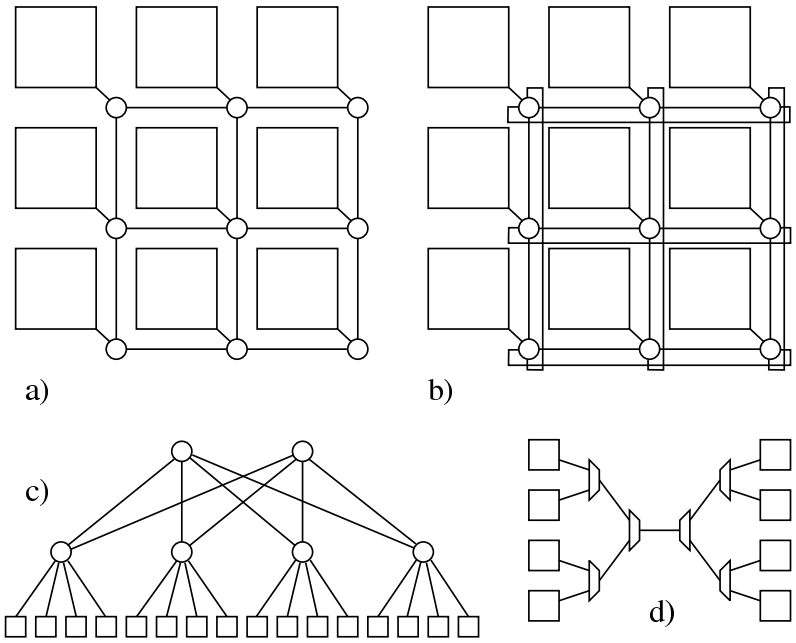


Figure 2.4: Regular NoC topologies. a) Mesh, b) Torus, c) Fat tree, d) Folded tree.

Figure 2.4 shows a few examples of regular topologies. In general mesh network provides good flexibility and link utilization, while tree based topologies have shorter paths and isolated local traffic. A detailed comparative evaluation of a set of recently proposed NoC topologies is presented in [10]. Three grid based topologies is compared to a ring based and two tree based topologies. The ring based topology does not scale well, but in a system with eight blocks any block can be reached within two hops. The choice of topology is a design trade off. Some topologies can sustain very high data rates at the expense of high power consumption and area overhead, while others provide a lower data rate at lower costs.

2.2.2 Irregular topologies

Irregular topologies are derived from mixing different forms in a hierarchical or hybrid fashion. An irregular topology does not scale linearly in area and power consumption. But on the other hand it can be used to create more specific topologies that fit the communications requirements of the system better and reduce overhead capacity.

A hierarchical network topology could use different topologies at lower levels in the hierarchy to optimize the network based on prior knowledge about the system. A different approach is to incorporate a sub-network into another topology, replacing the global topology in that region. The sub-network is seamlessly integrated into the global topology, but adds extra features or more resources to that region. This approach is proposed in [11].

The topologies mentioned above are all composed of one or more regular topologies. An extreme option is design a completely custom topology. Designing a network specifically for an application gives the best performance with lowest area overhead and power consumption, but requires much more design effort. The goal of most NoCs proposed until now is to provide a more or less general NoC platform, which can be used for a range of applications. However application specific optimizations as proposed in [12] and [13], show that much can be gained from custom networks. In regular networks it may be difficult to obtain high utilization in systems with many different blocks, but are on the other hand more flexible. The choice of using a general network or an optimized network is trade off between design cost, area overhead, power consumption and flexibility that must be done for the specific system.

2.3 Routing

Data is transferred through the network according to a communication protocol. *Switching* is defined as transport of data, while the protocol is the intelligence behind it that determines the path through the network. Comparing NoCs to the OSI model on networks as done in [1], the routing nodes must implement essential communication primitives which implement the protocol.

The protocol defines the use of available network resources. Important topics in protocol and routing node design addressed in NoC research are:

- *Circuit and packet switching.* In circuit switched systems resources are allocated and set up before use. They are hold until the data transfer is complete. This is opposed to packet switching, where data is just send through the network along with routing information.

- *Connection-oriented or connection-less.* Connection-oriented protocols involves setting up a logical connection before data can be transferred. In connection-less protocols no prior set up is required. Circuit switched systems are always connection-oriented, while packet switching can be both connection-oriented and connection-less.

- *Deterministic and adaptive routing.* In a deterministic routing protocol, the path between source and destination is found on the basis of their locations only. In adaptive routing, the path may be influenced by other factors. The path could be chosen to avoid congested links if the routing nodes have knowledge about the current state of the network.

- *Minimal routing.* A routing algorithm is minimal if it always chooses the shortest path between source and destination. This may not always be the best choice, especially when the routing scheme is adaptive.

- *Delay and loss.* In computer networks like the Internet, data is discarded in the network to loosen congestions and data must be retransmitted. A NoC is

a more controlled environment and makes it easier to prevent the situations to occur. Adding handling of data loss to the protocol will increase it complexity and cost significantly.

The majority of NoC research is based on packet switching networks. Usually connection-less communication is used for best effort traffic, while connection oriented communication is used to provide service guarantees.

A packet is a unit of data wrapped into a frame that may also contain extra information used by the network during transfer. This is usually placed before the data payload. As packets may be very large and vary in size it is useful to separate them in smaller parts. A *flit* is the smallest unit that can be handled by the network. A packet can span several flits, which can be transferred sequentially. Figure 2.5 illustrates the flit principle.

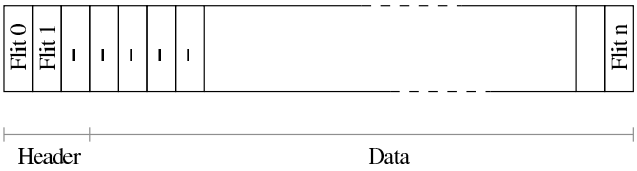


Figure 2.5: Packet and flits.

How the packets are passed between the routing nodes is known as *forwarding*. The forwarding strategy influences the buffering in the routing nodes and how resources are used in the network. The most common forwarding strategies are *store-and-forward*, *wormhole* and *virtual cut-through*.

Store-and-forward routing stores the entire packet before it is forwarded based on the information in the packet header. The routing node may stall if the receiver is not able to accept the packet.

Wormhole routing minimizes the latency by forwarding flits as soon as possible, creating a stream of flits making its way through the routing nodes and links like a worm. If the receiving routing node is unable to receive the flit, all routing nodes and links spanned by the packet may stall.

Virtual cut-through routing forwards packets by streaming flits, similar to wormhole routing. However to avoid stalling several routing nodes an links, virtual cut-through routing ensures that the receiving routing node is able to receive the entire packet before forwarding the first flit. If a packet is stalled it will aggregate in the buffer of the current routing node.

A very important part of the protocol is to provide flow control. Flow control dictates the movement of the packet along the path in the network. As there can be many packets in the network, they must be coordinated according to some strategy.

Resources should be shared in a fair way, so no packets experience starvation. Flow control may also include traffic priorities. Furthermore flow control must handle deadlocks in topologies prone to deadlocks. Deadlocks occur when the network reach a state where packets block each other by circular dependencies. Deadlocks

can be handled when they occur, but the simplest solution is to completely avoid deadlocks by design. Solving deadlocks require either handling of data loss or some other roll-back mechanism, which are both complex and expensive.

Virtual channels are used for sharing physical channels by several logical channels with individual and independent buffers. The use of virtual channels have a number of advantageous uses which includes avoiding deadlocks, optimizing wire utilization, traffic priorities and improving performance.

When a virtual channel stalls, it will not stall the physical link, which can be used by other virtual channels. This reduces the amount of stalls in the network significantly. Additionally virtual channels are very useful to prevent deadlocks.

In some network topologies is it possible to avoid deadlocks by choosing the path carefully. Others require virtual channels to avoid the possibility of circular dependencies. Routing and deadlock avoidance are described in more details in [14] chapter 10.

2.4 Services

The basic service provided by a network is transfer of data from source to destination. However demanding blocks may impose more strict requirements on the network than eventually performing the data transmission.

Quality of Service (QoS) is defined as service quantification that is provided by the network to a demanding block. The service could be low latency, high thoughtput, low power, etc. Such services must be negotiated with the network in order to allocate resources needed to provide the service.

Generally speaking services can be put into two classes: Best effort (BE) service and guaranteed services (GS). BE provides no guarantees at all, while GS do give do. In context of NoC research BE refers to traffic where only correctness and completion are guaranteed. In other words BE traffic will eventually arrive at its destination. NoC GS means traffic for which additional guarantees are given.

In macro networks service guarantees are usually statistical guarantees, as the hard guarantees require tighter coupling of the nodes in the network. In smaller systems like SoCs, hard guarantees are preferred and easier to implement as the communicating nodes are tight coupled and the network is limited in size. In SoCs the network will be static in most cases. No blocks will be physically added or removed during operation.

Hard guarantees require logical independence of other traffic in network, which means that GS must have independent resources in the network. GS requires connection-oriented routing to allocate these resources. GS connections are virtual circuits, which uses independent resources. These circuits may be implemented using virtual channels, time-slots, etc. Because of the independence requirement, more GS means a lot more network resources and increases the cost of the network significantly.

GS allows analytical system verification and is valuable tool when network resources are analyzed and distributed among the communicating blocks. Concerns

regarding traffic interference and other unpredictable conditions can be removed by GS. However GS does not utilize the network efficiently. The maximum number of simultaneous GS the network can support is determined by the worst case use of each GS set up in the network. The network will only be utilized fully, when all GS reservations are fully loaded. The utilization may also be limited by different kinds of GS influencing each other.

In order to make use of the overhead, that can not be avoided in NoCs supporting GS, it can be combined with BE. Idle resources that are reserved for GS can be used by BE traffic without affecting the GS. This approach is taken in most proposed NoCs that supports GS. [15] discusses the design of a GS router for the AEthereal[6] NoC, which supports BE too. BE traffic is handled at lowest priority and uses whatever resources are available, to fill up the gaps between the GS traffic.

If the traffic in a system is deterministic, GS may not be necessary to provide QoS. If all traffic can be predicted a NoC supporting BE should be sufficient in most cases. The communication can be scheduled during design to fit into the NoC. This approach is taken in [13] where QoS is guaranteed by mapping and planning the NoC using a traffic pattern obtained by simulation. The method presented is restricted to system with no or very little variation in the input streams.

Adding support for priorities to BE traffic makes it possible to plan the traffic in the network, even though not all traffic is known before hand. However it may not be possible to analyze or simulate all situations and GS using analytical verification is then the only option.

2.5 Programming model

Programming the network has two aspects: How to configure the network and how to use it. The network implementation is hidden to the blocks by the network adapter, but the interface presented to the block can handle this abstraction in different ways. A popular way of handling communication between processors and peripherals is memory mapping. Blocks are assigned to specific addresses in a global address space. Data reads and writes to these locations are transparently transferred to or from the particular system. This approach is intended for systems with one master and one or more slaves.

Systems with more blocks acting as masters can communicate using shared memory. This is not efficient for systems with close cooperation between blocks, as the shared memory will soon become a bottleneck. Using message passing instead will send data directly between communicating blocks avoiding the shared memory. Ordinary wires can be viewed as simple message passing and can easily be mapped into a network. The network adapter then supports passing messages to particular blocks instead of mapping them into a memory address. This still does not require any knowledge about the network, just the blocks available.

The network have to be configured initially to be able to hide its implementation to the blocks. The network adapters have to be configured to know where to send

data, as the blocks do not. This task has to be handled by one or more blocks that know the network and how to program it. A typical solution is to have a programming unit or responsible block, that programs the network adapters initially or when it is necessary.

A network may be reconfigured for different tasks. Each scenario is known as a *use-case*. The use-case describes the communication requirements of that particular task. A use-case can be illustrated by constructing an application graph as shown in Figure 2.6. The nodes of the graph represent processes in the application and the edges represent the communication between the processes. Mapping these graphs onto the NoC topology is known as *application mapping*. Systems may have a set of use-cases, and switch between these often, which mean reconfiguring the NoC. All use-cases may not even be known when designing the system. The NoC must be designed to handle all expected use-cases, which will have unused network resources in some use-cases.

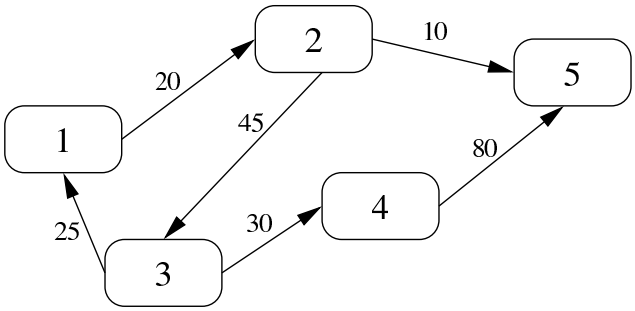


Figure 2.6: Application graph example.

Chapter 3

Digital Hearing Aids

This project is a case study of an existing system and an introduction to the system is therefore necessary to gain enough knowledge about the system to make proper design choices. This chapter will give a general introduction to digital hearing aids and then describe the digital system, which has the focus in this project. The system description will concentrate on the data flow in the system and general system behavior, which are relevant to communication structure design.

The system is the latest digital back-end chip used in the latest generation of wearable digital hearing aids by *Widex A/S*. The back-end system handles all digital signal processing in the hearing aid device. The back-end chip is interfaced to an analog frond-end chip, which performs preliminary analog signal processing. The current design is very application specific and optimized for low power consumption and area. That means specially designed hardcode blocks with limited configurability. The area is restricted by the fact that it has fit into hearing aids of very small physical dimensions. Hearing aids vary in sizes from rather large behind the ear models, to completely in canal models that can not be seen when worn by the user. The size of the current back-end chip is limited to approximately 2.5×3.5 mm.

Hearing loss is a wide spread problem and is target for intensive audiological research to improve hearing aids to help more people and help them better. This research leads to new and better algorithms for signal processing in hearing aids. A revised or new back-end system is required to support these new algorithms and features, which then becomes the next generation of hearing aid products.

As algorithms and features become more and more advanced, the requirements for the supporting hardware platform increase similarly. The system is currently developed as a very tightly coupled system, consisting of several communicating subsystems. Communication is mainly done using point-to-point connections and shared buses. New application features often requires new subsystems to be fitted into the system and additional communication is therefore introduced.

3.1 Introduction to Hearing Aids

To obtain a better understanding of the application and the supporting system, a quick introduction to digital hearing aids is given here.

The purpose of a hearing aid is to amplify sound in a way that it compensates for the user's hearing loss. This is done by recording sound using a microphone, amplify and process the sound electronically and deliver the amplified sound into the ear of the user, using a speaker. The basic idea of a digital hearing aid is illustrated in Figure 3.1.

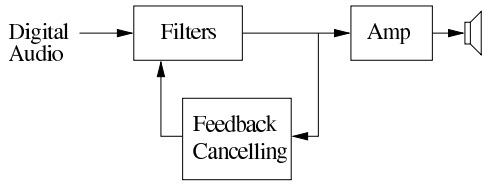


Figure 3.1: Basic data flow in hearing aid

The sound amplification and processing is not a simple task, and is under constant development. Previously hearing aids were implemented as analog electronic systems. Today digital signal processing has by far overtaken analog processing. Both filtering and amplification is done digitally in modern hearing aids.

Hearing aids are programmed to fit every user's special needs, to give the best possible experience. Looking at a digital hearing aid at a high level of abstraction, the system has a few major components. The signal from the microphone is passed through a front-end, which handles the analog to digital conversion. The digital audio is then passed on to the digital back-end. The audio is represented by samples, which are passed through a range of filters and amplified according to the needs of the particular user.

The audio signal is split into several frequency bands, which are treated separately and then added up to form the speaker output signal. Because the microphone and the speaker are situated millimeters apart, the system is very prone to feedback. This is handled by a special feedback canceling system, which plays a significant role in the signal processing done in the back-end system. Feedback canceling is done using the digital output signal, which is fed back through the system and used to create a compensating signal. This means that there are more than just data flowing straight from input to output. Various signals are fed back through the system too.

3.2 The back-end chip

The back-end system is a separate integrated circuit that is interfaced to the front-end, external memory and the speaker. The system configuration, including fitting settings for the particular user, is stored in the external memory and loaded into the back-end system at start up.

3.2.1 System architecture

The back-end system currently consists of 12 subsystems, as illustrated in Figure 3.2. These subsystems have very different tasks and vary significantly in size and communication requirements. Subsystems directly involved in the signal processing are in general larger and communicate more than the other units. The audio streams are passed between the subsystems as samples. In some parts of the datapath, the audio signal is treated in bands. Every full spectrum audio sample is split into 15 samples, each representing a frequency band. This gives 15 times as much data to process and communicate through the system. Modern hearing aids use two microphones in order to suppress background noise and locate the sound source. Parts of the system use samples from both microphones using two separate signals. In other subsystems only single samples or parameters are used.

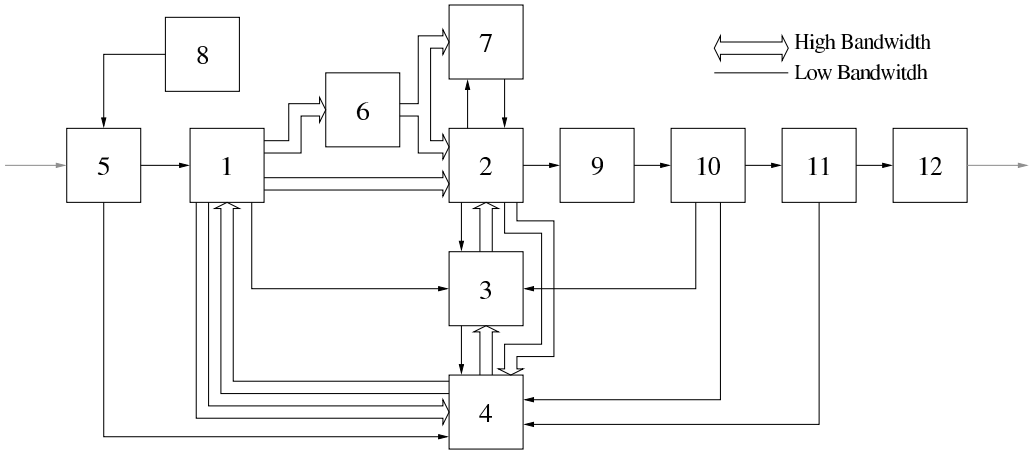


Figure 3.2: Traffic pattern. High bandwidth is indicated by bold arrows.

The back-end system is implemented as a globally synchronous system with a main clock domain and a few slower clocks running at frequencies divided down from the main clock. The main clock frequency is faster than the sampling rate, which allows subsystems to spend several clock cycles processing each sample, without having a deep pipeline. To keep the power consumption as low as possible, subsystems with a longer acceptable latency are using divided clock to minimize the switching activity. Subsystem may even be disabled by clock gating, when they are not used.

A hearing aid can be categorized as a real-time system. Audio must be processed and delivered to the speaker within a certain deadline. Too long processing time is inconvenient for the user. The delay from things happen to the sound is delivered to the user must be unnoticed. However a fairly large delay can be tolerated. It is more important that the delay is constant, to avoid having large buffers in the system.

When the system is configured during start up or at mode switches, speaker output is disabled. A delay in the order of milliseconds is acceptable by any user and is

not critical.

The hearing aid has different modes of operation for different situations that can be chosen by the user. For each mode the subsystems may need to be reconfigured. This involves loading in new configurations from the external memory and distributing the new configurations to the subsystems, for use in the new mode.

The internal design of the subsystems is not discussed in further details here, as they are not relevant to the communication structure. One of the basic ideas of NoC's is decoupling computation and communication.

3.3 Internal back-end communication

The communication between subsystems in the current design is handled by point-to-point connections. In addition to the signals in the datapath, a separate configuration bus is used to configure the subsystems.

The traffic in the system is deterministic. For every audio sample period the communication in the system is the same, when no mode changes occur. The delay through the back-end is a few sample periods, so more than one sample is concurrently being processes in a pipelined fashion. But every time a sample enters the back-end, processing starts over and is exactly the same as for the previous sample. This creates a constant flow of samples through the back-end system.

In general every signal has dedicated wires in the current design. As the system clock is faster than the sampling rate, wires are saved by time multiplexing the audio bands on one set of wires, which is able to transfer one audio band at the time in the parts of the system where audio is treated band-wise. Just as the band-wise information can be transferred sequentially, computational hardware is minimized by sequential processing of each audio band.

The back-end system operates on data streams. There is no shared memory involved in the signal processing and no need for a global addressing space. Dedicated wires mean no addressing is required. Because the processing repeats for every sample, each subsystem knows exactly what piece of information to expect next. No explicit identification of data received from other subsystems is necessary, except when bands are multiplexed.

3.3.1 Traffic types

Looking at the system from a communication point of view, it is possible to define a few different types of traffic with different requirements. The majority of the communication in the system is audio samples. Besides these are parameters, which are sent between the subsystems regularly, but not necessarily for every sample. During reconfiguration and mode shifts a third kind of traffic occurs, when new configurations are distributed to the subsystems.

Audio samples

Samples occur periodic at the rate of the audio sampling at the input and must be delivered to the speaker at the same rate. An audio sample is a relatively small amount of data, but processing generates much more data on the way through the system, before it is reduced to a single audio sample again at the end. The band-wise audio processing effectively generates 15 times as much audio data.

Parameters

Parameters are passed between the subsystems during operation to coordinate the behavior of the subsystems. Some parameters have hard deadlines similar to audio samples. Other parameters have soft deadlines that can be missed occasionally. If a filter parameter is updated for one sample or the next does not change the big picture. The deadline is not necessarily tight, but its is more important to know how much it can be missed. The bandwidth requirements of the parameters vary significantly. Not all parameters are passed for every audio sample, which means that the amount of parameters transferred between the subsystems is not the same for every sample period, but varies periodically.

Configuration paramaters

Configurations are written into the configuration registers in the subsystems during system initialization and mode shift. This is all handled by the configuration bus controller using the bus. At initialization all subsystems have to be configured, which generates a large amount of communication, while at mode shift only some subsystems have to be reconfigured. In general there is no deadline on configuration. But is has to be done within reasonable time. Speaker output is disabled until all systems have been configured.

3.3.2 Communication protocol

As the back-end system communication structure is based on dedicated wires and the subsystems always knows which data it will receive next, no advanced communication protocol is really needed. A simple data valid signal is used to indicate when new data is sent on the wires as illustrated in Figure 3.3.

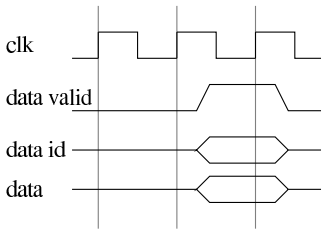


Figure 3.3: Communication protocol

A situation where the subsystem is busy and can not receive the next data sample, will never occur. Samples will always arrive in the same pace, which the system is designed to handle.

3.3.3 Traffic patterns

Communication within the system is far from evenly distributed among the subsystems. Some subsystems are connected to several other subsystems using high bandwidth, while others are having only a few connections. The system communication is illustrated in Figure 3.2. Because not all subsystems communicate with each other, locality exists in the traffic pattern. Very high bandwidth is required to some of the neighboring subsystems, while far less bandwidth is required to other subsystems.

The traffic density is not constant over time. The main flow is always active, but more communication takes place during mode shifts. As mentioned before, some signals are not active in every sample period, which causes the traffic density to vary in time.

A few signals are distributed to more than one subsystem effectively implementing a signal multicast.

The back-end system has four subsystems, which most communication takes place in between. These perform band wise signal processing. Two audio inputs are used for advanced filtering. Both of these are split into bands, which generate large amounts of traffic between these subsystems. The size of an audio sample is not constant throughout the system, but is general around 20 bits of data. The four main units are described in the following.

- *Unit 1* performs the band splitting. It has two audio samples as input and some parameters from Unit 4. It has two signals for the output samples to Unit 2 and 3. Parameters are passed for some bands to Unit 4 generating an amount of traffic similar to the two sample outputs. A small parameter signal is passed to Unit 3.

- *Unit 2* performs processing of each audio band received from Unit 1. Parameters are sent and received from Unit 3 and 4, but generate only a small amount of traffic compared to the sample inputs.

- *Unit 3* is the most busy unit in the system in terms of traffic. Large parameters are received from Unit 4 on two broad signals almost every clock cycle. A few parameters are sent and received from Unit 2 and 4. One 15 band audio data signal inputs audio samples from Unit 1.

- *Unit 4* gets parameters from many of the smaller subsystems and Unit 2. Parameters are sent and received from Unit 1. Parameters sent to Unit 3 generates by far the most traffic.

The signals already have a high utilization, which makes it hard to save wires by more sharing. These are targets of much optimization to reduce the power consumption as much as possible.

The bandwidth used for parameters between Unit 3 and 4 is approximately twice as big as the bandwidth used by any other subsystem. Table 3.1 below, shows the total input and output traffic for each subsystem, for every sample period. This includes all traffic under normal operation.

Unit	Input	Output
1	441	1629
2	801	593
3	2602	408
4	3137	2356
5	40	82
6	300	600
7	321	21
8	0	8
9	32	20
10	20	46
11	20	40
12	20	0

Table 3.1: Traffic to and from each subsystem. Off chip communication is left out. All numbers are total amount data in bits per sample period.

From a NoC perspective the Table 3.1 is the bandwidth requirement of interface between subsystem and network. The traffic is considered as connections between subsystems in Table 3.2, which reveals other important details about the traffic pattern. The number of connections between subsystems and bandwidth requirements are important in relation to NoC design.

3.3.4 Programming

Each hearing aid is fitted to the particular user by storing personal configuration parameters in the external memory in the hearing aid. The parameters are loaded into the back-end system when the hearing aid is switched on along with other parameters.

The parameters are loaded into special registers in each subsystem. When the user switches the operation mode of the hearing aid, other parameters are loaded into these registers to reconfigure the hearing aid.

The configuration registers are programmed using the configuration bus shown in Figure 3.4. The subsystems have a special configuration interface connected to this bus, which allows both read and write to the registers located in the subsystems. The bus is controlled by a single controller. The controller handles input interface to the bus, and all bus transactions goes through the controller. Reading parameters

Conn.	Bandwidth	*Conn.*	Bandwidth
1-2	300	5-1	41
1-3	309	5-4	1
1-4	720	6-2	300
1-6	300	6-7	300
2-3	324	7-2	21
2-4	216	8-5	8
2-7	21	9-10	20
2-9	32	10-11	20
3-2	180	10-3	13
3-4	228	10-4	13
4-1	400	11-12	20
4-3	1956	11-4	20

Table 3.2: Traffic between subsystems. All numbers are total amount data in bits per sample period.

and logs from subsystems is handled by the controller, which uses a simple address signal to specify which subsystem it is communicating with. As this bus is designed to communicate between the controller and the subsystems, it is not very well suited for communication between subsystems, but it is possible. The configuration bus is idle most of the time under normal use of the hearing aid. However, data may also be extracted from blocks occasionally for logging using the configuration bus.

The configuration bus is the only flexible communication structure available in the current system design. It is designed for programming the hearing aid and collecting logged data only. However it can be used for some simple fixes, where parameters have to be moved from one system to another. But the flexibility is very limited.

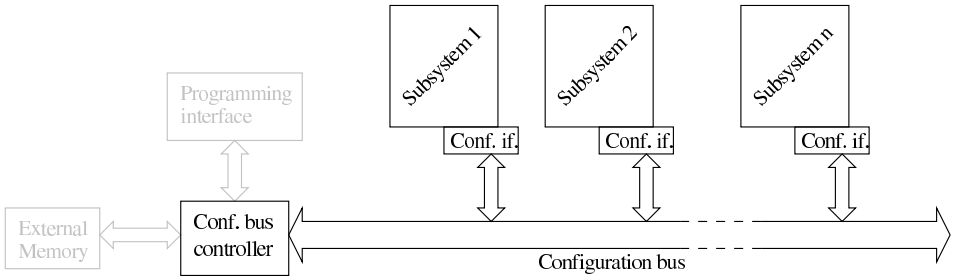


Figure 3.4: Configuration bus

The subsystems are designed in a way that the order of configuration, does not influence the system. Subsystems that can reach an illegal state if configured in a certain order, are self resetting when all configuration parameters are received.

3.4 Low power design

The back-end system has to fit into even the smallest hearing aids, which restricts the physical size of the system. The current design is approximately $8.75mm^2$. Some effort is put into area optimization to fit the design into this small area.

The main focus of optimization is power consumption. The system is powered by a single battery. The goal is as long operation on a battery as possible using the smallest battery as possible.

Low power consumption is obtained by considering power consumption in all phases of the design and implementation, from transistor to system level. At transistor level, a very energy and area efficient standard cell library is used for smallest possible footprint. Clock gating and slower clocks minimize the switching activity. During synthesis the design is optimized for power, to further reduce the power consumption.

Because of these very tight constraints, the system is very tightly integrated to save as many transistors as possible. When the back-end is designed, it is designed to support the current snapshot of algorithms developed for signal processing. No further improvements are introduced later. Late changes and bug fixes then have to be solved by ad-hoc solutions, which may not always be easy.

3.5 Future hearing aid systems

As CMOS technology progresses and gives design capacity for new and more advanced audio processing systems, a more flexible communication structure is wanted. The number of blocks increase and thereby also the communication needs. Instead of creating even more dedicated wires that are only used in special cases, a general communication structure is better solution.

Having a programmable communication structure, late changes to algorithms will be easier to accommodate, as it is only a matter of configuration to introduce new signals between subsystems. This assumes that the block designs will be changed similarly to exploit the flexible communication structure.

Subsystems may not be needed in all modes and can be bypassed and turned off in a more flexible system, instead of adding some pass through path in these subsystems.

Future generations of hearing aids may include more general processing units, that can be used for different tasks different places in the system, in different situations. A general processor could be introduced to take over the jobs of smaller subsystems. The order in which subsystems are used may be changed and rearranged in different modes. This requires that the system is able to redirect data and parameters between subsystems in a more flexible way, than is possible in the current system. These situations are illustrated in Figure 3.5.

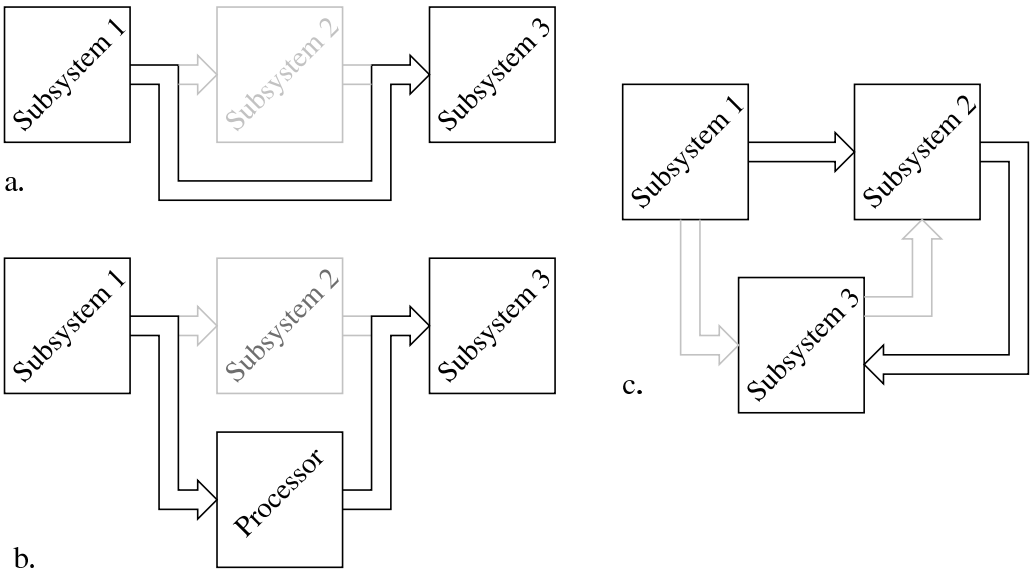


Figure 3.5: Configurable communication. *a.* Bypassing subsystems. *b.* Moving tasks between subsystems. *c.* Changing the processing flow.

Chapter 4

Design

This chapter will discus the design of NoCs for use in small lower power systems. In particular the back-end chip designed by *Widex A/S*.

From the introduction to NoCs given in Chapter 2, it is obvious to look for a network structure to provide a more flexible communication system. Alternatives are using more buses or multiplexing dedicated wires. Both solutions do not scale well and does not offer the same level of general flexibility as a NoC.

Before looking into the design of a NoC for the back-end chip, the requirements for the NoC must be specified. The design choices and trade offs to be made are based on the introduction to the system is given in Chapter 3.
After specifying the communication requirements, designs are discussed and proposed. The design of NoCs involves several choices and trade offs to be made. The design trade offs are strongly correlated and can not be considered separately. However, each topic will be discussed in separate sections here and then summarized and put together in the final section to form the network design.

4.1 Requirements

Based the information about the back-end chip in Chapter 3 and the concept of NoC introduced in Chapter 2, it is possible to derive the requirements for a suitable network.

The overall goal is to propose a new solution, which supports the ideas for future back-end chips introduced in Section 3.5. This will add some extra cost to the system in term of area and power consumption. These factors are focus of much optimization in the current design, and a new system should strain to keep these factors as low as possible too. The solution has to be simple but yet suitable. An area overhead of less than 10% is estimated for other NoCs [8], which is also the goal for this network.

As seen in Figure 3.2, the back-end system has a general data flow from input to speaker output. This main flow will be preserved under any use, but the path in between may change in different use-cases.

The traffic in the system is deterministic, which means that it is possible to predict the traffic patterns in the system. During normal operation the traffic pattern is periodic, repeating itself for every audio sample. When the use-case is switched, a smooth transition is not essential. The output can be turned of during the transition and no data or connection has to be preserved.

The network has to be able to cope with the current bandwidth requirements in the back-end chip. But additional capacity is needed to have room for other system configurations. The amount of unused capacity for reconfiguration is a trade off between the costs and expected use of the network. The NoC should support rerouting any signal to any destination and adding new signals. A minimum requirement is redirecting or adding at least one 15-audio band signal.

The current design uses a global clock for synchronization. The clock frequency is low to reduce the power consumption. As the system is already globally synchronous a clocked interconnection structure is also preferred.

The increased latency by using a network, should be kept as low as possible. However, audio processing is quite tolerant to latency, as long as it is consistent i.e. no jitter. Because of the low clock frequency additional clock cycles spend in the network are expensive.

The network must be able to handle both data, parameters and block configuration. Furthermore it must be ensured that data and parameters are guaranteed to have bounded latency and no jitter to occur during normal operation.

To summarize the network has to be as cheap as possible in terms of area and power, while maintaining flexibility. The system is characterized as a real-time system with well known traffic patterns. Reconfiguring network without data loss is not crucial, a few samples can safely be dropped.

4.2 Network Topology

The topology has great influence on most aspects of network design. Routing and application mapping depend directly on the chosen topology. The topology is very important in terms of area, power dissipation and latency. The majority of networks proposed in NoC research are not targeted specific applications, but are developed as more general interconnects, for use in a variety of systems. However, in this project the NoC is designed with only the back-end system in mind. Any application specific optimizations can be used to bring down the costs of using the NoC.

There is no straight forward way to choose a suitable network, it is a complicated design trade off with great influence on the NoC design. An optimal solution is hard to find and may not exist.

The approach taken here is to take the traffic pattern of the current system into consideration, and find a topology that this pattern can be mapped into in an efficient way. The traffic pattern of the current system shows a great diversity in bandwidth requirements. The most bandwidth intensive block requires more than a factor of 100 more bandwidth than the least communicating block. Furthermore the traffic pattern

shows locality that can be exploited. The following lists possible topologies. Others exist though, but these are common topologies, which are good candidates for small NoC designs.

4.2.1 Tree topologies

One approach is a tree based topology as described in Section 2.2. The short overall path between any two blocks in a bidirectional tree means low power dissipation and latency. However, the root is a bottleneck. The back-end system main traffic flow will causes a significant amount of traffic through the root links. The traffic locality must be exploited.

Binary tree

The mux/demux topology has simple routing and cheap routing nodes, which are essentially multiplexers and demultiplexers as known from CHAIN[2]. However, traffic locality can not be exploited, which is problem because of the bandwidth requirements. The root link has to cope with the entire bisection bandwidth on a single link, which makes this topology infeasible.

Fat tree topology

The fat tree topology counters the root bottleneck by having multiple roots. The short paths of the tree topology are maintained and traffic locality can be exploited. Two examples of fat tree topologies are the butterfly fat tree topology shown in [10], which has routing nodes with two parent links and four child links, and the SPIN[3], which is more link intensive by having four child and parent links for each routing node. The disadvantage is more complex routing nodes because of the increased number of routing node ports, and a more advanced routing as there are multiple paths between two blocks. Due to complex routing, limited support for physical clustering and mapping this topology is not suitable for this application.

4.2.2 Grid topologies

Another option is to use a grid based topology as introduced in Section 2.2. Opposed to hierarchical topologies, there is no single bottleneck in the network, which makes it very flexible for use in systems with very different use-cases. Link redundancy means support for advanced routing schemes. Care must be taken to avoid deadlocks.

Mesh topology

The mesh topology has great support for physical clustering, as four neighbors are just one link away and mapping is not restricted by any hierarchy. It is possible to balance the traffic among all links in the network, even though it may not be possible for real applications. The downside is the number of relatively complex routing

nodes, which means a high cost. High network utilization is hard to obtain, because of the link redundancy.

Torus topology

The torus topology is a simpler grid topology. Unidirectional links means low routing node costs. The downside is that it does not benefit from local traffic and routing is restricted. Lack of physical clustering has an impact on latency, which makes it less suitable for this application.

4.2.3 Custom topologies

A different option is to use a custom topology, where topology and mapping are integrated. Reducing the network resources to match the requirements of the application and thereby compromising the overall flexibility can reduce the NoC costs significantly.

In systems with well defined use-cases this optimization is obvious. Especially single use-case systems can reduce the network overcapacity to a minimum, while systems with many use-cases will have some overhead to support all use-cases. Knowing the exact system requirements is very important. For this application the exact system traffic is known and a level of overhead for flexibility has been specified.

Custom topologies may be optimized topologies or designed from scratch. The following lists different approaches and optimizations.

Custom from scratch

A custom topology can be designed from scratch inspired by the task graph of the application. But it will become a complex task in systems with multiple use-cases. A fully custom topology is an unstructured approach, which requires a huge design effort, which may not pay off in the end. It is a more safe choice to rely on optimizations for well understood topologies, which makes this option irrelevant to this application.

Optimized topology

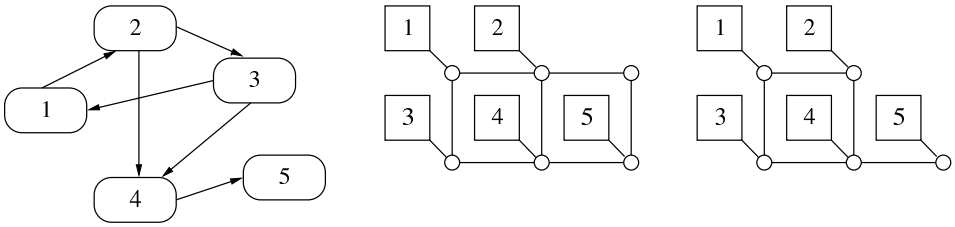
An optimized topology based on a regular topology is another option. The approach is introduced in [4] and [13].

In [13] the NoC design flow has three phases illustrated in Figure 4.1.

1. In the first phase the application is mapped to into hardware blocks and traffic model is developed for initial simulation from which traffic characteristics are obtained. These characteristics are used to generate graphs representing the traffic flow between the blocks and bandwidth constraints.

2. In the second phase the blocks are mapped onto a generic topology. The mapping considers several design objectives, such as minimizing power consumption and hop delay.

3. In the last phase post optimizations are carried out. In this step redundant switches, ports and links are removed. The result is a custom topology derived from a generic topology in order to save resources, while still meeting the application needs.



1) 2) 3)

Figure 4.1: Deriving custom NoC topologies. 1) Application graph, 2) Mapping to generic topology, 3) Post optimization.

Multiple blocks per node

Another way of reducing NoC costs is to connect more blocks to each routing node, and thereby reduce the network size. Blocks with low bandwidth requirements can share a port in routing node or use any unused ports left over from topology optimization or unused ports in nodes along the edges of the network. Both approaches are illustrated in Figure 4.2. The disadvantage is that the NAs must be able to interface all ports on the routing node, and the routing protocol will be slightly more complex. This approach suits the application very well because of the different bandwidth requirements among the blocks.

Multiple connections per block

Another option to reduce the overall NoC costs is to avoid network hot spots, which requires significant more bandwidth than the system average. This can be done by splitting the traffic among more interfaces to reduce the bandwidth requirements on links and NAs as seen in Figure 4.3. This will not change the overall network topology but care must be taken when mapping the application onto the topology.

Using multiple connection points to solve bandwidth issues for single blocks, is simpler than scaling up the network resources locally. This optimization works very well with the application as signals are already completely independent in the current design, thus they can easily be distributed among more network interfaces. Each connection can be treated independently except that physical placement of the block in relation to the network must be considered.

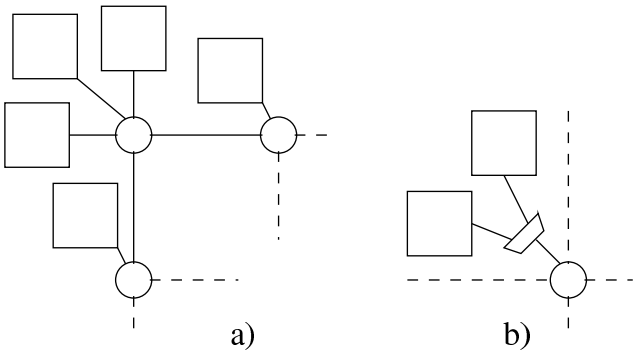


Figure 4.2: NoC optimizations. a) Connecting blocks to unused ports. b) Sharing ports.

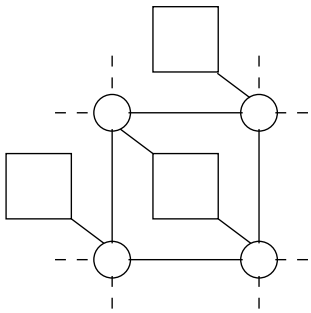


Figure 4.3: Connecting a single block to two nodes to load balance the traffic across two network interfaces.

Hybrids

It is also possible to use a hybrid solution. Special local requirements may be satisfied better by using a different local topology or a dedicated signal, than scaling up the entire network to handle this special situation. It can be viewed as a special purpose region in the network, which is completely transparent. The disadvantage of this approach is limited flexibility.

4.2.4 Design choices

Based on the previous discussion on different approaches on NoC topology and the requirements, a design choice has to be made. As the main reason for using a NoC in the system is to provide more flexibility in future systems, the topology should be as flexible as possible at the lowest possible costs.

Generic topologies offer most flexibility, while optimized topologies are most cost-efficient. It is therefore chosen implement two topologies for comparison: A generic *mesh topology* and an *optimized topology* based on a mesh topology.

Because of the great variation in bandwidth requirements of the blocks, both topologies will be supported by a small number of point-to-point connections. Furthermore both topologies will use interface splitting to even out the bandwidth requirements further.

The mesh topology offers the best flexibility and physical clustering, though it is expensive. The traffic requirements of the system are well known, which is a good foundation for optimization. Both topologies are based on the same routing nodes and NAs. The optimized topology will reduce the cost of a generic mesh topology by connecting multiple blocks to each routing node and remove unused routing nodes and links as proposed in [13]. The optimizations are closely related to the application mapping, which is discussed in the following section.

4.3 Application mapping

Application mapping determines how the application is mapped onto the NoC topology, i.e. how the blocks are arranged. The application is described by its use-cases represented by application graphs as introduced in 2.5. The problem of embedding one graph is intractable and is a well studied problem in literature with many heuristic algorithms available.

The back-end system contains different traffic types. These are considered equal during mapping to ensure predictable latencies for both audio samples and parameters. However, configuration parameters are not considered as they mainly occur during mode shift. Reconfiguration is discussed in Section 4.4.

Mapping can be done in various ways depending on the main objective, like minimizing resource requirements. Mapping strategies are described in the following.

Bandwidth constrained mapping

One strategy is bandwidth constrained mapping, which maps the use-cases onto the topology minimizing the link bandwidth requirements. Methods for bandwidth constrained mapping application graphs to generic topologies are presented in [16], [13] and [17].

In [16] an algorithms for mapping a single application graph onto a mesh topology is presented. The first algorithm uses minimal-path routing between blocks in a mesh topology and the second refined algorithm uses traffic splitting to distribute the traffic more even across the links in cases where multiple shortest paths exists. As mentioned in the previously, mapping and topology optimization can be combined for further improvement, which is done in [13].

A methodology for mapping multiple use-cases onto a NoC architecture is presented in [17]. Mapping such an application onto a NoC is a tedious task to do by hand. Instead use-cases are grouped to form compound use-cases and the bandwidth requirement is then the maximum of all bandwidth requirements of all involved use-cases. The mapping algorithm presented in the methodology scales the network to

fit by iteratively mapping the most bandwidth intensive path one by one from each use-case.

Power and latency optimization

Other optimization parameters are also possible during application mapping. An energy and performance aware approach is taken in [18]. An algorithm is proposed, which maps blocks onto regular topologies minimizing the power consumption under specified system constraints. The dynamic power consumption of the NoC is caused by the traffic. A coarse model of dynamic NoC power consumption can be expressed as follows.

$$E_{packet} = 2 \times E_{NA} + n_{Hops} \times E_{Node}$$

The power consumption of the NoC-links is lumped into E_{Node}, which makes the power consumption of the NoC directly proportional to the number of hops between source and destination. Reducing the number of number of hops saves power and reduces the NoC latency. It can be done by using minimal routing and by using topologies with more connections between routing nodes.

4.3.1 Design choices

Bandwidth constrained mapping has been chosen due to the minimal bandwidth requirements. However, this choice does not exclude power and latency from consideration. Bandwidth constrained mapping will also reduce power and latency, as the most bandwidth intensive blocks are mapped close to put load on fewer links.

Two topologies have been chosen previously, which the application graph representing the system has to be mapped onto. The application graph representing the current back-end system under normal use is shown in Figure 4.4.

As described in Chapter 3 the traffic between the four main blocks has by far the largest bandwidth requirements. How these are mapped onto the topology is essential to the system performance and the demands imposed on the NoC. It was chosen in Section 4.2.4 to use a few point-to-point signals for the most bandwidth intensive signals and use two NAs for the four most bandwidth demanding blocks. Removing these from the application graph and connecting the four main blocks to two routing nodes results in a modified application graph shown in Figure 4.5.

In certain situations data may extracted from blocks for statistical purposes. This additional data can interfere with the existing traffic in the network and cause increased latency. This can be treated as special use-case, but as latency for logging purposes is not important, the traffic can be routed along links with low utilization to limit the performance impact on the system. Data logging is not treated separately in this design, but is considered as part of the reconfiguration overhead included for new signals.

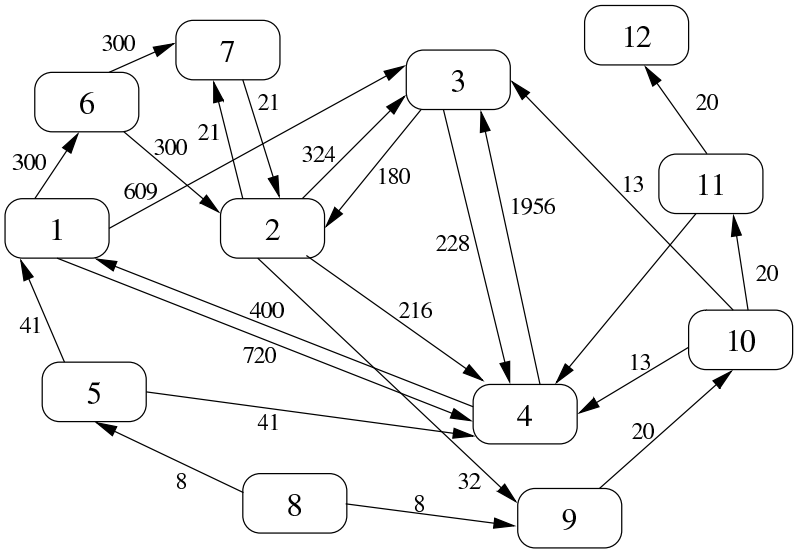


Figure 4.4: Application graph. All numbers are total number of bits per sample period.

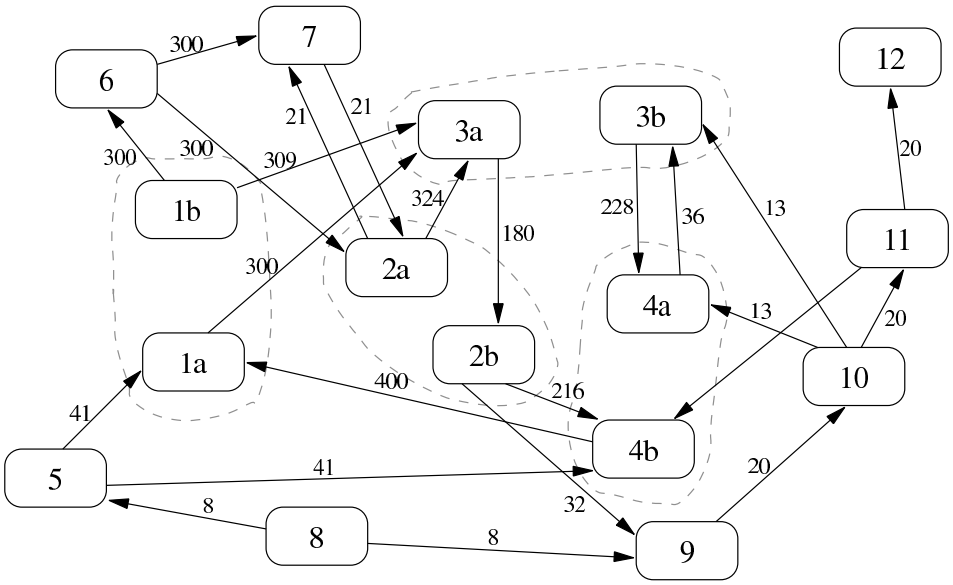


Figure 4.5: Application graph where block 1, 2, 3 and 4 are split into to two network interfaces. All numbers are total number of bits per sample period.

Application mapping onto the optimized topology more complicated than mapping onto the mesh topology because of the topology integration into the mapping. The following describes how mapping is performed on the two topologies.

Mesh topology mapping

The mapping is done by connecting the most bandwidth demanding blocks to neighboring routing nodes. Then mapping blocks with less requirements, to keep bandwidth intensive communication local. In the blocks using multiple connections, each connection is placed relative to the block to minimize bandwidth requirements.

Optimized topology mapping

Application mapping and topology are integrated in the optimized NoC. The overall structure is mesh-like, while the number of blocks connected to each routing node is determined by the application mapping. The mesh is sized to have enough unused ports for all required NAs. The mesh topology leaves three unused links at corner nodes and two unused links at nodes along the edges, while only one link is available at links embedded in the network.

The mapping is done iteratively by selecting groups of two or three blocks with local communication in the application graph and assigning them to nodes in the network. The sum of their global communication, additional capacity for redirecting or adding extra signals later, and global traffic passing through the node must not exceed the capacity of any link connecting the node to the network. All NAs on blocks with multiple connections must be placed in the vicinity of the block.

$$B_{link} \geq B_{global} + B_{pass-through} + B_{extra}$$

In nodes connected by more than one link to the network, the distribution of global traffic among the links, depend on the mapping of destination blocks. The condition above should hold for any link in the system, when all blocks have been mapped onto the NoC. Any unused routing nodes are removed, while all links on the remaining routing nodes are preserved for better flexibility and routing.

This mapping strategy saves area by reducing the number of routing nodes. It reduces latency as the network size is smaller and fewer hops between the blocks. Reduced distance also means lower power consumption. A higher utilization of the routing capacity of the routing node is expected. Instead of mapping demanding blocks on neighboring nodes, these should be mapped to the same node instead. The routing node traffic is illustrated in Figure 4.6.

4.4 Services and features

The services of the NoC are the guarantees offered by the network by design. Fundamentally a network is expected to eventually deliver data at its destination uncorrupted. But in context of the back-end system real-time guarantees are interesting.

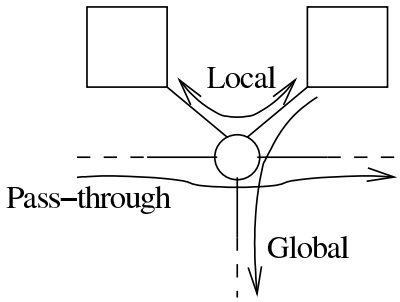


Figure 4.6: Routing node traffic with multiple NAs attached.

The design costs are very crucial for this application, which correlates the choice of services with the cost of their implementation. Choice are therefore made here on basis of design options discussed later in Section 4.5.1

As described in Chapter 3 traffic in the back-end system is samples and parameters pushed through dedicated wires between the blocks. This means minimal latency of data transfers and the exact clock cycle where data arrive is known. Removing the point-to-point signals and replacing them with a NoC will introduce some extra communication latency. Added latency is not a major concern, but predicable latency and low jitter are. If the network causes too much jitter, extra buffers will be needed which will increase the costs significantly. The design options are discussed in the following.

4.4.1 Guaranteed service

It is possible to provide hard guarantees in a tight coupled network as a NoC. GS requires connection-oriented communication for network resources reservation. GS can be thought of as a mere tool for system planning and traffic management. Traffic planning is an easy task using a GS enabled network when resources, such as bandwidth, are reserved.

However, it will be clear from the later discussion in Section 4.5.1 that connection-oriented communication and resource reservation leads to more complex network routers.

4.4.2 Best effort service

Opposed to GS, BE does not offer any hard guarantees. However, similar network performance is obtainable without using connection-oriented traffic with bandwidth guarantees. By using bandwidth constrained mapping as introduced in [16], traffic can be planned in NoCs that are not GS enabled if the use-cases are well defined.

Thorough traffic planning requires deterministic traffic to exactly define the use-cases. In systems containing non-deterministic traffic sources, precise traffic prediction is not possible. In these cases explicit hard service guarantees is the only way to ensure network performance.

Introducing prioritized classes of BE traffic, allow high priority traffic to get through congested parts of the network faster. However, having a slight overhead network capacity will decrease the difference in performance of the priority classes, as there is room for everyone.

4.4.3 Use-case switching

An important design consideration in relation to NoC services, is how services are set up. A major concern in recent NoC research is reconfiguring the NoC when changing use-case. This is mainly a problem in connection-oriented communications such as GS, where connections have to be set up and taken down without data loss, for a smooth transition between the use-cases. For connection-less communication the use-case change must be synchronized between all parts of the system. But if a smooth transition is not required, it is simpler to reconfigure connection-less NoCs as there is no connection handling.

4.4.4 Multicast

An obvious feature to include in the communication system is multicast support. Multicast can either be a feature of the network or handled by the blocks them selves. The NA can implement multicast by supporting special multicast transactions from the block and then send data to blocks. The NA will have an internal list of multicast destinations, which must be configured before hand, similar to the routing paths. It will add overhead to every NA in the network and increase the bandwidth requirement on link attached to the NA proportional to the number of blocks receiving multicast transfers. Multicast support can be moved to a special multicast unit situated in a central spot in the network. Multicast is then available to anyone who needs it, but adds another unit to the system. If the use of multicast is limited in the system and the number of receiving blocks is just a few, the best solution is not to support explicit multicast and let the blocks duplicate the transactions themselves.

4.4.5 Design choice

BE is chosen due to the fact that it is cheapest, and the traffic of the application is well defined. All signals are deterministic and periodic which means that thorough planning is possible. Under normal operation no unexpected traffic interference will ever occur, which means that guarantees can be given from the use-cases. The current traffic pattern of the back-end system will be used with bandwidth constrained mapping as discussed in Section 4.3, which will ensure that bandwidth is sufficient. Most of the signals have the same level of importance to the overall system performance, which means that prioritizing traffic is not useful. The latency will depend on the traffic interference, but will be low as long as the links are not at full capacity.

The total system latency can be determined by simulation and thereby taken into account by the block design. The speaker output has to be periodic. Under normal

use, this means that no buffering is needed in the system, as the latency is constant. However, if special events, like data log extraction occur, additional traffic is added to the network, which may influence the system latency. As long as the total amount of traffic does not exceed the network capacity in any resource, the system will still operate correctly, but the latency will be different. At the moment additional traffic is introduced in the NoC, the period between the output-samples will be longer. But once the latency has stabilized, the output-samples will arrive periodically again as illustrated in Figure 4.7. To account for varying system latency buffering is needed at the system output and the algorithms need to be able to tolerate limited variation of the system latency.

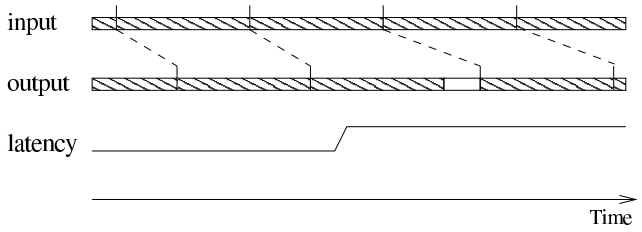


Figure 4.7: Latency variation impact on system output.

Varying latency is also the main reason why handshake protocol interfacing is important. Blocks need to wait for data to arrive.

When the use-case is switched the blocks and the NoC are reconfigured. Block reconfiguration involves resetting the block, so no smooth transition is required. Any connections can be set up during the transition without paying attention to data loss. However, the system uses many signals. If each signal require its own resource reservations, the NoC will become quite complex to handle all of them. Furthermore as discussed later in Section 4.5.1, connection-oriented communication requires some kind of connection-less communication service to set up the connections.

During mode shifts the network and the blocks are reconfigured, which will cause a burst of additional traffic in the network. The system is reconfigured as fast as possible, which means that congestion may occur on some links temporarily during reconfiguration. Thus the traffic will not flow as planned before everything is reconfigured.

Multicast is not supported by the NoC and must be handled by the block itself. Figure 3.2 shows very limited use of multicast and only involves two receivers.

4.5 Routing

Routing dictates the use of the network topology and the communication primitives offered by the network, and thereby implements the network services. Routing is implemented by the routing node and network adapter design and involves important design choices. The following sections will discuss design choices regarding switch-

ing, forwarding strategy, buffering and protocol which are all very important to the network component design.

4.5.1 Switching

Switching determines how data and connections are handled by the network and is a very important choice to both routing node design and the network services. The two switching concepts have been introduced shortly in Section 2.3, but further discussion is necessary to make a design choice.

Switching and network services are entangled design trade offs. Both GS and BE considerations are discussed in this section because the cost of GS is the reason for choosing BE in Section 4.4.

Circuit switching

In circuit switching resources are reserved throughout the network for that particular connection. Circuits are logically independent and can not interfere in any way.

Circuit switching is advantageous in networks providing hard service guarantees. Latency and bandwidth guarantees are inherent features. But the greedy resource reservation has the disadvantage of low resource utilization. Resources are idle when the circuit is idle, even if other connections might be able to use the resources. Circuit switched network has capacity for a certain number of circuits which cannot be exceeded. Circuit switching is strict design approach that does not support BE traffic.

Looking at the back-end system a circuit switched network is a good choice from a service point of view. Planning the traffic in a circuit switched network is straight forward, as it is guaranteed by design that no signals interfere with each other. However, circuit switching has limited network flexibility, especially in cases where the actual bandwidth is sufficient, but the conservative resource reservation prevent further connections to be established. Though circuit bandwidth can be scaled by allocating fewer or more resources, the diverse communication demands in this application will cause a large resource overhead.

Packet switching

Packets are forwarded independently between routing nodes through the network using available resources along its path. Because packets are independent entities, they can be interleaved arbitrarily. This means that network resources are never reserved and idle. Any idle resource can be used at any time as illustrated in Figure 4.8. There is no limitation on the number of connections through the network, just limited network resources which limits the overall performance. Adding more traffic will in general decrease the performance experienced by every individual data stream. Resources are used efficiently and a high utilization is obtainable. But the lack of resource reservation may cause congestion. Congestion has severe impact on packet latency and system performance and should be avoided in real-time sys-

tems, like this application. Congestion can be avoided by thorough characterization of use-cases and network resource planning.

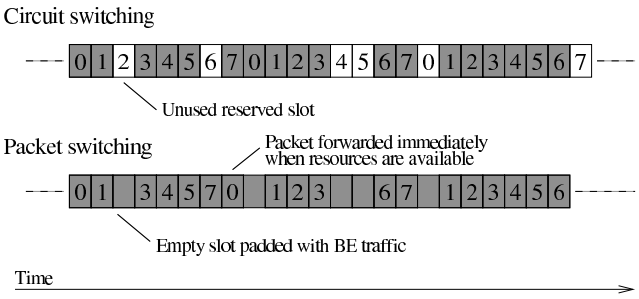


Figure 4.8: Network resource utilization using circuit and packet switching.

Packet switched networks fundamentally treats all traffic as BE. Combining connection oriented packet switching with service guarantees and resource reservation give similar advantages to circuit switched networks. However, sharing idle reserved resources is still possible as done in the router designed in [15] and in the EAthereal NoC [6], where BE traffic can use unused reserved time slots. Combining BE and GS traffic is the key to high resource utilization in GS enabled NoCs. The downside of this approach is that the routing nodes have to handle the two types of traffic. Instead of a having a router supporting either connection-oriented GS or BE, the router has to include both, which will have an impact on the cost of the routing nodes. Though BE support is needed for GS enabled NoCs for configuration anyway or a special configuration bus must be used, which will be discussed in Section 4.6.

Design choice

Packet switching is chosen due to greater flexibility. The streaming behavior of the signals in the system makes a circuit switched solution appealing. But circuit switching is better suited for systems with constant data streams with similar bandwidth requirements. Packet switching flexibility leads to better resource utilization in this application. Keeping the cost as low as possible is a main objective, which requires high network efficiency and is another reason for choosing a packet switched design.

Traffic planning is not as easy in packet switched networks as communication interfere. Non-deterministic traffic causes uncertain latencies because of unpredictable interference. However, this uncertainty can be reduced or removed by GS support. In deterministic systems like back-end system, use-cases are well known and traffic can be planned using the bandwidth requirements as discussed in Section 4.3 even without GS.

The packet switched communication is chosen to be *connection-less* without GS support due to the choice made in Section 4.4.

4.5.2 Forwarding

The forwarding mechanism is as important design choice in packet switched networks. It influences the buffering requirements, but also the protocol when packets a blocked. Three forwarding approaches were introduced in Section 2.3, which will be treated in further detail here to make a design choice.

Store and forward

The store and forward strategy is illustrated in Figure 4.9a. Stalled packets block just the routing node, where it is currently located. The disadvantage is the buffer requirements of the routing nodes. The buffers have to be able to hold the entire packet, which limits the packet size. If the packet size is small, the links can be made wide enough to transfer an entire packet in one step. This is the only way to obtain a one cycle latency using store and forward. If the packet is split into more flits, the first flit will be waiting for the tail flit to arrive before continuing, even if resources ahead is available. This means a large total latency, which is given by:

$$l_{NoC} = n_{flits} \times l_{flit} \times n_{hops} + 2 \times l_{NA}$$

In this application the clock frequency is low, which means latency is quite important and links must be wide enough to transfer the entire packet in parallel.

Wormhole

Wormhole routing approach is illustrated in Figure 4.9b. The advantage is the low buffering requirements. Essentially each node just need buffer capacity to hold a single flit. On the other hand if a packet is stalled, it will stall all nodes it is currently spanning across. This can be avoided by using virtual channels but require redundant buffers in each routing node. Wormhole routing use buffer space efficiently and has low latency even for multi-flit packets.

$$l_{NoC} = n_{flits} + l_{flit} \times n_{hops} + 2 \times l_{NA}$$

Low latency, no packet size limitation and low buffering requirements makes this approach suitable for this application.

Virtual cut-through

Virtual cut-through is illustrated in Figure 4.9c. Low latency is ensured by not waiting for the last flit to arrive before continuing. But the next node has to be able to buffer the entire packet before accepting the first flit. Virtual cut-through requires more buffering than wormhole routing and packet size is limited. The additional buffer space compared to wormhole routing is only used, when packets are blocked. The network latency is equivalent to wormhole routing.

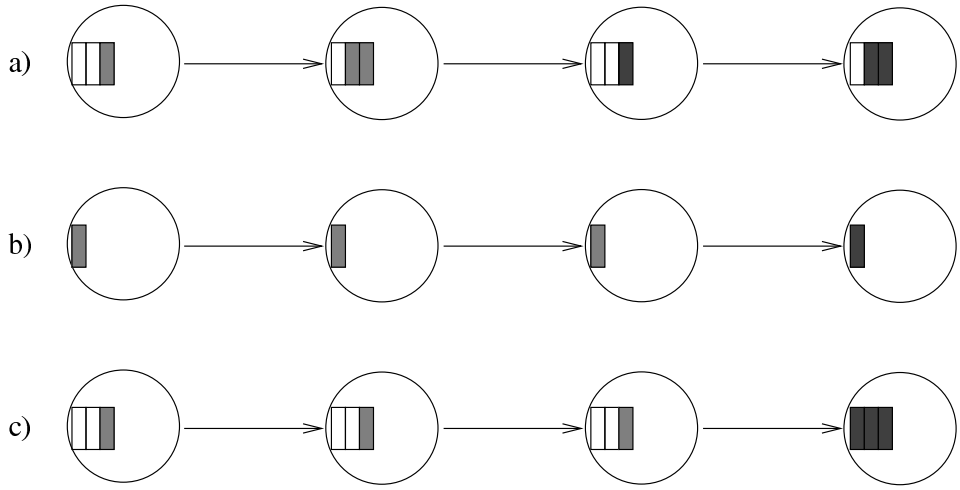


Figure 4.9: Forwarding strategies. a) Store-and-forward. b) Wormhole routing. c) Virtual cut-through.

Design choice

Wormhole routing is chosen due to the fact that it has the lowest buffering requirements and low latency. Wormhole routing does not restrict the packet size. The signals of the back-end system have different bit widths, which makes flexible packet sizes the best choice. The link width is chosen to be wide enough for transferring packets containing a standard audio sample in parallel to reduce the latency.

4.5.3 Buffering

Buffers are a fundamental part of any network router. Each routing node needs buffers to decouple the input ports from the output in order to segment the network. Buffers account for the main router area costs in by far the most NoC architectures presented in NoC research. Minimizing the buffer requirements under given performance requirements is a major concern of NoC research.

The two main considerations involved in choosing buffering strategy is the buffer size and their location within the router. These design choices will be discussed in the following.

Buffer size

The buffer size influences the flow through the network. The minimum buffer size is determined by the forwarding strategy as discussed previously, though buffer size may be increased to act like queues. Large buffers smooth the traffic flow in case of bursts, but can not prevent congestion. If traffic is running smooth the buffer size can be kept low.

In [12] an algorithm which sizes buffers in a mesh based NoC on basis of traffic pattern is presented. It is shown that intelligent buffer allocation results in great savings in area. Long queues are only needed if the traffic contains bursts.

In case of wormhole routing the buffer size also influences blocked packets. Longer queues can reduce the number of nodes blocked by stalled packets, as they allow more flits to assemble at each routing node.

For this application minimum buffering is major concern because of the area costs, but decoupling buffers are needed. Considering decoupling, it is not possible to have buffering capacity of less than two flits per port, while maintaining a throughput of one flit per cycle. This is illustrated in Figure 4.10. Single flit buffers behave like a pipeline, and need every flit throughout the system to move simultaneously as illustrated in Figure 4.10a and 4.10b. A stalled flit may block the path of other flits that may be several hops away. All depending flits must know immediately if they can continue or not, which requires global signaling. To decouple output from input, the input must be able to accept without knowing if the output is blocked or not, which requires a buffer capacity of at least two flits, Figure 4.10d. Otherwise routing node latency will be at least two cycles as shown in 4.10c.

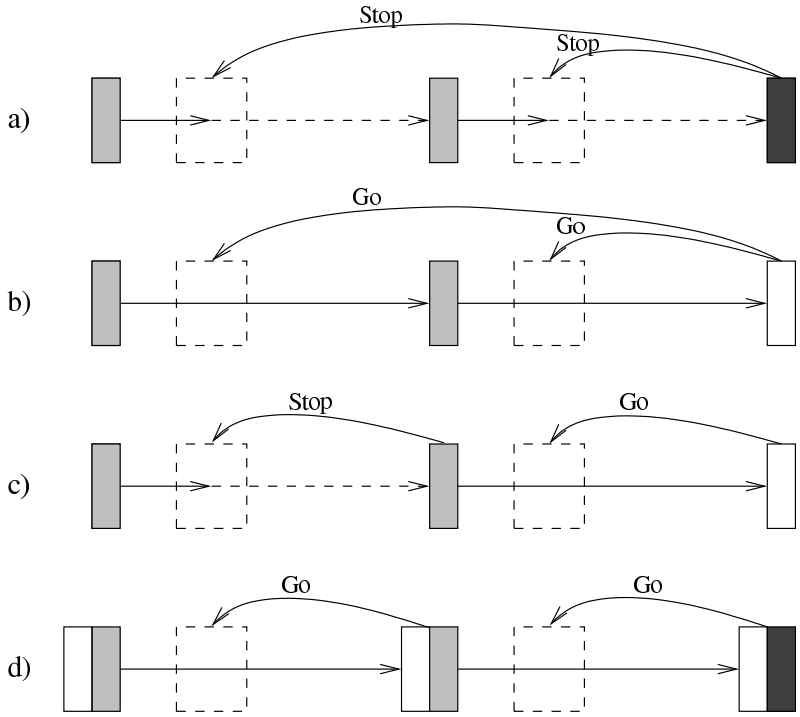


Figure 4.10: Decoupling buffers. a) Global signaling: Blocked flit. b) Global signaling: Free flow. c) Local signaling: Flit is blocked even though forwarding is possible. The flit must wait for the next buffer to be empty, which increases the latency. d) Local signalling using two-flit buffer: Full throughput.

Buffer location

The buffer location and the number of buffers are both important design choices in relation to routing node performance and cost.

The buffers can be situated at the input or the output of the router. The buffering location mainly influences how packets block each other. In long input queues the flit at the head of the queue may block flits destined to other ports, causing unnecessary performance reduction. However, if queues are short, the problem is less significant and not present for single-flit buffers. Blocking can be reduced using virtual channels, introduced shortly in this section and treated in more detail in Section 4.5.4.

Buffers may be distributed and dedicated to a port or shared in pools by more ports. One approach is to have one dedicated queue for each input or output. Buffering capacity can be utilized more efficiently by more advanced buffering strategies, like managing buffers in centralized buffer pools. However, these are found too expensive in area due to the overhead of controlling logic, as stated in [1].

Distributed buffering may be combined with the virtual channel (VC) concept. VC is basically sharing physical links by several logically separate channels with individual and independent buffer queues, Figure 4.11. VCs allow packets to overtake stalled packets among other performance advantages that will be discussed in Section 4.5.4. In relation to buffers, the independent buffers of VCs introduce more buffers in the router design, but may allow shorter queues.

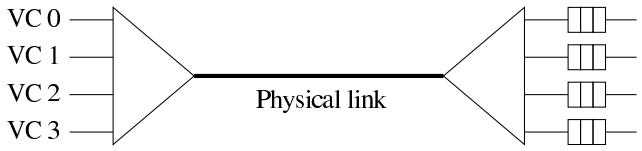


Figure 4.11: Virtual Channels.

Design choice

For this application cost is the main factor in the design trade offs. Wormhole routing was chosen earlier due to low buffering requirements. It is chosen to combine wormhole routing with *virtual channels* to minimizes the impact of stalled packets in the network and obtain better buffer utilization.

Input buffering is chosen to minimize the link feedback signals required for an efficient VC based design. Output buffering would require buffer status of all port to be available to neighboring nodes to make proper protocol decisions.

The buffer size is chosen to be minimal, while maintaining full routing node throughput and a latency of one cycle. As stated earlier a buffer capacity of two flits is the minimum for full throughput. Instead of having a two flit queue, two independent buffers can be used in an alternating manner. Thus two VCs are offered for only the overhead of a switching unit with two inputs per port. Each VC can not utilize

more than half the link bandwidth, which prevent data intensive communication from starving others packets using the same link. The router has to alternate between each VCs for full link utilization as illustrated in Figure 4.12.

This design uses *two VCs* to offer full link utilization and full decoupling (i.e. a one cycle latency) with just *two single-flit buffers* at each input port. The link utilization depend on the chosen VC handling, which is determined by the routing protocol. The buffer size is minimal, but still efficient because of the VC principle.

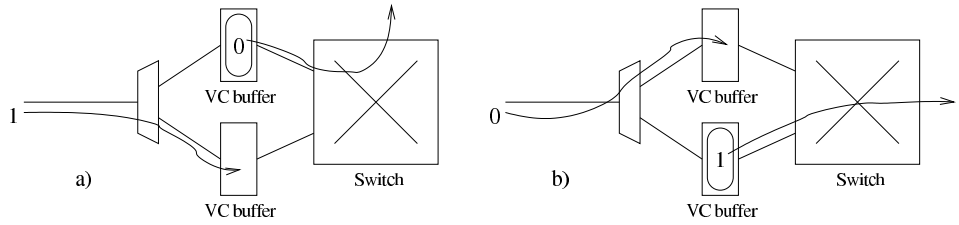


Figure 4.12: VC input buffers. Full link utilization using one-flit buffers by alternating between the VCs.

4.5.4 Protocol

A network is a medium shared by many users. The protocol determines how to share the resources in a fair way and avoid any conflicts. The protocol design includes important choices on path selection, deadlock handling and resource allocation. A NoC is a very restricted network that will under normal circumstances never be expanded or reduced in size, capacity or number of blocks attached. The controlled environment thus it can be assumed that data loss and corruption will never occur.

Path selection

The most fundamental task of the routing protocol is to determine the path from source to destination, which data is to be forwarded along. Network topologies with multiple paths between two locations make choosing the path a part of the protocol. The aspects affecting the path selection have been introduced in Section 2.3, but will be discussed in the following in relation to this application.

Routing can be either deterministic or adaptive. Deterministic routing has the advantage of being simpler than adaptive routing. Adaptive routing involves dynamic arbitration schemes based on current conditions in the network. An alternative path may be chosen if one path is currently congested or based on some other condition. This makes the path selection logic more complicated. The routing nodes have to be directly involved in the path selection.

Comparing deterministic and adaptive routing it is obvious that adaptive routing comes at the price of more complex routing nodes. In systems with deterministic traffic conditions are predictable, and adaptive routing is unnecessary overhead.

The path can be chosen to be minimal to minimize the overall network load and power dissipation. But an alternative path may be a better choice to avoid congestion in overloaded resources. In this application minimal routing is the best choice from a performance point of view. But allowing different paths too, will increase the system flexibility as network resources can be utilized better.

The actual path may be determined by source routing or distributed routing. In distributed routing the path is chosen by the routing nodes. Each routing node decides by itself in which direction to forward the packet.

In source routing the path is determined by the source network adapter. The path through the network is included as the routing information in the packet header. The path is read by the routing nodes to determine where to forward the packet. Relative path descriptions allow simpler routing nodes, as they do not have to know their location in the network. Figure 4.13 shows the principle of source routing using directional instructions. If the paths given by the source NA can be altered, the network is not stuck with one routing scheme. Paths may be changed arbitrarily under different use-cases.

The path can be selected according to different schemes. XY-routing is a popular deterministic routing schemes used in mesh based NoCs. XY-routing is characterized by strict minimal dimension-wise routing. Furthermore XY-routing has an advantage regarding deadlocks.

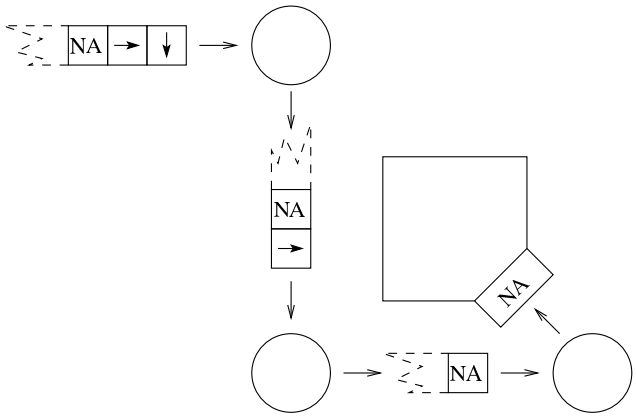


Figure 4.13: Source routing.

Avoiding Deadlocks

The previously chosen mesh based topologies have the possibility of circular dependencies and are therefore prone to deadlocks. Deadlock avoidance is therefore a very important part of the protocol design.

Deadlocks can be avoided by design or solved when they occur. Deadlock avoidance is a conservative approach restricting the routing in a way that the system will never reach a deadlock state. Solving deadlocks when they occur does not impose

restrictions on the routing scheme, but needs to discard data or use a roll back mechanism.

Avoiding deadlocks by design is by far the cheapest approach, as there is no need for special deadlock handling. In [14] it is proved that XY-routing in bidirectional mesh topologies is deadlock-free under the assumption that packets are always removed from the network at its destination.

The proof of absence of deadlocks in a network in [14] is summarized here for the purpose of proving that XY-routing in optimized mesh-like networks is deadlock-free. When a mesh network is optimized by removing unused links, as discussed in Section 4.2, strict XY-routing is no longer be possible without modifications to account for removed links.

The proof constructs a channel dependency graph to demonstrate that no circular channel dependencies exist. Figure 4.14a shows the dependency graph of a regular bidirectional mesh network using XY-routing, while Figure 4.14b shows that adding an extra node creating an irregular topology, does not introduce cyclic dependencies. The dashed line indicates routing that does not comply with the XY-scheme, caused by the missing links.

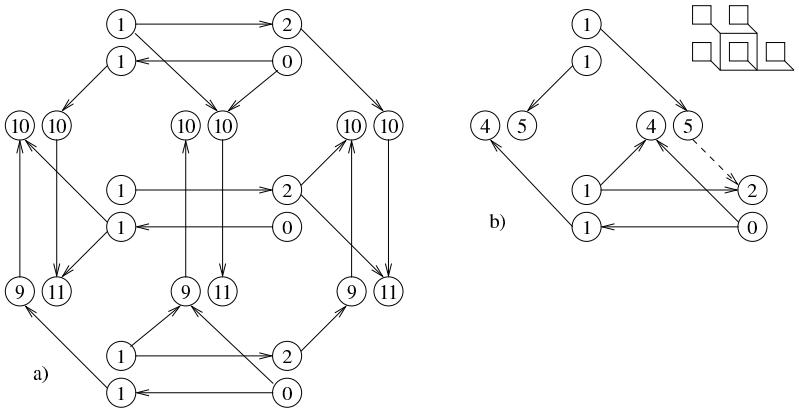


Figure 4.14: Channel dependency graphs which show absence of cyclic dependencies when using XY-routing. a) Dependency graph for a regular 3 by 3 bidirectional mesh without VCs. b) Dependency graph for irregular mesh-like network.

Resource allocation

Resource allocation is an important design topic in relation to NoC performance. The resources of the network are the links, buffers and VCs.

Order preservation is a very valuable network property for the system designer. The resource allocation therefore has to ensure that packets between two blocks preserves ordering, while they can be interleaved arbitrarily with other packets. Deterministic routing is a prerequisite for network guaranteed order preservation.

The use of VCs and wormhole routing means that flits may overtake each other. However, they are a part of a packet that must be logically separated from other packets. Flit handling is how packets are treated as entities, while packet ordering is how to ensure data ordering. These are both important choices along with input port arbitration.

Flit handling

One approach is to forward each individual flit using any available VC without regard to packets. This means that every flit has to contain routing information as the routing node has no way to distinguish flits from different packets. The header is a large overhead, which makes this choice infeasible for this application.

Another approach is keep flits of a packet logical separated from other flits by VC reservation. When a multi-flit packet arrives at a routing node, the node reserves a VC on the output link for that particular packet. The VC is released when the last flit of the packet has been forwarded. Thus the flits of each packet are handled as a logical entity. As long as each VC is reserved for each packet while it is passed through, no one else can use that VC. However, the physical link is still available to the other VC, if the packet is blocked. The advantage of this approach is that the routing node knows that packets are send separately using only one VC. When the destination of the packet has been determined from the packet header in the first flit, all the following flits can be forwarded in the same direction until the last flit marks the end of the packet and releases the VC reservation. Thus no routing information is need for the following flits.

Packet ordering

One approach is to assign specific directions to a specific VC. The routing node determines the VC by the direction of the packet in the next node. Packets destined for a particular output port on the following node is always assigned to one VC. Thereby packets heading for the same destination can not overtake each other preserving ordering. This approach is very inflexible, especially in this case where the number of VCs is less than the number of output ports. More outputs must be assigned to a single VC, which means that packets may block each other for no reason.

Another approach is to forward packets using any available VC and preserve ordering by recording the order of arrival. It may happen that two packets with the same destination use both VCs. The first to arrive must be forwarded first to preserve ordering. This solution is less restrictive and more efficient with a low number of VCs. The downside is that in a worst case scenario packets for the same destination may end up blocking all VCs on an input link. However, this situation is unlikely to occur.

Port arbitration

One approach is to use a dynamic port arbitration scheme, to ensure a fairer scheduling of the packets arriving at the input ports. Round-robin is an example of scheduling scheme providing weak fairness. However, such schemes become complex when multi-flit packets come into consideration.

Another option is static scheduling. The input ports are handled by a prioritized list. If the first port has flit waiting it is forwarded otherwise the next port is handled and so on. This approach does not guarantee any fairness, but copes well with VC reservation. Input with ongoing VC reservations can easily be assigned higher priority.

Design choice

The size of the NoC and the number of blocks attached to it, make *deterministic routing* a better choice than adaptive routing schemes. Traffic conditions are predictable and eliminate the need of adaptive routing.

Source routing is chosen due to simpler routing node design. The network adapters hold an address table containing paths to which it needs to communicate with. Obviously these addresses must be inserted into the table before any communication can take place. Redirecting a signal is done by changing the corresponding path in the address table of the source network adapter.

The chosen topologies are mesh-based, which makes *XY-routing* the best choice due to the deadlock prevention. In the optimized network topology, strict XY-routing may not be possible. In these cases routing is done as close to XY-routing as possible with slight modifications to take removed nodes and links into account. These modifications makes source routing an excellent choice as the paths are explicitly programmed into the NAs. The routing nodes do not need to know anything about neighboring nodes or attached blocks. Packets are forwarded solely on basis of the path included in the packet header. This means that non-minimal routes are supported too, for better system flexibility.

Flit handling is chosen to be handled by *VC reservation* as it is most efficient. Packet ordering is done by *input ordering*. Deterministic traffic, bandwidth constrained mapping and alternating use of VCs makes it is unlikely that two packets will end up blocking both VCs an a link. *Static input arbitration* is chosen.

4.6 Programming model

The programming model design specifies how NoC is configured, which is important in relation to the system flexibility. Connection-less packet switching using relative source routing have been chosen earlier, which means that the routing nodes do not need any configuration. Only the NA configuration is necessary. The design choices to be considered are who is responsible for configuring the NAs and how it is done.

4.6.1 Distributed programming

One approach is to let each block configure the NA it is attached to. This requires network-aware blocks and the blocks has to be able to access the network configuration. The advantage is that any block is in control of its own communication. However, all blocks have to network-aware and hold its the configuration locally or have access to some other communication structure to get it elsewhere.

4.6.2 Centralized programming

Another approach is to let one network-aware unit configure all NAs in the entire network. This unit knows the network topology and is able to send configurations to all NAs in the NoC. The advantages are that the blocks can be network-unaware and only one unit needs to have access to initial network configuration, which typically stored in externally.

4.6.3 Programming method

Programming can either be done using the network itself or a dedicated configuration bus. Having separate communication structure just for configuration seems unnecessary. But the network is unable to transfer any data before it is configured initially, this is the only option. However, the routing nodes in this design are not configurable and fully operational, which makes programming using the network itself possible.

Configuring the NAs using special configuration packets means that any attached to the network is able to change the configuration of any NA.

4.6.4 Design choices

The back-end system already contains a unit responsible for configuring the blocks. It is obvious to use a similar unit attached to the network to both configure the NAs and the blocks.

The blocks in the current back-end system design is configured using a special configuration bus, which is responsible for loading configuration parameters into the system from external non-volatile memory. This bus is sought replaced by the general network, but a configuration unit is still needed.

It is therefore chosen do initial NA configuration *centralized* by a special unit attached directly to the NoC which has access to the configuration in external memory. The configuration of the NAs is done by *configuration packets* containing the routing information to be written into the NA address table. The packets are tagged using a configuration flag in the packet header. Anybody able to insert configuration packets can program any NA. This is a valuable feature if a future back-end system will include a general purpose processor. It will then be able to take control and redirect data streams if it is needed.

4.7 Interfacing

The network interface design involves important design choices, as it defines the abstraction of the network to the blocks attached. Possible abstraction models were introduced shortly in Section 2.5.

4.7.1 Abstraction

One approach is memory mapping, which is very suitable fro systems that are already operating with memory abstraction like general purpose processors do. The blocks are network unaware and the NA appears as a normal memory interface to the attached block and network communication is hidden as read and write operation.

Another approach is message passing, which is network aware. The blocks communicate by sending messages to each other identified by logical identifiers. The NAs will then translate the message into network traffic.

Both approaches mentioned above are primarily targeted SoCs using processors. There is no global memory abstraction in the current back-end system but data streams. Instead of introducing a higher level of abstraction than necessary, data streams are handled like virtual wires. The signals in the back-end system does not use explicit read or acknowledges write operations, data is pushed on the next block without any feedback. There is no need for a higher level of abstraction as long as the system is constructed from dedicated blocks. Furthermore the virtual wire model means simpler NA design by avoiding blocking commands.

4.7.2 Interface

The blocks in the current system does not use any kind of common interfacing, which means that some blocks uses many signals of various bit width while others use just a few. In order to design a generic network adapter, it is assumed that these signals are multiplexed into a single interface.

Similarly it is assumed that the block is able to demultiplex incoming streams. Demultiplexing multiple signals from the same source can be done by counting, as deterministic routing using certain VC arbitration ensures that data ordering is always preserved. Blocks receiving signal from different sources require indication of the source of the data it is currently receiving. This can be done by including a source and/or stream ID in the header of the packet or explicitly as a part of the packet payload. It has to be unique, which requires the ID to be agreed between sender and receiver. The agreed ID has to be reconfigurable in both ends, to allow redirecting signals to other blocks. But the choice is left for the block design.

The interfacing protocol used in the existing design is a simple data valid protocol that pushes the data on without any feedback. This protocol is not sufficient in a NoC based system, as it can no longer be assumed that the receiver, which is now the network, is ready to accept the data. A request/acknowledge protocol has to be employed instead.

4.7.3 Design choices

The chosen interface design is simple data in- and output using *request/acknowledge*. It is assumed that signal are multiplexed and demultiplexed by the block itself. Signal multiplexing and dealing with the request/acknowledge protocol requires insight to the actual implementation of the blocks, which is not investigated further here. Replacing a point-to-point communication system in a very tightly coupled system with a NoC, will require some modifications of the blocks related to interfacing.

The block configuration bus is replaced by ordinary network traffic. The block itself must separate from the incoming data. Block configuration might be tagged by the configuration unit. This is left for the block design.

The block interface is illustrated in Figure 4.15 and consists of a data input, a destination selection input, which used by the NA to select the right destination, and the request/acknowledge signals.

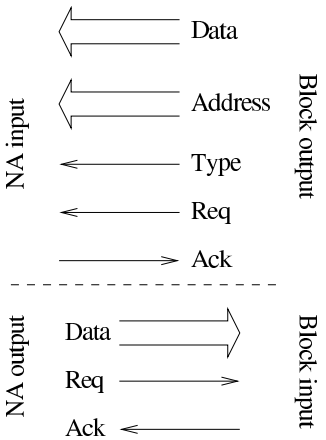


Figure 4.15: The NA/block interface.

4.8 The networks

This chapter has discussed the design trade offs involved in designing a NoC to replace the point-to-point connections and the configuration bus of the back-end system used in digital hearing aids by *Widex A/S*. This section will summarize the design choices and apply the design methodologies to form the network adapter, the routing node and the network designs.

Area and power dissipation have been the main focus throughout the design discussion. The major challenge in designing NoCs for very small systems is to determine essential features of the NoC that are required by the application. Two NoCs has been chosen for implementation using the same NoC components.

1. A NoC based on a regular mesh topology.

2. A NoC based on optimized mesh-like topology.

Both NoCs follow the same design specification and differs only by topology and application mapping. Source routing and topology unaware NAs and routing nodes mean that the NoC components can be used to construct both topologies. The design choices are summarized below.

- *Connection-less packet switching* implementing *BE* communication ensures simple but flexible communication.
- *Wormhole routing* means minimal buffering and no packet size limitations.
- *Source routing* means programmable paths and topology unaware routing nodes. Deterministic *XY-routing* ensures data order preservation and prevents deadlocks.
- Two *virtual channels* for efficient buffer utilization and better traffic flow.
- *Bandwidth constrained mapping* and known use-cases ensures that latencies are predictable.

4.8.1 The routing node

The routing node is a generic 5-port router illustrated in Figure 4.16. All ports are equal and have one physical link for incoming data and one for outgoing data, both having two VCs. The port names in Figure 4.16 are insignificant. Any port can be connected to either another routing node or a NA.

Figure 4.17 illustrates the routing node design. The VC buffers are located at each port. The VC buffers are available to the output ports through the switch. The switch is not fully connected. Packets can not turn around and return the way they came. The switch is controlled by routing logic at each output port, which determines which packets to be forwarded. If the packets do not compete for the same output port, full throughput on all inputs and outputs is possible simultaneously. If two ports try to access the same port, one of them is chosen by the static arbitration scheme. Starving one source is theoretically possible, but is not a problem as long as the links of the network is not fully loaded.

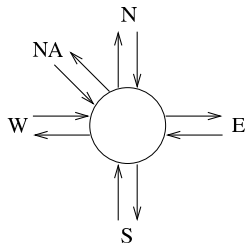


Figure 4.16: 5 port routing node

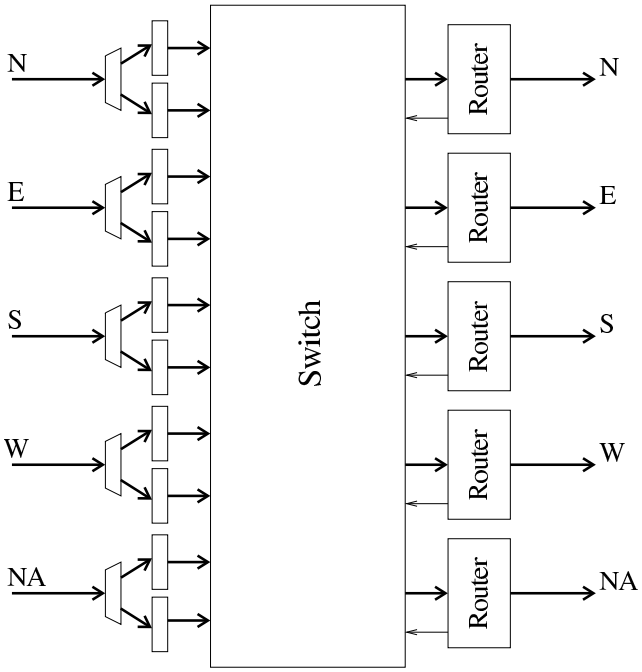


Figure 4.17: Routing node design overview.

The links between the routing nodes transfer one flit at the time in parallel in each direction. Data serialization is expensive and requires NoC transfer rates significantly faster than block throughput.

The back-end system operates at a low frequency, which means pipelining the implementation is not necessary. Buffers are expensive in terms of area, should only be used for decoupling purposes in relation to routing and forwarding.

Source routing means that the path in the packet header is represented by a series of direction instructions, one for each routing node. Each node read the first instruction and forwards the packet in the specified direction after the path has been rotated one instruction in the packet header.

4.8.2 The network adapter

The NAs are designed to use only one VC at the time, as handling both VCs simultaneously dramatically increases the complexity of the NA. Furthermore restricting all NAs to use only one VC means they can only use half the bandwidth of any network link. Thus the flit interleaving on the links prevent data intensive streams from starving other streams.

The network adapter has a block interface and a network interface matching the ports of the routing node. NAs are connected to the network using the same type of links as between the routing nodes and behaving similarly, which mean that NAs can be attached to any port on the routing node. This is exploited in the optimized

topology. The NA buffers block data inputs and wraps into packets. Buffering the data will release the block earlier in case of large packets, which are split into more flits. Similarly flits received by the NA have to be buffered before the payload can be assembled and delivered to the block. Just as the NAs are designed to use only one VC when sending packets, they cannot receive packets simultaneously on both VCs.

The NoC latency is determined by several factors. The main contributor is the logical distance between source and destination. Ideally one hop is one clock cycle, however interference with other traffic may increase the latency. Furthermore flit serialization and alternating use of VCs increases the latency for each additional flit, which leads to this general expression of the NoC latency.

$$l_{packet} = 2 \times l_{NA} + (n_{flits} - 1) \times l_{flits} + n_{Hops} \times l_{Node} + l_{interference}$$

Inserting know latencies into the expression gives the following latency estimate.

$$l_{packet} = 2 + 2 \times (n_{flits} - 1) + n_{Hops} + l_{interference}$$

Source routing means that the NA is not topology dependent. The NA just has to know the path to be included in the packet header. This path is hold in an address table indexed by the address signal from the block. The address table is initially empty and has to be programmed at system initialization using programming packets. The contents of the received programming packets are written directly into the address table. Figure 4.18 illustrates the design of the network adapter.

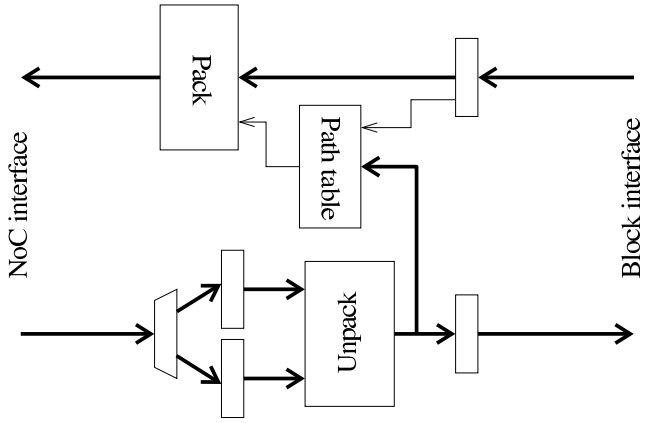


Figure 4.18: Network Adapter design overview.

4.8.3 The mesh

The mesh based NoC is constructed using the previously designed NoC components. As discussed in Section 4.3 the traffic of the four main blocks of the system is split across two NAs. 16 NAs are therefore needed to accommodate the entire system. A

4 by 4-node mesh is designed for the purpose. The programming unit is connected to an unused link at the edge of the network. The application graph is mapped to the topology by placing bandwidth intensive blocks close as discussed in Section 4.3. Figure 4.19 shows the system design. Two signals have been assigned to dedicated point-to-point links to reduce the bandwidth requirements. These are not shown in the figure.

The bisection bandwidth of the 4 by 4-node mesh equals the total bandwidth available to the cores. Congestion is therefore local problem, it is solved by the mapping.

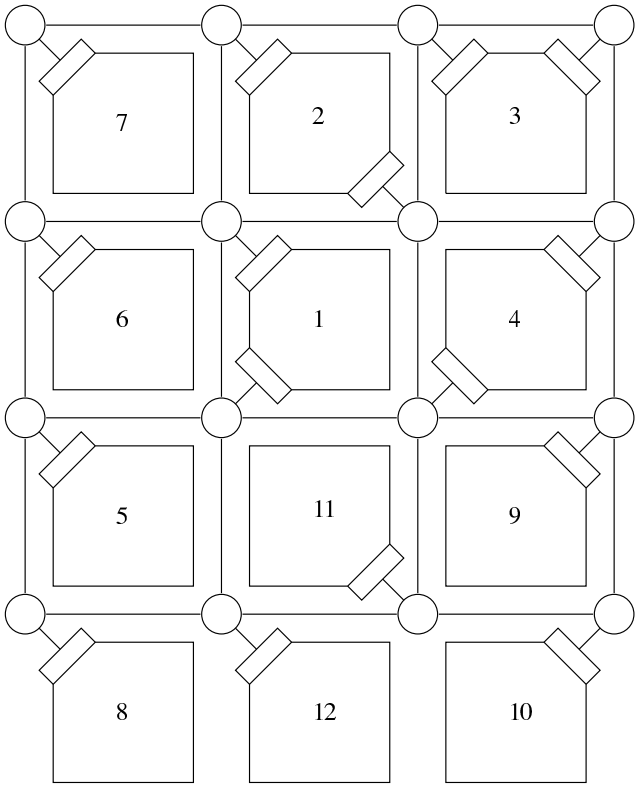


Figure 4.19: The mesh based NoC with mapping.

4.8.4 The optimized mesh

The optimized mesh network has to be sized to accommodate 16 NAs and leave one additional link for the programming unit. The network is aimed to be square shaped to minimize the maximum number of hops between two blocks.

At least 7 routing nodes are required to accommodate the 16 NAs in a mesh-like topology. One routing node is connected by only one link. The mapping is done according to the method discussed in Section 4.3. The network layout and mapping

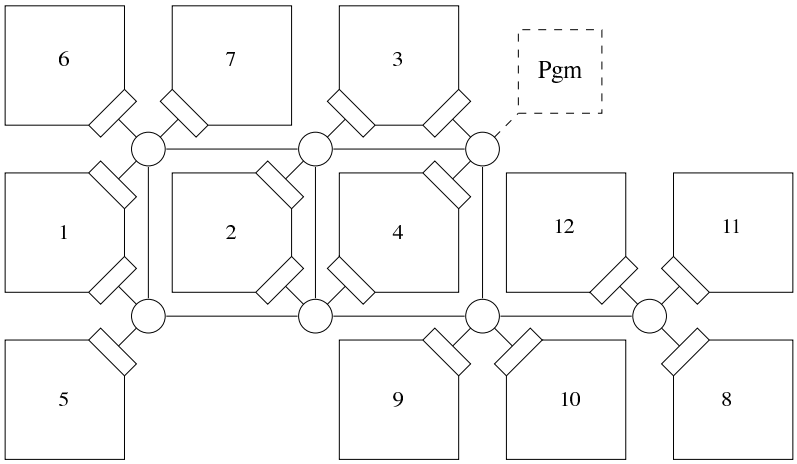


Figure 4.20: The optimized NoC with mapping.

is illustrated in Figure 4.20. The XY-routing has to be modified slightly to account for the missing node.

Reducing the number of nodes brings the blocks closer in terms of network distance. This means that the maximum number of hops between two blocks is reduced. Bits can be saved in the packet header, when the path is shorter.

The optimized mesh topology brings the blocks closer, which means further decrease in power consumption than what is saved by having fewer routing nodes and links.

Chapter 5

Implementation

This chapter presents the NoC implementations and how results have been obtained. An overview of the design tool flow is presented and power estimation is discussed.

A NoC is not a system by itself, it is the interconnection structure that binds the system together. The essential parts of a NoC, which is the routing nodes and the NAs have been implemented at RTL, while other parts have been modeled by behavioral simulation components. The NoCs have not been integrated into the current back-end system design, as it would require modifying the block interfaces as discussed in the design. The purpose of the implementation is to find a realistic estimate of the NoC costs for future system designs.

The blocks are modeled by traffic generators, emulating the traffic pattern of the current back-end system. The programming unit responsible for configuring both block and network is modeled by behavioral code, as it is the communicating with off-chip memory. The implementation is done in VHDL, both for behavioral and RTL implementations. Tools for code generation and traffic generation have been developed in C and Perl. The VHDL source code is included in Appendix C.

The following sections will give an overview of the implementation of the network adapter and the routing node, used to construct the two NoCs implemented in this project.

5.1 Routing node

The routing node has five bidirectional ports that are completely independent and can operate simultaneously at full bandwidth. Distributing the control logic among the ports are therefore the most simple and efficient implementation. The routing node implementation is split into sub-entities. The VC buffers are located at the input ports and are implemented as buffers with room for a single flit. The header decoding is handled by a separate unit for each input buffer. The actual routing logic is implemented as an entity placed at the output ports. The routing node implementation is illustrated in Figure 5.3.

Before looking into the internals of the routing node, the NoC links and the flits are described.

5.1.1 NoC links and flits

Wormhole routing means that packets can span multiple flits. The routing information is only included in the first flit as illustrated in Figure 5.1, while the following flits only contains data. When the first flit has been forwarded, the routing node has to remember the output port and VC for the following flits. The path included in the header is a string of instruction to the routing nodes. Each instruction dictates the forwarding direction. The string is rotated when the flit is forwarded, thus the first instruction of the string is always the next to be executed.

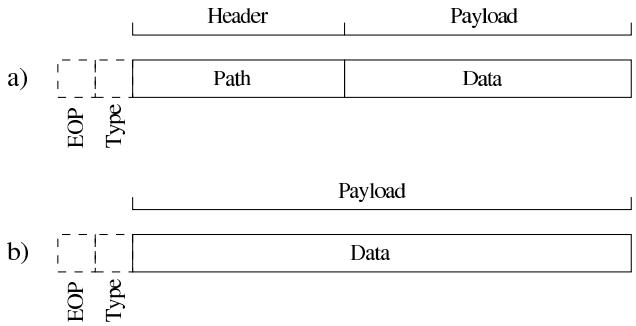


Figure 5.1: Flits. a) The first flit of a packet. b) Additional flits if the packets span more flits.

The NoC links consists of a signal wide enough for transferring one flit in parallel along with the packet type flag and signaling bits. The signaling bits are not a part of the flit, but indicates if data is valid, the active VC and if the current flit is the last flit of the packet. The NoC link has a signal in reverse direction to indicate whether buffer space is available or not. Figure 5.2 illustrates the NoC link.

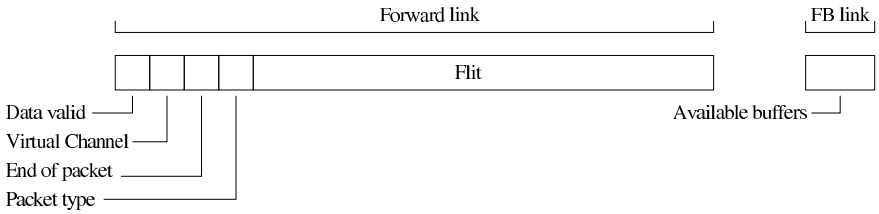


Figure 5.2: NoC link.

5.1.2 Flit handler

The flit handlers is located just after the input buffers and handles header decoding. The buffer status is feed back to the neighboring node, for use by its router. The path

is decoded and the direction is stored in case of a multi-flit packet. The presence of a flit signaled to all relevant routers. When a new flit arrives to either VC buffer, a signal is toggled to indicate the newest flit in order to preserve packet ordering.

5.1.3 Router

The routers implement that actual routing mechanism. Several flits may be destined for the same output port, which requires input arbitration. The arbitration is static prioritizing among the input VC buffers. As long as the network is not fully loaded, the arbitration scheme has very little or no impact on the NoC performance. Flits for reserved VCs are given higher priority to finish packets as soon as possible and release the VC. A VC is available for new packet transmission if it is not reserved and the buffer in the receiving node is empty. The VC is released as soon as the last flit has been forwarded.

The routers can not forward packets from their own input, but select flits from any other input buffer using an eight input multiplexer.

The implemented logic is simplified by the fact that the buffers are able to hold one flit only, but is otherwise highly configurable. Adding deeper buffers will require modifications to the input arbitration. However the cost of adding deeper buffers has a much greater cost impact than a slightly more complicated arbitration scheme.

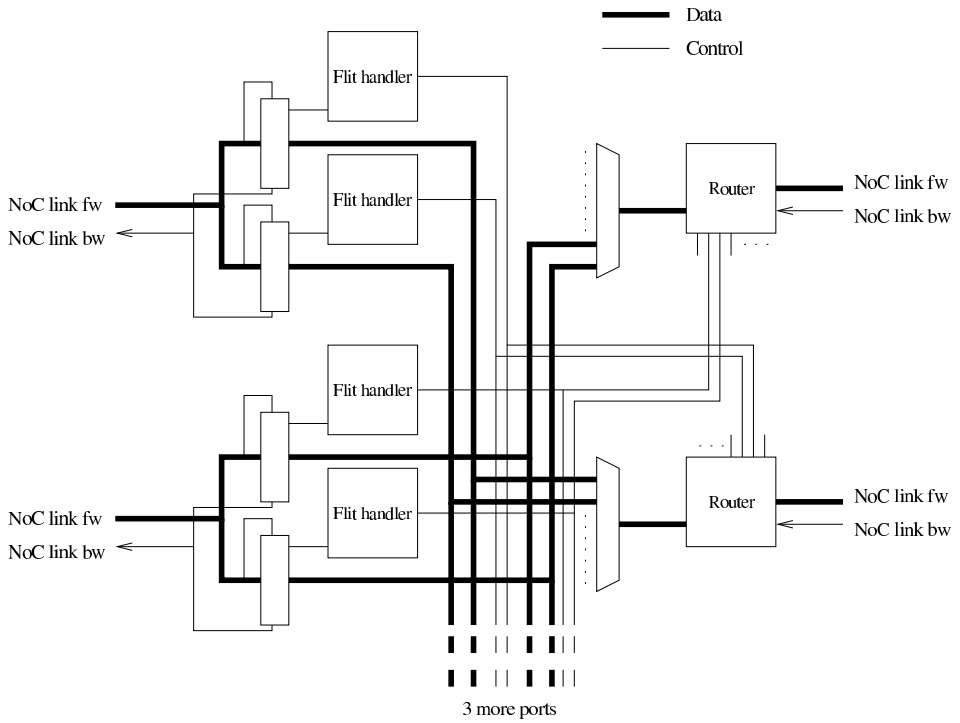


Figure 5.3: Routing node implementation showing only two ports.

5.2 Network adapter

The network adapter has a NoC and a block interface, which are both bidirectional. The NoC interface communicates directly with any port on a routing node.

The block input and output ports are separate signals. The input port has two additional signals for indicating address and packet type. The address selects the data destination and the packet type indicates whether the packet is ordinary data or a NA configuration packet. This allows any block connected to the NoC to configure NAs.

The NoC interface has an input and an output port just as the routing node shown in Figure 5.2. End of packet indicates that the flit is the last flit of that packet, which clears any VC reservation in the routing node. The type flag is used only by the NA for configuration packets. These are used to configure the NA and will never reach the output port on the block interface.

The NA implementation is separated into sub entities similar to the blocks shown in the Figure 4.18.

5.2.1 Unpack unit

The two VC buffers are located at the NoC input, followed a single unpacking unit. Header data is discarded and data is reassembled in an internal buffer. It is then presented on the block output or written to the address table if the received packet is a configuration packet. Packets may arrive at both VCs, however the NA implementation handles one packet at the time, blocking any packets arriving on the other VC. Worst case, the NA is capable of receiving flits at the rate of a single VC.

5.2.2 Pack unit

The packing unit converts data input to flits. As mentioned earlier, multi-flit packets are forwarded using reserved VCs. Packets are inserted at half the rate of the NoC link, unless two packets are generated simultaneously and forwarded interleaved by the NA. The implementation supports one packet to be sent at the time. The data block input is buffered before it is wrapped into flits. The header of first flit contains the path selected by the address input in the address table. The following flits contain just data. The NA can be configured to any data width and flit size at instantiation. However the flit size must be large enough to hold the header.

The address table of the network adapter is a register file which is quite expensive in area, thus keeping the number of entries as low as possible is important. Most blocks only communicate with a few other blocks, but a general purpose processor in a future back-end system might need to communicate with all other block without reconfiguring the NA. The number of entries is therefore implemented as parameter specified by instantiation.

5.3 Network configuration

Source routing means that the address tables of the NAs must be configured before any communication is possible. For simulation purposes a behavioral programming unit has been implemented along with a tool for generating configuration packets to initialize the NoC. The XY-routing is implemented by the NoC configuration. The routing scheme can be changed by using other paths in the NA configurations. This flexibility makes optimized topologies possible, as missing links and routing nodes can be taken into account in the configuration. However, deadlock-free routing must be reconsidered, as discussed in Section 4.5.4, if the routing scheme is changed.

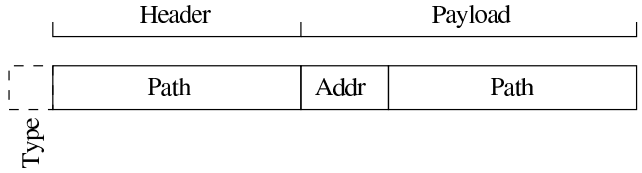


Figure 5.4: NA configuration packet. The payload contains the address to be the configured and the path to be inserted.

5.4 Traffic generators

The blocks connected to the NoC are modeled by behavioral traffic generators. The traffic generator uses a file for input, which lists data to be input along with a time reference indicating when to send the data. Data received by the traffic generator is logged to an output file. Tools have been developed in C and Perl to generate packets and analyze logged packets.

5.5 Synthesis

To estimate the costs of the NoCs, the RTL code must be translated into a physical implementation. The area and power consumption depends strongly the target platform for implementation. Different technologies have different characteristics and leads to results that are not directly comparable. In order to produce results that can be compared to the existing back-end system implementation, a similar design flow must be used.

5.5.1 Design flow

Widex A/S provided access to their design tools, flow and technical assistance for this project, to produce as realistic results as possible. The design flow used is the flow used for the current back-end system implementation. Thus the results are expected to give a realistic idea of the costs of the NoCs. The flow will be described briefly to provide an overview.

The target technology used by *Widex A/S* is low-power 0.13μ. The flow is in general very focused on low power design and illustrated in Figure 5.5. It consists of various tools, scripts and configurations.

The first step towards a physical implementation of the RTL code is logic synthesis. This is done using Synopsys which maps the RTL description into a netlist of primitives available in the design library. The output is a verilog netlist and timing information in SDF (Standard Delay Format). Synopsys is also able to give pre-layout area estimate, based on the netlist.

The verilog netlist is used to verify the design at gate-level with timing information. The verilog model inserted into a VHDL test bench for stimuli and simulated using Cadence NCsim. A module is incorporated in the verilog model to record switching activity at all internal signals in the netlist in SAIF (Switching Activity Interchange Format).

The switching activity information is fed back into Synopsys along with the original verilog netlist to optimize the netlist with regard to power consumption. This means that the netlist is altered to reduce the power as much as possible with the switching activity causes by the test bench. The power optimized netlist is then used by Synopsys to estimate the area and power consumption of the design after new switching activity has been recorded using gate-level simulation.

The next step towards an ASIC implementation is physical planning of the primitives in the netlist. This is done at *Widex A/S* using Synopsys Physical Compiler. As the NoC is only the communication structure of a whole system, this step has not been performed.

5.5.2 Power estimation

The EDA tools are able to estimate power consumption using the gate netlist before physical mapping and wire routing. The exact wire lengths are not known and approximated loads are used to model the loads of the wires. This model is fairly accurate. The flow used at *Widex A/S* usually predicts the power consumption with accuracy around of 10% of the actual chip, from the netlist alone.

A NoC is not a separate part of a system, which makes accurate power estimation more complicated. Synthesizing the NoC alone will make the flow treat as a complete system, placing all logic close together. If the NoC is embedded into the back-end system, the logic is more likely to be distributed among the blocks of the system, i.e. NAs will be placed within the blocks and the routing nodes close by. The links will be long wires, which are not accounted fore in the pre-routing power estimation. Furthermore additional buffers and stronger drivers may be introduced during physical planning, which will affect the power consumption too. However physical planning of the NoC is not possible without the rest of the system.

The only option is to rely on the pre-routing power estimation. The estimate is based on the actual switching activity in the netlist produced by the synthesis tool, extracted from the gate-level simulation. However the estimated will be lower than what can be expected for a real NoC implementation. On the other hand the power

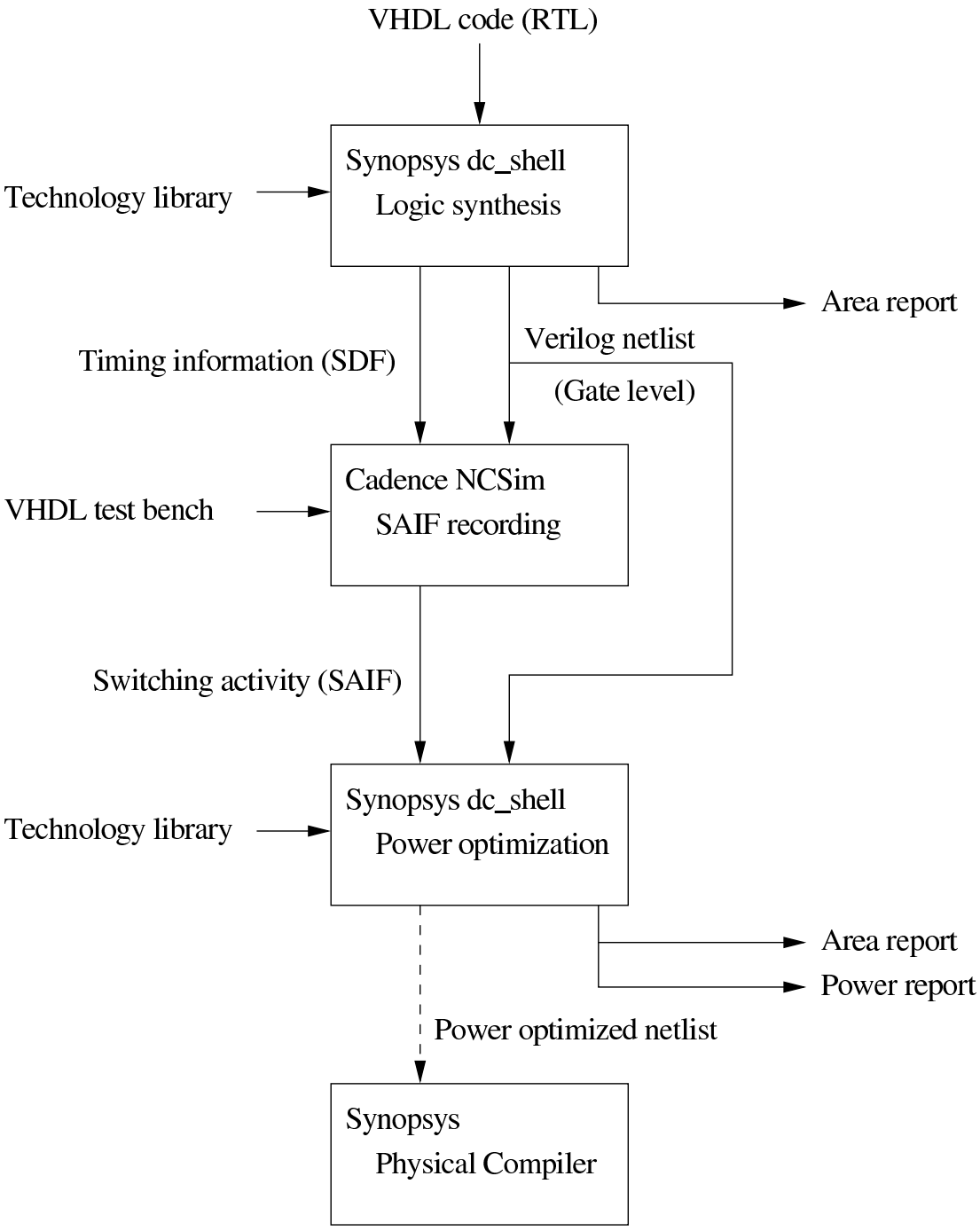


Figure 5.5: Synthesis flow.

consumption without wire loads can be directly compared to the power consumption of the current back-end chip, where similar wires are already present. The power consumption reported pre-routing can be viewed as an estimated overhead that will be introduced by the NoC. Otherwise the power consumption of the wires in the current system should be deducted from the total power consumption for a fair comparison.

A more correct result can be obtained by manually adding load to the links to model the expected wire loads. Wires loads can be estimated by various more or less realistic models. An overview of wire models can be found in [19]. For this purpose a lumped capacitive load is sufficient. Many factors come into play for the actual wire loads, which are impossible to account for. The purpose of adding wire loads is to get a slightly more realistic power estimate. The actual power consumption can not be found without integrating the NoC into the actual system and do physical planning and route the wires.

The link length can quickly be estimated from the dimensions of the current back-end chip, and used along with the technology specification to estimate the capacitance of a wire. This capacitance could be introduced in the netlist either by directly editing the net list or by adding logic load similar to the wire load. The netlist is generated by the flow and is not human readable, which makes modifications more difficult. Another approach is to add extra gates to the wire with a load similar to the wire capacitance estimate. But the advanced synthesis algorithms may avoid the extra loading. A third option is to use the Synopsys facility for wire load modeling. Synopsys supports different wire load models provided by the technology library for estimating wire loads before physical planning. Adding the load before synthesis, will make the tool account for the load and create more realistic driving circuits to the loaded wires. The Synopsys model has been applied.

Power consumption and area estimates have been produced both with and without modeling the wire loads. Excluding the wire load model estimates the cost of the logic and the buffering of the network. The wire model setup approximates the wire loads based on the size of the system. Overriding the size by specifying the size of the entire system, i.e. the size of the current back-end system, gives more realistic wire loads. Furthermore driving circuits are synthesized to fit the increased wire loads. The downside of the Synopsys model is that wire loads are distributed evenly among all wires. In a real implementation the wires within the routing nodes and the NAs are most likely to be very short with no significant wire load, while the NoC links are longer and have greater loads. Thus the estimate including wire load is very pessimistic, but provides an upper limit on power consumption.

5.6 Verification

Testing and verification are integral parts of design implementation. Testing is done both at RTL and gate-level. RTL simulation is cycle accurate and preserves all internal signals, which makes it excellent for debugging. The purpose of gate-level simulation is to verify the functionality of the netlist generated by the synthesis tool.

This simulation includes the gate timing information to verify the system timing. Furthermore the switching activity of all internal nets can be extracted for power estimation.

Functional tests have been carried out at RTL to verify the implemented components. Initially blocks have been tested manually in small setups, before putting them together to form a NoC. A testbench for the entire NoC has been made for full scale functional test and to evaluate the NoC performance. In this main testbench the NoC is configured and stimulated by behavioral black-box components acting as the programming unit and the blocks found in the back-end system. The test data used by these components are generated using scripts and tools implemented in C and Perl. The black-box components generate packets to resemble the communication of the back-end system and log incoming packets. These logs are then processed by scripts to verify the NoC functionality and extract NoC performance data, like latencies and link utilization. The main test bench is illustrated in Figure 5.6.

The main testbench is a functional test that tests both the indidual NoC components and their ability to communicate. The NAs are programmed by the programming unit and then the test blocks exercise the network. The test packets are tagged to identify the packets at the destination. The traffic logs are compared to the traffic input and the network configuration to verify the network functionality. The test pattern can be changed by generating new configuraions for the test blocks using the test tools developed.

The network has been synthesized and verified by gate-level simulations at nominated clock frequency of the current back-end system. The testbench from the RTL simulation is reused to verify the netlist produced by the synthesis tool and the timing constraints of the current back-end system are met.

Neither functional testing nor gate-level simulation revealed any bugs or timing constraint violations. The test pattern used resembles the data flow in the back-end system and thereby gives a fair picture of the network functionality. Routing nodes and NAs are tested under different load conditions in different part of the system. However, a structural test of each individual components is the only way for complete verification. This has not been done.

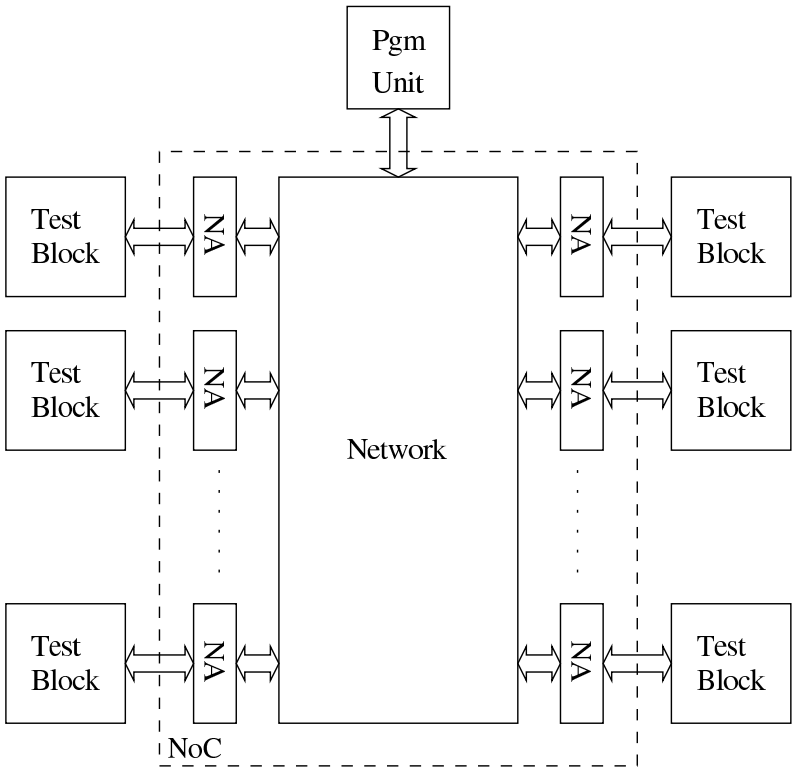


Figure 5.6: Main testbench.

Chapter 6

Results

In this chapter the results of the implemented NoCs are presented. The NoCs have been simulated at RTL using ModelSim to evaluate performance, while physical costs have been estimated using the tool flow provided by *Widex A/S*.

To put the results obtained into perspective they are compared to the current back-end system design. The absolute numbers are inside information held by *Widex A/S* that can not be brought here. All results are therefore listed relative to these figures instead.

6.1 NoC performance

The performance evaluation of the NoCs requires realistic traffic. Random traffic will not give a fair picture of the NoC performance, as it has been designed specifically for the back-end system.

Using the signals of the current back-end system and their usage, a traffic model has been developed to emulate the behavior of the back-end system. The model is designed to model the traffic as realistically as possible. Packets in the model are sized to fit the bit-width of the signals in the back-end system. Signals may be combined into one packet to reduce the network overhead. These kinds of optimizations have not been included in the model, as they require more insight into the blocks of the system to know if they are feasible. The only assumption is that the blocks are able to multiplex and demultiplex the signals internally. To simplify the model, all packet sizes are scaled up to a common size.

6.1.1 NoC utilization

The load of the NoC is determined by the mapping of the application graph onto the designed NoCs. The load on each link can be estimated using the bandwidths listed in Table 3.2 and the mapping shown previously in Figure 4.20. The exact flow of packets through the network is determined using RTL simulation. The network

utilizations are illustrated in Figure 6.1 and Figure 6.2. The numbers are packets per sample period.

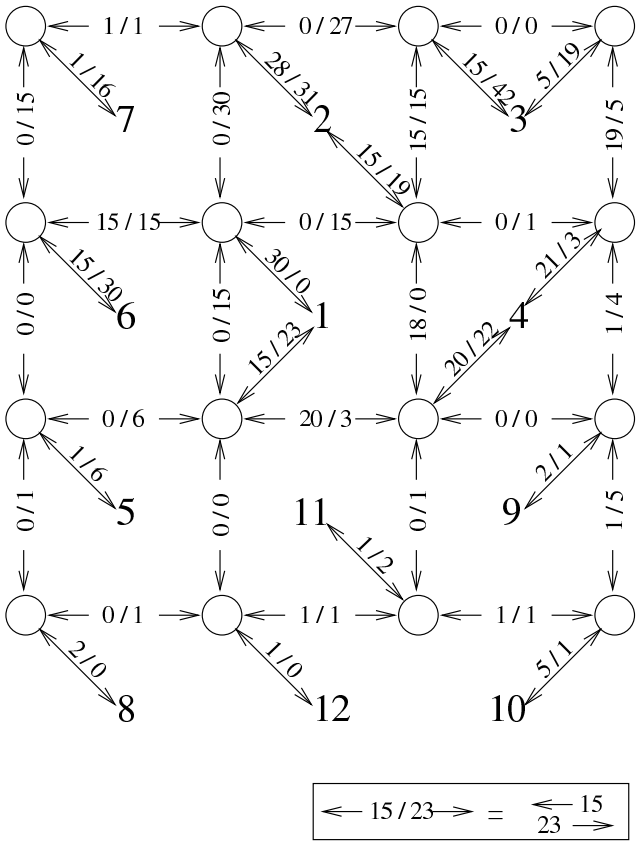


Figure 6.1: NoC utilization in the regular mesh topology.

6.1.2 NoC latency

The NoC introduces long buffered paths to the communication structure. Using a network will cause increased communication latency compared to point-to-point communication. The back-end system operates at a low clock frequency, which makes NoC latency even more critical. The NoC latency is influenced by the topology, application mapping and implementation.

The latencies have been simulated using the traffic of the model described previously in this section. The latency distributions of all communication in the system are shown in Figure 6.3, while the average latency for each topology is listed in Table 6.1. The latencies are measured from input to NA at the source to output from the NA at the destination including any delays caused by the NoC. The latencies presented here are all listed as clock cycles.

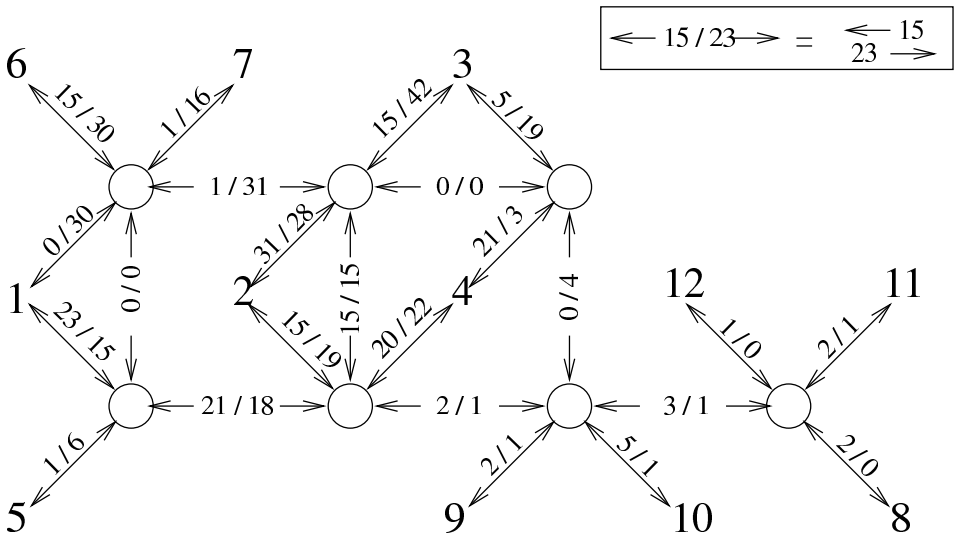


Figure 6.2: NoC utilization in the optimized mesh topology.

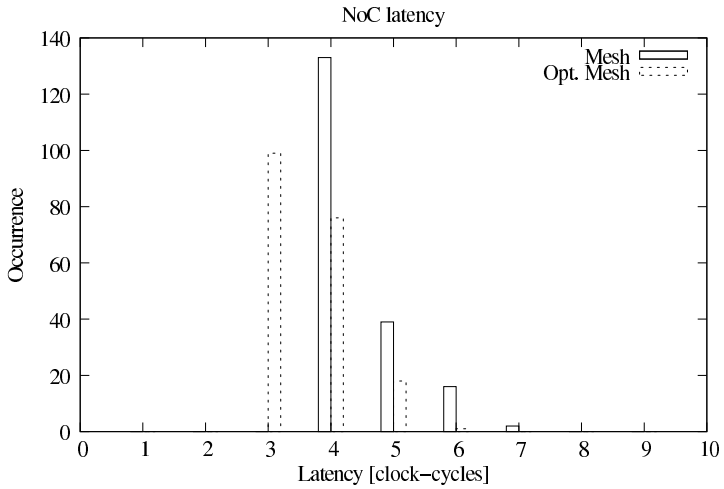


Figure 6.3: Latency distribution in the NoC using mesh and optimized mesh topology

Topology	Avg. latency [clock cycles]
Mesh	4.50 cycles
Optimized mesh	3.67 cycles

Table 6.1: Average end-to-end latencies.

The latency impact of adding an extra 15-channel sample stream in the optimized NoC is seen in Figure 6.4. The added signal is worst case, travelling the longest possible distance in the network and arriving at Unit 12. That accounts for the high latency impact in Unit 12, while the only block affected is Unit 11 by an increased average latency of one clock cycle.

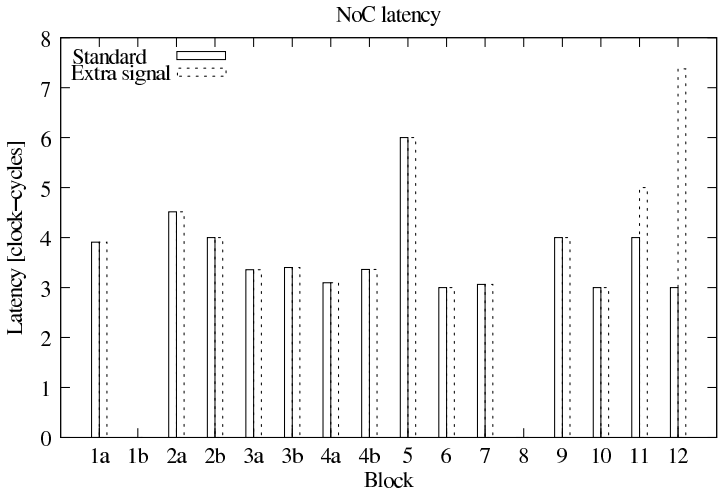


Figure 6.4: Latency impact of adding extra signal.

6.2 Network costs

Estimating the costs of a NoC implementation of the back-end system is the main goal of this project. The costs have been estimated using the methods described in Section 5.5 using Synopsys. All results are estimates based on the cell net list alone. No floor-planning, placement or wire routing has been done. The main results are presented in Table 6.2 below, while further comparisons and results are presented in the following sections. These results represent the costs of the NoC alone. Replacing the communication structure in a system requires modifications to the blocks, which have not been accounted for in these cost estimations.

NoC	Area	Power	Power (w. wire load)
Mesh	16.3%	12.1%	-
Optimized mesh	7.5%	5.9%	11.6%

Table 6.2: NoC cost summary.

6.2.1 Area

The area estimate is based on the synthesized cell net list. It contains all cells used to implement the NoC in the 0.13μ technology. This is a fairly good area estimate, which can be compared directly to the cost of the current back-end system design. However the net interconnect area is not included. All area results are based on synthesis without wire modeling.

Both NoCs have been synthesized to estimate area and power costs. The results are compared to current back-end system costs in Table 6.2.

The area costs of the two implemented topologies are compared in Figure 6.5, which reveals the significant area reduction of the optimized NoC. It is also clear from the figure that the main contributor to the area costs is the routing nodes. The number of NAs is the same for both NoCs, the cost reduction is obtained by reducing the number of routing nodes alone.

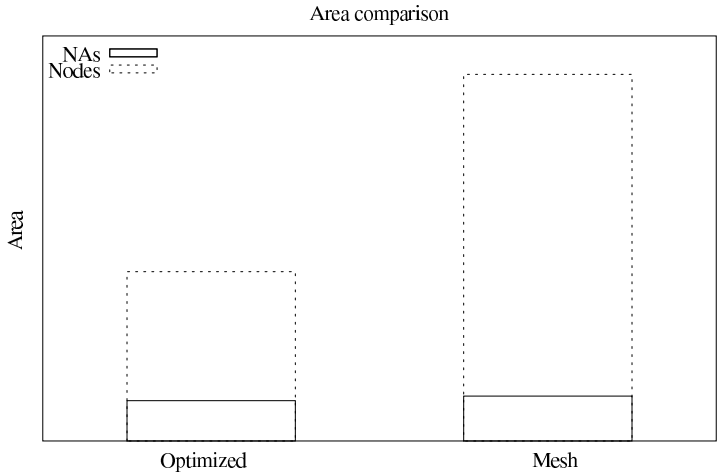


Figure 6.5: Area comparison.

A more detailed cost overview is available in Appendix A. All routing nodes and NAs have been optimized individually during synthesis for optimal results, which causes the slight variation in area costs among the entities. Overall, the combinational and noncombinational areas contribute equally to the total area.

Synopsys synthesis using the wire load model adds load to all internal wires and creates a netlist with stronger driving circuits for all signals. This netlist therefore very pessimistic for area estimation and has not been used.

6.2.2 Power consumption

The power consumption estimates are based on realistic switching activity in synthesized net lists. Estimates both with and without wire load modeling have been conducted to approximate the NoC influence on power consumption.

Both implemented NoC topologies have been synthesized and internal switching activity has been obtained by gate-level simulations. The NoC has been stimulated by packet flow of the traffic model of the back-end system containing random payload data. The power estimates can not be compared directly to the power consumption of the back-end system, because it includes the power consumption of wires that will be replaced by the NoC. However, the estimates without wire loading only include the cell power consumption, which will be added to the total system power consumption. This power consumption overhead caused by the NoC hardware is listed in Table 6.2.

Wire load modeling prior to physical planning is based on statistics gathered from other designs and is not very accurate. Using the power consumption of the wires in the current back-end system might be a better estimate than using a standard model. However, synthesis using a standard wire load model has been carried out for comparison. As mentioned earlier, the wire model is applied to all wires in the design, which makes the estimate very pessimistic for NoC based systems. The power consumption of the optimized NoC including wire loads is listed in Table 6.2. This includes all power consumption caused by block communication except the two preserved dedicated wires and can not be considered as the NoC power consumption overhead. The real power consumption overhead is expected to be somewhere between the two power estimates. The wire load model has not been applied to the power estimate of the regular mesh network.

When comparing the two NoC topologies, the results reveal an overall power consumption reduction similar to the area reduction. The contributions of each individual routing node and NA are listed in Appendix A. From the variations in power consumption it is obvious that the switching activity has great influence. Routing nodes with high utilization consume more energy. One might expect higher average power consumption within nodes in the optimized topology due to higher utilization. However, shorter paths lead to less switching activity and reduces the power consumption. The average power consumption of the routing nodes of the two topologies differs by a mere 5%.

An interesting figure to determine in relation to system flexibility is the cost of adding an extra signal to the system. The price is obviously more switching activity and increased power consumption. An alternative traffic model including an extra signal has been power estimated in the optimized NoC. The signal generates traffic equivalent to 15 audio samples per sample period. The power estimate revealed increased power consumption in nodes along the signal path and caused an overall increase of 11%.

Chapter 7

Discussion

This chapter will discuss the results obtained in this project, put them into perspective and suggest future work.

7.1 Result Elaboration

Looking at the latency impact of the two NoC topologies implemented, it is seen that using a NoC has a price. Signals no longer arrive immediately at the destination, which will affect the system and must be accounted for in the block design.

Most NoCs presented in NoC research are targeted at large SoC systems as general interconnect structures. In these systems performance and modular design are important factors. The NoC replaces advanced bus interconnects, to provide a fast and flexible communication structure. The blocks are typically processors, memory and interface controllers, which are pipelined for maximum performance and designed to cope with communication bottlenecks by local caching. Replacing the communication structure with a NoC does not require overall changes in the system design.

Very small battery powered systems, like digital hearing aids, are characterized by tight area and power constraints. Tightly integrated design means a high level of application specific optimization and limited flexibility. The blocks are small and work close together using point-to-point communication. Using a NoC approach for internal communication in this type of system means a rather different view on block communication. The existing point-to-point solution is efficient, but not flexible. What distinguish NoCs designed for large SoCs from a NoC solution for small dedicated system is that the NoC is not used to solve bottlenecks in bus interconnects, but adding flexibility to the system. The NoC adds extra costs in terms of area and power consumption to the system, but adds value in terms of a more flexible system design. It is a trade off between costs and system flexibility.

The costs of the NoC approach are compared to the current design to justify whether it is feasible or not for future system designs. An area overhead of 6.6% is predicted in [8] for large system, when using a NoC interconnection structure. To

obtain a similar overhead for a small system, like a digital hearing aid, the NoC has to be designed to exactly match the system requirements.

The results show an area overhead of 7.5% for the optimized NoC, which is reasonable considering the size of the entire system. Even though the power estimates have several sources of uncertainties, they do provide a reasonable guess. The cell power consumption has a mere overhead of 5.9%, while the wire load model estimates a power consumption of 11.6% compared to the total back-end system power consumption, which means the power consumption after physical planning can be expected to be somewhere in between. The wire load is hard to estimate as there is no way to predict the physical layout precisely. But the estimated cell power consumption has a small margin of error.

Regarding flexibility, the power cost of adding or redirecting signals in the system is very interesting. A NoC power consumption increase of 11% seems drastic. However, this setup demonstrates the absolute worst case scenario, by adding a signal with high bandwidth demands at the longest possible distance in the NoC. Several similar signals are already a part of the system communication, which shows the importance of minimizing communication paths during application mapping. The highly optimized application mapping means low power consumption in use-cases used for the application mapping. Slightly increased power consumption should be expected when redirecting or adding signals. However this price may be worth paying for added functionality or fixing bugs.

Both NoC topologies are based on the same components, which are designed to use as few decoupling buffers as possible to keep the cost and latency down to a minimum. As stated in the results around half the area is spend on buffers. The number of buffers in the routing node can not be further reduced without significant performance impact. Area can be saved by reducing the bit width of the links, which reduces the width of the buffers similarly. Reducing the link width means more flits per packet, less overall bandwidth and increased latency. This can be countered by increasing the clock frequency, which will on the other hand increase the power consumption. The clock frequency used by the current back-end system is thereby the main limiting factor of bandwidth and latency. Granted that the traffic does not increase similarly, faster clocks in future systems will allow more data serialization and save area in the routing nodes.

Using a particular topology, the latency is affected by the routing scheme and the mapping of the application to the topology. Minimal routing is employed for both NoCs implemented and the application is mapped with minimizing the number of hops is mind. However the topology has great influence too, which becomes clear when looking at the average latencies of the two NoCs. The average latency of the regular mesh is 22% longer than in the optimized NoC. This is a direct consequence of attaching more blocks to each routing node.

Comparing the regular topology to the custom one show clearly that a general NoC solution is not feasible. The regular mesh has greater flexibility, but can not justify approximately double area and power consumption. The optimized topology still maintains the specified level of flexibility, which makes custom topologies the

best choice for application specific systems.

Comparing the NoC design presented in this project to other NoCs, like AEthereal[6] and MANGO[1], shows that this design differs from the others by being application optimized and rely on application mapping and planning rather than avanced network features to avoid expensive GS implementations. It is not suitable as a generic NoC, but a low-cost NoC targeted at small systems with a limited number of use-cases which allow application specific optmizations.

Due to the lack of NoC suited interfaces on the blocks of the current system design, the NoC has not been integrated into the actual system. The results of this project serve as a rough estimate of the costs that can be expected by using a NoC interconnect in future back-end system designs. The system interconnect strategy is a fundamental system design decision.

It is clear that replacing a point-to-point structure with a NoC in a tightly integrated system based on hard-coded processing units does not add much flexibility to the system. A NoC is a different system design methodology, which adds flexible communication to a flexible system design. System configurability has to be an integral part of the system design, to fully exploit the potential of a NoC.

7.2 NoC in DSP-systems

This project is a case-study on a NoC communication structure for a digital hearing aid. However the overall results obtained are not restricted to this system, but apply to DSP-system in general.

Increased digital processing power has moved the trend towards general flexible processing platforms and implementing signal processing algorithms in software. This does not only apply for large DSP-systems, but also for small embedded DSP-systems. Low power DSP-platforms with configurable embedded DSP-units are commercially available.

A more flexible hardware platform means easier reuse. The DSP hardware platform may last several product generations by just algorithm improvements. Hard-coded DSP-algorithms are prone to bugs, which are extremely hard to correct later. A more flexible software solution will allow such late changes and field upgrades.

NoCs call for a standard interface for system blocks. A standard interface requires more design effort in the first place, but means easier reuse later. New systems can be compiled from existing and new blocks. Soft core solutions are already commercially available [20] and demonstrated for DSP-applications in [21]. Similar systems may very well be based on NoC communication structures in the future. The NoC flexibility has great potential for DSP systems, where processing units can be rearranged to accommodate different algorithms.

Future CMOS technologies, like 65 nm, will give a significantly higher system transistor count. Furthermore, decreasing standard supply voltage means that the clock frequency can be increased dramatically even for battery powered applications.

A fast system clock makes it feasible to include general purpose processing blocks in the system, which makes NoC based communication very interesting.

Flexible communication structures like NoCs enable a new kinds of system designs. An extreme approach could be a fine grained NoC, probably a hierachical topology, where relatively small DSP-units are attached directly to the NoC, which controls the data flow through the units. A hybrid solution using both general purpose DSP-units and dedicated blocks connected by a NoC may be a more efficient solution. The design must then determine which blocks to be implemented in dedicated blocks and which to implemented as code for DSP-units. It is a trade off that trades power for area and flexibility. A software approach saves area if it is using general purpose resources already available, but on the other hand it comsumes more power than a dedicated block. This approach seems reasonable for future hearing aid systems and other low power applications.

7.3 Future work

Much work is still left on block interfacing and system design before NoC based digital hearing aids will emerge. System design based on NoCs and characterization of NoC suited applications are topics which require more attention in NoC research. A better understanding on how to exploit NoCs is necessary before the added costs become feasible. This is especially true for small DSP-systems.

The NoC components designed in this project are very flexible and can be used to form various topologies. More components could be designed to create a library of NoC components, like the Xpipes library [5], for constructing application specific NoCs for DSP-like systems. This could be extended further by automated topology generation from a specific application graph, like the xpipes compiler [22].

Chapter 8

Conclusion

This project has investigated NoC as a flexible communication structure for small DSP-systems, based on a case study of the digital back-end system designed by *Widex A/S*. The communication requirements of the system have been characterized, in order to derive a suitable NoC design.

A generic NoC and a custom optimized topology NoC have been implemented and synthesized. Area and power reductions of a factor of two have been obtained by optimizing the topology for the application compared to the generic topology. This shows that much can be gained from optimized topologies, which is very important especially in small systems. The NoC components designed are very flexible and can be used to form any topology using routing nodes with five ports.

Synthesis has been done using the *Widex A/S* design flow to enable comparison of the NoC implementation and the current system implementation. The optimized NoC adds a 7.5% area overhead to the system, while the power consumption overhead is estimated to be equivalent to 5.9% of the current back-end system power consumption. These results are reasonable for today's technology and promising for future technologies.

Looking at the obtained results in a greater perspective, they show that NoCs are feasible communication solutions for even small system designs. When the system design is highly optimized, the NoC must also undergo a similar degree of optimization. GS and resource reservation are expensive for small systems, which should rather rely on BE based NoCs optimized for the particular application and use-case planning. Future CMOS technologies will decrease the relative overhead and make NoCs a promising communication structure for even the smallest systems.

Increased system flexibility does not come from a NoC alone, but by exploiting the NoC in the overall system design. The NoC costs estimated here indicate that feasible NoC based hearing aid systems may emerge in the future.

Bibliography

[1] T. Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005. Supervised by Assoc. Prof. Jens Sparsø, IMM.

[2] John Bainbridge and Steve Furber. Systems on a chip - chain: A delay-insensitive chip area interconnect. *IEEE Micro - Institute of Electrical and Electronics Engineers*, 22(5):16–23, 2002.

[3] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, pages 250 –256, 2000.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Qnoc: Qos architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2-3):105–128, 2004.

[5] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine, IEEE*, 4(2):18–31, 2004.

[6] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, 2005.

[7] Frits Steenhof, Harry Duque, Björn Nilsson, Kees Goossens, and Rafael Peset Llopis. Networks on chips for high-end consumer-electronics tv system architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 148–153, March 2006.

[8] Brian Towles and William J. Dally. Route packets, net wires: On-chip inteconnectoin networks. *dac*, 00:684–689, 2001.

[9] W.J. Dally and C.I. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[10] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *Computers, IEEE Transactions on*, 54(8):1025–1040, 2005.

[11] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. *VLSI on Annual Symposium, IEEE Computer Society ISVLSI 2002*, pages 117–124, 2002.

[12] Jingcao Hu and R. Marculescu. Application-specific buffer space allocation for networks-on-chip router design. *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 354–361, 2004.

[13] S. Murali, L. Benini, and G. de Micheli. Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees. *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference 2005 (IEEE Cat. No.05EX950C)*, pages 27–32 Vol. 1, 2005.

[14] David E. Culler, Jaswind Pal Singh, and Anoop Gupta. *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann, 1999. CUL d 99:1 2.Ex.

[15] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, 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. *Computers and Digital Techniques, IEE Proceedings-*, vol.150, no.5:294–302, 2003.

[16] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2:896–901, 2004.

[17] Srinivasan Murali, Martijn Coenen, Andrei Rădulescu, Kees Goossens, and Giovanni De Micheli. A methodology for mapping multiple use-cases on to networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 118–123, March 2006.

[18] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(4):551–562, 2005.

[19] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits - A Design Perspective*. Prentice Hall Electronics and VLSI series. Prentice Hall, 2003.

[20] Dresden silicon. http://www.dresdensilicon.com.

[21] T. Limberg, E. Matus, H. Seidel, and G. Fettweis. On design of high performance vector math coprocessors for mobile applications: Samira case study. In *Proc. Dresdner Arbeitstagung Schaltungs- und Systementwurf (DASS'06), Dresden, Germany*, May 2006.

[22] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. xpipescompiler: a tool for instantiating application specific networks on chip. *Proceedings. Design, Automation and Test in Europe Conference and Exhibition*, pages 884–9 Vol.2, 2004.

Appendix A

NoC costs

A.1 Mesh NoC internal costs

A.2 Optimized NoC internal costs

Component	Area %	Power %
node 00	5.5	3.8
na 00	0.8	0.3
node 10	5.5	3.8
na 10	0.8	0.3
node 20	5.5	3.9
na 20	0.8	0.3
node 30	5.6	5.7
na 30	0.8	0.4
node 01	5.5	4.1
na 01	0.8	0.4
node 11	5.4	4.8
na 11	0.8	1.0
node 21	5.4	5.3
na 21	0.8	1.2
node 31	5.5	4.1
na 31	0.8	0.3
node 02	5.5	7.0
na 02	0.8	1.3
node 12	5.4	6.1
na 12	0.8	0.9
node 22	5.4	5.5
na 22	0.8	1.0
node 32	5.5	5.4
na 32	0.8	0.8
node 03	5.6	6.2
na 03	0.8	0.6
node 13	5.5	7.3
na 13	0.8	1.5
node 23	5.5	8.0
na 23	0.8	1.5
node 33	5.6	6.7
na 33	0.8	0.7

Table A.1: Mesh NoC internal costs.

Component	Area %	Power %
node 00	11.0	12.2
na 00a	1.5	2.1
na 00w	1.5	0.8
node 10	10.8	14.4
na 10a	1.5	2.0
na 10s	1.5	2.4
node 20	10.8	5.3
na 20a	1.5	0.5
na 20s	1.5	0.7
node 30	11.0	7.7
na 30n	1.5	0.5
na 30e	1.5	0.6
na 30s	1.5	0.6
node 01	10.8	13.1
na 01n	1.5	1.2
na 01w	1.5	1.8
na 01a	1.5	2.6
node 11	10.8	14.4
na 11n	1.5	3.1
na 11a	1.5	3.1
node 21	10.8	7.9
na 21n	1.5	1.5
na 21a	1.5	1.6

Table A.2: Optimized NoC internal costs.

Appendix B

CD contents

The attached CD-ROM includes all source code from Appendix C.

- **noc_components** All network components and subentities.
- **r_mesh** The optimized NoC including main testbench.
- **mesh_4x4** The mesh NoC including main testbench.
- **tools** C programs and Perl scripts to generate NoC configurations.

Appendix C

Source code

Source for NoC components and NoCs.

C.1 NoC components

C.1.1 Flit handler

```
-------------------------------------------------------------------------------
-- Flit handler
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity flit_handler is
  generic (
    INPUT : rt_direction);
  port (
    clk         : in  std_logic;        -- Clock
    rst         : in  std_logic;        -- Reset
    eop         : in  std_logic;        -- End of packet
    empty       : in  std_logic;        -- Data valid
    rd          : in  std_logic;        -- Read buffer (First flit detection)
    dir_i       : in  std_logic_vector(HOP_BITS-1 downto 0);
    dir_o       : out std_logic_vector(HOP_BITS downto 0);
    rot         : out std_logic);       -- Rotate header (Indicates header flit)

end flit_handler;

architecture fh of flit_handler is

signal dir_in, dir_reg : std_logic_vector(HOP_BITS downto 0);
signal dir_reg_en : std_logic;

begin  -- fh

  process (dir_i, eop, dir_reg, empty, rd)
  begin  -- process
    dir_in <= "0" & dir_i;
    dir_o <= "0" & dir_i;
```

```
    dir_reg_en <= '0';
    rot <= '0';
    if empty = '0' then
    if eop = '0' then
      if dir_reg = rt_x then
        rot <= '1';
        if rd = '1' then
          dir_reg_en <= '1';
        end if;
        if INPUT = rt_a then
          dir_o <= "0" & dir_i;
        else
          if dir_i = INPUT(1 downto 0) then
            dir_in <= rt_a;
            dir_o <= rt_a;
          else
            dir_o <= "0" & dir_i;
          end if;
        end if;
      else
        rot <= '0';
        dir_o <= dir_reg;
      end if;
    else
      if dir_reg /= rt_x then
        rot <= '0';
        dir_reg_en <= '1';
        dir_in <= rt_x;
        dir_o <= dir_reg;
      else
        rot <= '1';
        if INPUT = rt_a then
          dir_o <= "0" & dir_i;
        else
          if dir_i = INPUT(1 downto 0) then
            dir_o <= rt_a;
          else
            dir_o <= "0" & dir_i;
          end if;
        end if;
      end if;
    end if;

    end if;
end process;


-- purpose: Register
-- type   : sequential
-- inputs : clk, rst, dir_in, dir_reg_en
-- outputs: dir_reg
process (clk, rst)
begin  -- process
  if rst = '0' then                     -- asynchronous reset (active low)
    dir_reg <= rt_x;
  elsif clk'event and clk = '1' then    -- rising clock edge
    if dir_reg_en = '1' then
      dir_reg <= dir_in;
    end if;
  end if;
end process;
```

```
end fh;
```

C.1.2 BE router

```
--------------------------------------------------------------------------------
-- BE Router
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity be_router is

  generic (
    DIR : rt_direction := "000");        -- Routing direction

  port (
    clk      : in  std_logic;           -- Clock
    rst      : in  std_logic;           -- Reset
    src_0    : in  std_logic_vector(4 downto 0);             -- Source 0
    src_1    : in  std_logic_vector(4 downto 0);             -- Source 1
    src_2    : in  std_logic_vector(4 downto 0);             -- Source 2
    src_3    : in  std_logic_vector(4 downto 0);             -- Source 3
    src_4    : in  std_logic_vector(4 downto 0);             -- Source 4
    src_5    : in  std_logic_vector(4 downto 0);             -- Source 5
    src_6    : in  std_logic_vector(4 downto 0);             -- Source 6
    src_7    : in  std_logic_vector(4 downto 0);             -- Source 7
    vc_empty : in  std_logic_vector(VC_bit downto 0);   -- Empty output VC's
    vc_sel   : out std_logic_vector(0 downto 0);  -- VC select
    src_sel  : out std_logic_vector((4*VC)-1 downto 0));  -- Source select/read

end be_router;

architecture src_order of be_router is

signal src_0_dir, src_1_dir, src_2_dir, src_3_dir, src_4_dir, src_5_dir,
  src_6_dir, src_7_dir : rt_direction;        -- Directions
signal src_0_eop, src_1_eop, src_2_eop, src_3_eop, src_4_eop, src_5_eop,
  src_6_eop, src_7_eop : std_logic;           -- EOP
signal src_0_empty, src_1_empty, src_2_empty, src_3_empty, src_4_empty,
  src_5_empty, src_6_empty, src_7_empty : std_logic;           -- Empty
signal vc0_res, vc1_res, vc0_res_reg, vc1_res_reg
  : std_logic_vector(3 downto 0);  -- VC reservation registers

begin  -- src_order

src_0_dir <= src_0(2 downto 0);
src_0_eop <= src_0(3);
src_0_empty <= src_0(4);
src_1_dir <= src_1(2 downto 0);
src_1_eop <= src_1(3);
src_1_empty <= src_1(4);
src_2_dir <= src_2(2 downto 0);
src_2_eop <= src_2(3);
src_2_empty <= src_2(4);
src_3_dir <= src_3(2 downto 0);
src_3_eop <= src_3(3);
src_3_empty <= src_3(4);
src_4_dir <= src_4(2 downto 0);
```

```
src_4_eop <= src_4(3);
src_4_empty <= src_4(4);
src_5_dir <= src_5(2 downto 0);
src_5_eop <= src_5(3);
src_5_empty <= src_5(4);
src_6_dir <= src_6(2 downto 0);
src_6_eop <= src_6(3);
src_6_empty <= src_6(4);
src_7_dir <= src_7(2 downto 0);
src_7_eop <= src_7(3);
src_7_empty <= src_7(4);


-- purpose: Routing algorithm
-- type   : combinational
-- inputs : src_0, src_1, src_2, src_3, src_4, src_5, src_6, src_7
-- outputs: src_sel, vc_empty, vc0_res, vc1_res
rt: process (src_0_dir, src_1_dir, src_2_dir, src_3_dir, src_4_dir, src_5_dir,
             src_6_dir, src_7_dir, src_0_eop, src_1_eop, src_2_eop, src_3_eop,
             src_4_eop, src_5_eop, src_6_eop, src_7_eop, src_0_empty,
             src_1_empty, src_2_empty, src_3_empty, src_4_empty, src_5_empty,
             src_6_empty, src_7_empty, vc0_res_reg, vc1_res_reg)
begin  -- process rt
  vc0_res <= vc0_res_reg;
  vc1_res <= vc1_res_reg;
  -- Ongoing transmission through VC0
  if vc0_res_reg = "1000" and src_0_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"01";
    if src_0_eop = '1' then
      vc0_res <= "0000";
    end if;
  elsif vc0_res_reg = "1001" and src_1_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"02";
    if src_1_eop = '1' then
      vc0_res <= "0000";
    end if;
  elsif vc0_res_reg = "1010" and src_2_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"04";
    if src_2_eop = '1' then
      vc0_res <= "0000";
    end if;
  elsif vc0_res_reg = "1011" and src_3_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"08";
    if src_3_eop = '1' then
      vc0_res <= "0000";
    end if;
  elsif vc0_res_reg = "1100" and src_4_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"10";
    if src_4_eop = '1' then
      vc0_res <= "0000";
    end if;
  elsif vc0_res_reg = "1101" and src_5_empty = '0' and vc_empty(0) ='1' then
    vc_sel <= "0";
    src_sel <= X"20";
    if src_5_eop = '1' then
      vc0_res <= "0000";
    end if;
```

```
elsif vc0_res_reg = "1110" and src_6_empty = '0' and vc_empty(0) ='1' then
  vc_sel <= "0";
  src_sel <= X"40";
  if src_6_eop = '1' then
    vc0_res <= "0000";
  end if;
elsif vc0_res_reg = "1111" and src_7_empty = '0' and vc_empty(0) ='1' then
  vc_sel <= "0";
  src_sel <= X"80";
  if src_7_eop = '1' then
    vc0_res <= "0000";
  end if;
-- Ongoing transmission through VC1
elsif vc1_res_reg = "1000" and src_0_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"01";
  if src_0_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1001" and src_1_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"02";
  if src_1_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1010" and src_2_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"04";
  if src_2_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1011" and src_3_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"08";
  if src_3_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1100" and src_4_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"10";
  if src_4_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1101" and src_5_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"20";
  if src_5_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1110" and src_6_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"40";
  if src_6_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc1_res_reg = "1111" and src_7_empty = '0' and vc_empty(1) ='1' then
  vc_sel <= "1";
  src_sel <= X"80";
  if src_7_eop = '1' then
    vc1_res <= "0000";
  end if;
elsif vc_empty(0) = '1' then
```

```
    vc_sel <= "0";
    if src_0_dir = DIR and src_0_empty = '0' and vc1_res_reg /= "1000" then
      src_sel <= X"01";
      if src_0_eop = '0' then
        vc0_res <= "1000";
      end if;
    elsif src_1_dir = DIR and src_1_empty = '0' and vc1_res_reg /= "1001" then
      src_sel <= X"02";
      if src_1_eop = '0' then
        vc0_res <= "1001";
      end if;
    elsif src_2_dir = DIR and src_2_empty = '0' and vc1_res_reg /= "1010" then
      src_sel <= X"04";
      if src_2_eop = '0' then
        vc0_res <= "1010";
      end if;
    elsif src_3_dir = DIR and src_3_empty = '0' and vc1_res_reg /= "1011" then
      src_sel <= X"08";
      if src_3_eop = '0' then
        vc0_res <= "1011";
      end if;
    elsif src_4_dir = DIR and src_4_empty = '0' and vc1_res_reg /= "1100" then
      src_sel <= X"10";
      if src_4_eop = '0' then
        vc0_res <= "1100";
      end if;
    elsif src_5_dir = DIR and src_5_empty = '0' and vc1_res_reg /= "1101" then
      src_sel <= X"20";
      if src_5_eop = '0' then
        vc0_res <= "1101";
      end if;
    elsif src_6_dir = DIR and src_6_empty = '0' and vc1_res_reg /= "1110" then
      src_sel <= X"40";
      if src_6_eop = '0' then
        vc0_res <= "1110";
      end if;
    elsif src_7_dir = DIR and src_7_empty = '0' and vc1_res_reg /= "1111" then
      if src_7_eop = '0' then
        vc0_res <= "1111";
      end if;
      src_sel <= X"80";
    else
      src_sel <= X"00";
    end if;
  elsif vc_empty(1) = '1' then
    vc_sel <= "1";
    if src_0_dir = DIR and src_0_empty = '0' and vc0_res_reg /= "1000" then
      src_sel <= X"01";
      if src_0_eop = '0' then
        vc1_res <= "1000";
      end if;
    elsif src_1_dir = DIR and src_1_empty = '0' and vc0_res_reg /= "1001" then
      src_sel <= X"02";
      if src_1_eop = '0' then
        vc1_res <= "1001";
      end if;
    elsif src_2_dir = DIR and src_2_empty = '0' and vc0_res_reg /= "1010" then
      src_sel <= X"04";
      if src_2_eop = '0' then
        vc1_res <= "1010";
      end if;
    elsif src_3_dir = DIR and src_3_empty = '0' and vc0_res_reg /= "1011" then
```

```
      src_sel <= X"08";
      if src_3_eop = '0' then
        vc1_res <= "1011";
      end if;
    elsif src_4_dir = DIR and src_4_empty = '0' and vc0_res_reg /= "1100" then
      src_sel <= X"10";
      if src_4_eop = '0' then
        vc1_res <= "1100";
      end if;
    elsif src_5_dir = DIR and src_5_empty = '0' and vc0_res_reg /= "1101" then
      src_sel <= X"20";
      if src_5_eop = '0' then
        vc1_res <= "1101";
      end if;
    elsif src_6_dir = DIR and src_6_empty = '0' and vc0_res_reg /= "1110" then
      src_sel <= X"40";
      if src_6_eop = '0' then
        vc1_res <= "1110";
      end if;
    elsif src_7_dir = DIR and src_7_empty = '0' and vc0_res_reg /= "1111" then
      src_sel <= X"80";
      if src_7_eop = '0' then
        vc1_res <= "1111";
      end if;
    else
      src_sel <= X"00";
    end if;
  else
    src_sel <= X"00";
    vc_sel <= "0";

  end if;

end process rt;

-- purpose: Reservation registers
-- type   : sequential
-- inputs : clk, rst, vc0_res, vc1_res
-- outputs: vc0_res_reg, vc1_res_reg
res_reg: process (clk, rst)
begin  -- process res_reg
  if rst = '0' then                     -- asynchronous reset (active low)
    vc0_res_reg <= (others => '0');
    vc1_res_reg <= (others => '0');
  elsif clk'event and clk = '1' then    -- rising clock edge
    vc0_res_reg <= vc0_res;
    vc1_res_reg <= vc1_res;
  end if;
end process res_reg;

end src_order;
```

C.1.3 Data buffer

```
-------------------------------------------------------------------------------
-- Data buffer
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
```

```
use IEEE.std_logic_1164.all;
use work.types.all;

entity data_buffer is
  generic (
    DATA_WIDTH : integer := 32;
    ADDR_BITS : integer := 4);
  port (
    data_i : in  std_logic_vector(ADDR_BITS + DATA_WIDTH-1 downto 0);
    data_o : out std_logic_vector(ADDR_BITS + DATA_WIDTH-1 downto 0);
    clk    : in  std_logic;                                 -- Clock
    rst    : in  std_logic;                                 -- Reset
    rd     : in  std_logic;                                 -- Read
    wr     : in  std_logic;                                 -- Write
    full   : out std_logic;                                 -- Write
    empty  : out std_logic);                                -- Empty

end data_buffer;

architecture buf_reg of data_buffer is

begin  -- buf_reg

  -- purpose: Buffer register
  -- type   : sequential
  -- inputs : clk, rst, data_i, rd, wr
  -- outputs: data_o, empty
  reg: process (clk, rst)
  begin  -- process reg
    if rst = '0' then                   -- asynchronous reset (active low)
      data_o <= (others => '0');
      empty <= '1';
      full <= '0';
    elsif clk'event and clk = '1' then  -- rising clock edge
      if rd = '1' then
        empty <= '1';
        full <= '0';
      end if;
      if wr = '1' then
        data_o <=  data_i;
        empty <= '0';
        full <= '1';
      end if;
    end if;
  end process reg;

end buf_reg;
```

C.1.4 VC buffer

```
-------------------------------------------------------------------------------
-- VC buffer - Virtual Channel buffer
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity vc_buffer is
```

```
  port (
    data_i : in  std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data input
    data_o : out std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data output
    clk    : in  std_logic;                                -- Clock
    rst    : in  std_logic;                                -- Reset
    rd     : in  std_logic;                                -- Read
    wr     : in  std_logic;                                -- Write
    full   : out std_logic;                                -- Full
    empty  : out std_logic);                               -- Empty

end vc_buffer;

architecture buf_reg of vc_buffer is

begin  -- buf_reg

  -- purpose: Buffer register
  -- type   : sequential
  -- inputs : clk, rst, data_i, rd, wr
  -- outputs: data_o, empty
  reg: process (clk, rst)
  begin  -- process reg
    if rst = '0' then                   -- asynchronous reset (active low)
      data_o <= (others => '0');
      empty <= '1';
      full <= '0';
    elsif clk'event and clk = '1' then  -- rising clock edge
      if wr = '1' then
        data_o <=  data_i;
        empty <= '0';
        full <= '1';
      end if;
      if rd = '1' then
        empty <= '1';
        full <= '0';
      end if;
    end if;
  end process reg;

end buf_reg;
```

C.1.5 Address table

```
-------------------------------------------------------------------------------
-- NA address tbl - NA address table
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.types.all;

entity na_address_tbl is

  generic (
    ADDR_BITS : integer := 4);            -- Number of entries

  port (
    addr_i : in std_logic_vector(ADDR_BITS-1 downto 0);
    addr_rd : in std_logic_vector(ADDR_BITS-1 downto 0);
```

```
    dat_i : in std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    dat_o : out std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    rw : in std_logic;                  -- Read/write
    clk : in std_logic;
    rst : in std_logic);
end na_address_tbl;


architecture tbl of na_address_tbl is

  type Regfile_type is array (natural range<> )
    of std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);

  signal Regfile_Coff : Regfile_type(0 to 2**4);

begin

dat_o <= Regfile_Coff(conv_integer(unsigned(addr_rd)));

process (clk)
begin  -- process
  if clk'event and clk = '1' then    -- rising clock edge
    if rw = '1' then
      Regfile_Coff(conv_integer(unsigned(addr_i))) <= dat_i;
    end if;
  end if;
end process;

end tbl;
```

C.1.6 Pack Unit

```
-------------------------------------------------------------------------------
-- NA packet gen - Pack unit
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity na_packet_gen is

  generic (
    DATA_WIDTH : integer);              -- Input data width

  port (
    data_i    : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    route_i   : in  std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    type_i    : in  std_logic_vector(TYPE_BITS-1 downto 0);
    noc_fw_o  : out noc_link_fw;
    noc_bw_i  : in  noc_link_bw;
    buf_rd    : out std_logic;
    buf_empty : in  std_logic;
    clk       : in  std_logic;
    rst       : in  std_logic);

end na_packet_gen;

architecture packet_gen of na_packet_gen is
```

```
  signal flit_no, flit_no_reg, vc_res, vc_res_reg : integer;
  signal testv : integer := 0;
begin  -- packet_gen

  process(data_i, buf_empty, route_i, noc_bw_i, flit_no_reg, type_i, vc_res_reg)
  begin  -- process data_i, buf_empty, route_i, noc_bw_i
    noc_fw_o(EOP_BIT_POS) <= '1';  -- eop
    noc_fw_o(TYPE_BITS+TYPE_BIT_POS-1 downto TYPE_BIT_POS)
      <= type_i(TYPE_BITS-1 downto 0);
    noc_fw_o(FLIT_WIDTH + VC_bit) <= '0';  -- dv
    noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "0";  --vc_id
    noc_fw_o(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET)
      <= route_i;
    noc_fw_o(DATA_WIDTH-1 downto 0) <= data_i(DATA_WIDTH-1 downto 0);
    flit_no <= flit_no_reg;
    vc_res <= vc_res_reg;
    buf_rd <= '0';
    if DATA_WIDTH < HDR_ADDR_OFFSET then
      noc_fw_o(HDR_ADDR_OFFSET-1 downto DATA_WIDTH) <= (others => '0');
    end if;

    if buf_empty = '0' then
      if (noc_bw_i(0) = '1' and (vc_res_reg = 0 or vc_res_reg = VC)) then
        if DATA_WIDTH <= HDR_ADDR_OFFSET then
          noc_fw_o(EOP_BIT_POS) <= '1';  -- eop
          noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
          noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "0";  --vc_id
          noc_fw_o(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET)
            <= route_i;
          noc_fw_o(DATA_WIDTH-1 downto 0) <= data_i(DATA_WIDTH-1 downto 0);
          if DATA_WIDTH < HDR_ADDR_OFFSET then
            noc_fw_o(HDR_ADDR_OFFSET-1 downto DATA_WIDTH) <= (others => '0');
          end if;
          buf_rd <= '1';
        else
          if flit_no_reg = 0 then
            noc_fw_o(EOP_BIT_POS) <= '0';  -- eop
            noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
            noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "0";  --vc_id
            noc_fw_o(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET)
              <= route_i;
            noc_fw_o(HDR_ADDR_OFFSET-1 downto 0)
              <= data_i(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET);
            buf_rd <= '0';
            flit_no <= flit_no_reg +1;
            vc_res <= 0;
          else
            noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
            noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "0";  -- vc_id
            if DATA_WIDTH-HDR_ADDR_OFFSET-flit_no_reg*FLIT_DATA_WIDTH < 0 then
              noc_fw_o(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                       *FLIT_DATA_WIDTH-1 downto 0)
                <= data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                          *FLIT_DATA_WIDTH-1 downto 0);
              noc_fw_o(FLIT_WIDTH-3 downto DATA_WIDTH-HDR_ADDR_OFFSET
                       -(flit_no-1)*FLIT_DATA_WIDTH) <= (others => '0');
              noc_fw_o(EOP_BIT_POS) <= '1';  -- eop
              buf_rd <= '1';
            testv <= 1;
              flit_no <= 0;
              vc_res <= VC;
            else
```

```
          noc_fw_o(FLIT_WIDTH-3 downto 0)
            <= data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                      *FLIT_DATA_WIDTH-1 downto
                      DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg)
                      *FLIT_DATA_WIDTH);
          noc_fw_o(EOP_BIT_POS) <= '0';  -- eop
          buf_rd <= '0';
          flit_no <= flit_no_reg +1;
          vc_res <= 0;
        end if;
      end if;
    end if;
  elsif noc_bw_i(1) = '1'  and (vc_res_reg = 1 or vc_res_reg = VC) then
    if DATA_WIDTH <= HDR_ADDR_OFFSET then
      noc_fw_o(EOP_BIT_POS) <= '1';  -- eop
      noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
      noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "1";  -- vc_id
      noc_fw_o(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET)
        <= route_i;
      noc_fw_o(DATA_WIDTH-1 downto 0) <= data_i;
      if DATA_WIDTH < HDR_ADDR_OFFSET then
        noc_fw_o(HDR_ADDR_OFFSET-1 downto DATA_WIDTH) <= (others => '0');
      end if;
      buf_rd <= '1';
    else
      if flit_no_reg = 0 then
        noc_fw_o(EOP_BIT_POS) <= '0';  -- eop
        noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
        noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "0"; --vc_id
        noc_fw_o(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET)
          <= route_i;
        noc_fw_o(HDR_ADDR_OFFSET-1 downto 0)
          <= data_i(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET);
        buf_rd <= '0';
        flit_no <= flit_no_reg +1;
        vc_res <= 1;
      else
        noc_fw_o(FLIT_WIDTH + VC_bit) <= '1';  -- dv
        noc_fw_o(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH) <= "1";  -- vc_id
        if DATA_WIDTH-HDR_ADDR_OFFSET-flit_no_reg*FLIT_DATA_WIDTH < 0 then
          noc_fw_o(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                   *FLIT_DATA_WIDTH-1 downto 0)
            <= data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                      *FLIT_DATA_WIDTH-1 downto 0);
          noc_fw_o(FLIT_WIDTH-3 downto DATA_WIDTH-HDR_ADDR_OFFSET
                   -(flit_no-1)*FLIT_DATA_WIDTH) <= (others => '0');
          noc_fw_o(EOP_BIT_POS) <= '1';  -- eop
          buf_rd <= '1';
          flit_no <= 0;
          vc_res <= VC;
        else
          noc_fw_o(FLIT_WIDTH-3 downto 0)
            <= data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                      *FLIT_DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET
                      -(flit_no_reg)*FLIT_DATA_WIDTH);
          noc_fw_o(EOP_BIT_POS) <= '0';  -- eop
          buf_rd <= '0';
          flit_no <= flit_no_reg +1;
          vc_res <= 1;
        end if;
      end if;
    end if;
```

```
      end if;
    end if;
  end process;



-- purpose: Keeps track of multiple flit packets
-- type   : sequential
-- inputs : clk, rst, flit_no
-- outputs: flit_no_reg
flit_no_counter: process (clk, rst)
begin  -- process flit_no_counter
  if rst = '0' then                     -- asynchronous reset (active low)
    flit_no_reg <= 0;
    vc_res_reg <= VC;
  elsif clk'event and clk = '1' then    -- rising clock edge
    flit_no_reg <= flit_no;
    vc_res_reg <= vc_res;
  end if;
end process flit_no_counter;

end packet_gen;
```

C.1.7 Unpack Unit

```
-------------------------------------------------------------------------------
-- NA unpack - Unpack Unit
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity na_unpack is

  generic (
    DATA_WIDTH : integer);              -- Input data width

  port (
    data_o      : out std_logic_vector(DATA_WIDTH-1 downto 0);
    type_o      : out std_logic;
    vc0_data_i  : in  std_logic_vector(FLIT_WIDTH-1 downto 0);
    vc1_data_i  : in  std_logic_vector(FLIT_WIDTH-1 downto 0);
    vc0_empty_i : in std_logic;
    vc1_empty_i : in std_logic;
    vc0_rd_o    : out std_logic;
    vc1_rd_o    : out std_logic;
    dv          : out std_logic;
    clk         : in  std_logic;
    rst         : in  std_logic);

end na_unpack;

architecture unpack of na_unpack is

  signal flit_no, flit_no_reg, vc_res, vc_res_reg : integer := 0;
  signal packet, packet_reg : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal s_type : std_logic;
  constant last_flit : integer := (DATA_WIDTH+HDR_ADDR_WIDTH) / FLIT_DATA_WIDTH;
```

```
begin  -- packet_gen

    data_o <= packet;
    type_o <= s_type;

process (vc0_data_i, vc1_data_i, vc0_empty_i, vc1_empty_i, packet_reg,
         vc_res_reg, flit_no_reg)
begin  -- process
  if DATA_WIDTH <= HDR_ADDR_OFFSET then
    packet <= vc0_data_i(DATA_WIDTH-1 downto 0 );
    s_type <= vc0_data_i(TYPE_BIT_POS);
    vc0_rd_o <= '0';
    vc1_rd_o <= '0';
    dv <= '0';
    if vc0_empty_i = '0' then
      packet <= vc0_data_i(DATA_WIDTH-1 downto 0 );
      s_type <= vc0_data_i(TYPE_BIT_POS);
      vc0_rd_o <= '1';
      dv <= '1';
    elsif vc1_empty_i = '0' then
      packet <= vc1_data_i(DATA_WIDTH-1 downto 0 );
      s_type <= vc1_data_i(TYPE_BIT_POS);
      vc1_rd_o <= '1';
      dv <= '1';
    end if;
  else
    packet <= vc0_data_i(HDR_ADDR_OFFSET-1 downto 0)
              & packet_reg(DATA_WIDTH-HDR_ADDR_OFFSET-1 downto 0);
    s_type <= vc0_data_i(TYPE_BIT_POS);
    if flit_no_reg = 0 then
      vc0_rd_o <= '0';
      vc1_rd_o <= '0';
      dv <= '0';
      if vc0_empty_i = '0' then
        packet <= vc0_data_i(HDR_ADDR_OFFSET-1 downto 0)
                  & packet_reg(DATA_WIDTH-HDR_ADDR_OFFSET-1 downto 0);
        s_type <= vc0_data_i(TYPE_BIT_POS);
        vc0_rd_o <= '1';
        if vc0_data_i(EOP_BIT_POS) = '1' then
          dv <= '1';
          flit_no <= 0;
          vc_res <= VC;
        else
          dv <= '0';
          flit_no <= 1;
          vc_res <= 0;
        end if;
      elsif vc1_empty_i = '0' then
        packet <= vc1_data_i(HDR_ADDR_OFFSET-1 downto 0)
                  & packet_reg(DATA_WIDTH-HDR_ADDR_OFFSET-1 downto 0);
        s_type <= vc1_data_i(TYPE_BIT_POS);
        vc1_rd_o <= '1';
        if vc1_data_i(EOP_BIT_POS) = '1' then
          dv <= '1';
          flit_no <= 0;
          vc_res <= VC;
        else
          dv <= '0';
          flit_no <= 1;
          vc_res <= 1;
        end if;
      end if;
```

```
    else
      dv <= '0';
      vc0_rd_o <= '0';
      vc1_rd_o <= '0';
      packet <= packet_reg;
      s_type <= vc0_data_i(TYPE_BIT_POS);
      vc_res <= vc_res_reg;
      flit_no <= flit_no_reg;
      if vc_res_reg = 0 and vc0_empty_i = '0' then
        vc0_rd_o <= '1';
        if flit_no_reg = last_flit then
          dv <= '1';
          packet <= packet_reg(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET
                               -(flit_no_reg-1)*(FLIT_DATA_WIDTH))
                    & vc0_data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                                 *(FLIT_DATA_WIDTH)-1 downto 0);
          s_type <= vc0_data_i(TYPE_BIT_POS);
          flit_no <= 0;
          vc_res <= VC;
        else
          dv <= '0';
          packet <= packet_reg(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET
                               -(flit_no_reg-1)*(FLIT_DATA_WIDTH))
                    & vc0_data_i(FLIT_DATA_WIDTH-1 downto 0)
                    & packet_reg(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                                 *(FLIT_DATA_WIDTH)-(FLIT_DATA_WIDTH)-1 downto 0);
          s_type <= vc0_data_i(TYPE_BIT_POS);
          flit_no <= flit_no_reg +1;
          vc_res <= vc_res_reg;
        end if;
      elsif  vc_res_reg = 1 and vc1_empty_i = '0' then
        vc1_rd_o <= '1';
        if flit_no_reg = last_flit then
          dv <= '1';
          packet <= packet_reg(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET
                               -(flit_no_reg-1)*(FLIT_DATA_WIDTH))
                    & vc1_data_i(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                                 *(FLIT_DATA_WIDTH)-1 downto 0);
          s_type <= vc1_data_i(TYPE_BIT_POS);
          flit_no <= 0;
          vc_res <= VC;
        else
          dv <= '0';
          packet <= packet_reg(DATA_WIDTH-1 downto DATA_WIDTH-HDR_ADDR_OFFSET
                               -(flit_no_reg-1)*(FLIT_DATA_WIDTH))
                    & vc1_data_i(FLIT_DATA_WIDTH-1 downto 0)
                    & packet_reg(DATA_WIDTH-HDR_ADDR_OFFSET-(flit_no_reg-1)
                                 *(FLIT_DATA_WIDTH)-(FLIT_DATA_WIDTH)-1 downto 0);
          s_type <= vc1_data_i(TYPE_BIT_POS);
          flit_no <= flit_no_reg +1;
          vc_res <= vc_res_reg;
        end if;
      end if;
    end if;

  end if;
end process;

-- purpose: Keeps track of multiple flit packets
-- type   : sequential
-- inputs : clk, rst, flit_no
-- outputs: flit_no_reg
```

```
flit_no_counter: process (clk, rst)
begin  -- process flit_no_counter
  if rst = '0' then                     -- asynchronous reset (active low)
    flit_no_reg <= 0;
    vc_res_reg <= VC;
    packet_reg <= (others => '0');
  elsif clk'event and clk = '1' then    -- rising clock edge
    flit_no_reg <= flit_no;
    vc_res_reg <= vc_res;
    packet_reg <= packet;
  end if;
end process flit_no_counter;

end unpack;
```

C.1.8 Programming Unit

```
-------------------------------------------------------------------------------
-- PGM Unit - Behavioral programming unit
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use std.textio.all;
use work.types.all;

entity pgm_unit is

  generic (
    filename   : string);

  port (
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic);

end pgm_unit;

architecture pgm of pgm_unit is

  component na_packet_gen
  generic (
    DATA_WIDTH : integer);              -- Input data width
  port (
    data_i    : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    route_i   : in  std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    type_i    : in  std_logic_vector(TYPE_BITS-1 downto 0);
    noc_fw_o  : out noc_link_fw;
    noc_bw_i  : in  noc_link_bw;
    buf_rd    : out std_logic;
    buf_empty : in  std_logic;
    clk       : in  std_logic;
    rst       : in  std_logic);
  end component;

  signal buf_i_empty, buf_i_rd : std_logic;
  signal s_data : std_logic_vector(HDR_ADDR_OFFSET-1 downto 0);
  signal s_route : std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
```

```
  signal s_type_i : std_logic_vector(TYPE_BITS-1 downto 0);

  file input : text open read_mode is filename;


begin  -- na

s_type_i <= "1";

  pckt_gen : na_packet_gen
  generic map(
    DATA_WIDTH => HDR_ADDR_OFFSET)
  port map(
    data_i    => s_data,
    route_i   => s_route,
    type_i    => s_type_i,
    noc_fw_o  => noc_fw_o,
    noc_bw_i  => noc_bw_i,
    buf_rd    => buf_i_rd,
    buf_empty => buf_i_empty,
    clk       => clk,
    rst       => rst);

  process (clk)
    variable l : line;
    variable v : std_logic_vector(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto 0);
    variable w : integer := 0;
  begin  -- process
--    assert endfile(input) report "End of file" severity note;
    if rst = '1' and clk = '1' then
      if buf_i_rd = '1' then
        w := 0;
        buf_i_empty <= '1';
      end if;
      if w = 0  and not endfile(input) then
        readline(input, l);
        read(l,v);
        buf_i_empty <= '0';
        w := 1;
      end if;
        s_data <= v(HDR_ADDR_OFFSET-1 downto 0);
        s_route <= v(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1 downto HDR_ADDR_OFFSET);
    end if;
    if rst = '0' then
      buf_i_empty <= '1';
    end if;
  end process;


end pgm;
```

C.1.9 Network Adapter

```
--------------------------------------------------------------------------------
-- Mesh NA - Network Adapter
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;
```

```
entity mesh_na is

  generic (
    ADDR_BITS    : integer := 4;
    DATA_WIDTH_I : integer := 32;
    DATA_WIDTH_O : integer := 32);

  port (
    noc_fw_i : in  noc_link_fw;
    noc_bw_o : out noc_link_bw;
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic;
    data_i   : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o   : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_i   : in  std_logic_vector(ADDR_BITS-1 downto 0);
    type_i   : in  std_logic_vector(TYPE_BITS-1 downto 0);
    req_i    : in  std_logic;
    req_o    : out std_logic;
    ack_i    : in  std_logic;
    ack_o    : out std_logic);

end mesh_na;

architecture na of mesh_na is

  component data_buffer
  generic (
    DATA_WIDTH : integer := 32;
    ADDR_BITS : integer := 4);
  port (
    data_i : in  std_logic_vector(ADDR_BITS + DATA_WIDTH-1 downto 0);
    data_o : out std_logic_vector(ADDR_BITS + DATA_WIDTH-1 downto 0);
    clk    : in  std_logic;                                 -- Clock
    rst    : in  std_logic;                                 -- Reset
    rd     : in  std_logic;                                 -- Read
    wr     : in  std_logic;                                 -- Write
    full   : out std_logic;                                 -- Full
    empty  : out std_logic);                                -- Empty

  end component;

  component na_address_tbl
  generic (
    ADDR_BITS : integer := 4);            -- Number of entries

  port (
    addr_i : in std_logic_vector(ADDR_BITS-1 downto 0);
    addr_rd : in std_logic_vector(ADDR_BITS-1 downto 0);
    dat_i : in std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    dat_o : out std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    rw : in std_logic;                  -- Read/write
    clk : in std_logic;
    rst : in std_logic);
  end component;

  component na_packet_gen
  generic (
    DATA_WIDTH : integer);              -- Input data width
  port (
```

```
    data_i    : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    route_i   : in  std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
    type_i    : in  std_logic_vector(TYPE_BITS-1 downto 0);
    noc_fw_o  : out noc_link_fw;
    noc_bw_i  : in  noc_link_bw;
    buf_rd    : out std_logic;
    buf_empty : in  std_logic;
    clk       : in  std_logic;
    rst       : in  std_logic);
  end component;

  component na_unpack
  generic (
    DATA_WIDTH : integer);              -- Input data width
  port (
    data_o      : out std_logic_vector(DATA_WIDTH-1 downto 0);
    type_o      : out std_logic;
    vc0_data_i  : in  std_logic_vector(FLIT_WIDTH-1 downto 0);
    vc1_data_i  : in  std_logic_vector(FLIT_WIDTH-1 downto 0);
    vc0_empty_i : in std_logic;
    vc1_empty_i : in std_logic;
    vc0_rd_o    : out std_logic;
    vc1_rd_o    : out std_logic;
    dv          : out std_logic;
    clk         : in  std_logic;
    rst         : in  std_logic);
  end component;

  component vc_buffer
  port (
    data_i : in  std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data input
    data_o : out std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data output
    clk    : in  std_logic;                                -- Clock
    rst    : in  std_logic;                                -- Reset
    rd     : in  std_logic;                                -- Read
    wr     : in  std_logic;                                -- Write
    full   : out std_logic;                                -- Full
    empty  : out std_logic);                               -- Empty
  end component;

  signal buf_i_empty, buf_i_full, buf_i_wr, buf_i_rd, pgm_wr, vc0_wr, vc0_full,
    vc0_empty, vc0_rd, vc1_wr, vc1_full, vc1_empty, vc1_rd, s_type, s_req_o
    : std_logic;
  signal vc0_data, vc1_data : std_logic_vector(FLIT_WIDTH-1 downto 0);
  signal s_data : std_logic_vector(DATA_WIDTH_I-1 downto 0);
  signal s_buf_input, s_buf_output : std_logic_vector(TYPE_BITS + DATA_WIDTH_I
                                                      + ADDR_BITS -1 downto 0);
  signal pgm_addr, s_addr : std_logic_vector(ADDR_BITS-1 downto 0);
  signal pgm_route, s_route : std_logic_vector(HDR_ADDR_WIDTH-1 downto 0);
  signal s_data_o : std_logic_vector(DATA_WIDTH_O-1 downto 0);
  signal s_type_i : std_logic_vector(TYPE_BITS-1 downto 0);

begin  -- na
  s_buf_input <= type_i & addr_i & data_i;
  s_addr <= s_buf_output(ADDR_BITS+DATA_WIDTH_I-1 downto DATA_WIDTH_I);
  s_data <= s_buf_output(DATA_WIDTH_I-1 downto 0);
  s_type_i <= s_buf_output(TYPE_BITS+ADDR_BITS+DATA_WIDTH_I-1 downto
                           ADDR_BITS+DATA_WIDTH_I);
  ack_o <= req_i and buf_i_empty;
  buf_i_wr <= req_i and not buf_i_full;
  data_o <= s_data_o;
```

```
  buf_i : data_buffer
  generic map (
    DATA_WIDTH =>  DATA_WIDTH_I,
    ADDR_BITS => ADDR_BITS+TYPE_BITS)
  port map(
    data_i => s_buf_input,
    data_o => s_buf_output,
    clk    => clk,
    rst    => rst,
    rd     => buf_i_rd,
    wr     => buf_i_wr,
    full   => buf_i_full,
    empty  => buf_i_empty);

  addr_tbl : na_address_tbl
  generic map (
    ADDR_BITS => ADDR_BITS)
  port map (
    addr_i => pgm_addr,
    addr_rd => s_addr,
    dat_i => pgm_route,
    dat_o => s_route,
    rw => pgm_wr,
    clk => clk,
    rst => rst);

  pckt_gen : na_packet_gen
  generic map(
    DATA_WIDTH => DATA_WIDTH_I)
  port map(
    data_i    => s_data,
    route_i   => s_route,
    type_i    => s_type_i,
    noc_fw_o  => noc_fw_o,
    noc_bw_i  => noc_bw_i,
    buf_rd    => buf_i_rd,
    buf_empty => buf_i_empty,
    clk       => clk,
    rst       => rst);

  buf_vc0 : vc_buffer
    port map (
      data_i => noc_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => vc0_data,
      clk    => clk,
      rst    => rst,
      rd     => vc0_rd,
      wr     => vc0_wr,
      full   => vc0_full,
      empty  => vc0_empty);

vc0_wr <= '1' when (noc_fw_i(DV_bit) = '1'
                    and noc_fw_i(FLIT_WIDTH-1 + VC_bit
                                 downto FLIT_WIDTH) = "0") else '0';

    buf_vc1 : vc_buffer
    port map (
      data_i => noc_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => vc1_data,
      clk    => clk,
      rst    => rst,
      rd     => vc1_rd,
```

```
      wr     => vc1_wr,
      full   => vc1_full,
      empty  => vc1_empty);

vc1_wr <= '1' when (noc_fw_i(DV_bit) = '1'
                    and noc_fw_i(FLIT_WIDTH-1 + VC_bit
                                 downto FLIT_WIDTH) = "1") else '0';

noc_bw_o <= not vc1_full & not vc0_full;

  unpack : na_unpack
  generic map(
    DATA_WIDTH => DATA_WIDTH_O)              -- Input data width
  port map(
    data_o      => s_data_o,
    type_o      => s_type,
    vc0_data_i  => vc0_data,
    vc1_data_i  => vc1_data,
    vc0_empty_i => vc0_empty,
    vc1_empty_i => vc1_empty,
    vc0_rd_o    => vc0_rd,
    vc1_rd_o    => vc1_rd,
    dv          => s_req_o,
    clk         => clk,
    rst         => rst);

  req_o <= s_req_o and not s_type;
  pgm_addr <= s_data_o(DATA_WIDTH_O-1 downto DATA_WIDTH_O-ADDR_BITS);
  pgm_route <= s_data_o(DATA_WIDTH_O-ADDR_BITS-1
                        downto DATA_WIDTH_O-ADDR_BITS-HDR_ADDR_WIDTH);
  pgm_wr <= s_req_o and s_type;
end na;
```

C.1.10 Routing Node

```
--------------------------------------------------------------------------------
-- Mesh Router - Routing Node
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity mesh_router is

  port (
    clk         : in  std_logic;        -- Clock
    rst         : in  std_logic;        -- Reset
    -- Input links
    link_w_fw_i : in  noc_link_fw;      -- Input link west data
    link_w_bw_o : out noc_link_bw;      -- Input link west ack
    link_n_fw_i : in  noc_link_fw;      -- Input link north data
    link_n_bw_o : out noc_link_bw;      -- Input link north ack
    link_e_fw_i : in  noc_link_fw;      -- Input link east data
    link_e_bw_o : out noc_link_bw;      -- Input link east ack
    link_s_fw_i : in  noc_link_fw;      -- Input link south data
    link_s_bw_o : out noc_link_bw;      -- Input link south ack
    link_a_fw_i : in  noc_link_fw;      -- Input link adapter data
    link_a_bw_o : out noc_link_bw;      -- Input link adapter ack
    -- Output links
```

```
    link_w_fw_o : out noc_link_fw;      -- Input link west data
    link_w_bw_i : in  noc_link_bw;      -- Input link west ack
    link_n_fw_o : out noc_link_fw;      -- Input link north data
    link_n_bw_i : in  noc_link_bw;      -- Input link north ack
    link_e_fw_o : out noc_link_fw;      -- Input link east data
    link_e_bw_i : in  noc_link_bw;      -- Input link east ack
    link_s_fw_o : out noc_link_fw;      -- Input link south data
    link_s_bw_i : in  noc_link_bw;      -- Input link south ack
    link_a_fw_o : out noc_link_fw;      -- Input link adapter data
    link_a_bw_i : in  noc_link_bw);     -- Input link adapter ack

end mesh_router;

architecture mesh_router of mesh_router is

  component vc_buffer
    port (
    data_i : in  std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data input
    data_o : out std_logic_vector(FLIT_WIDTH-1 downto 0);  -- Data output
    clk    : in  std_logic;                                -- Clock
    rst    : in  std_logic;                                -- Reset
    rd     : in  std_logic;                                -- Read
    wr     : in  std_logic;                                -- Write
    full   : out std_logic;                                -- Full
    empty  : out std_logic);                               -- Empty
  end component;

  component be_router
    generic (
      DIR : rt_direction);        -- Routing direction
    port (
      clk      : in  std_logic;         -- Clock
      rst      : in  std_logic;         -- Reset
      src_0    : in  std_logic_vector(4 downto 0);           -- Source 0
      src_1    : in  std_logic_vector(4 downto 0);           -- Source 1
      src_2    : in  std_logic_vector(4 downto 0);           -- Source 2
      src_3    : in  std_logic_vector(4 downto 0);           -- Source 3
      src_4    : in  std_logic_vector(4 downto 0);           -- Source 4
      src_5    : in  std_logic_vector(4 downto 0);           -- Source 5
      src_6    : in  std_logic_vector(4 downto 0);           -- Source 6
      src_7    : in  std_logic_vector(4 downto 0);           -- Source 7
      vc_empty : in  std_logic_vector(VC-1 downto 0);   -- Empty output VC's
      vc_sel   : out std_logic_vector(0 downto 0);  -- VC select
      src_sel  : out std_logic_vector((4*VC)-1 downto 0));  -- Source select/read
  end component;


  component flit_handler
  generic (
    INPUT : rt_direction);
  port (
    clk         : in  std_logic;        -- Clock
    rst         : in  std_logic;        -- Reset
    eop         : in  std_logic;        -- End of packet
    empty       : in  std_logic;        -- Data valid
    rd          : in  std_logic;        -- Read buffer (First flit detection)
    dir_i       : in  std_logic_vector(HOP_BITS-1 downto 0);
    dir_o       : out std_logic_vector(HOP_BITS downto 0);
    rot         : out std_logic);       -- Rotate header (Indicates header flit)

  end component;
```

```
  signal w_vc0_wr, w_vc1_wr, n_vc0_wr, n_vc1_wr, e_vc0_wr, e_vc1_wr, s_vc0_wr,
    s_vc1_wr, a_vc0_wr, a_vc1_wr : std_logic;           -- Write VC buffer 0
  signal w_vc0_rd, w_vc1_rd, n_vc0_rd, n_vc1_rd, e_vc0_rd, e_vc1_rd, s_vc0_rd,
    s_vc1_rd, a_vc0_rd, a_vc1_rd : std_logic;          -- Read VC buffer 0
  signal src_sel_w, src_sel_n, src_sel_e, src_sel_s, src_sel_a
    : std_logic_vector((4*VC)-1 downto 0);  -- Source select
  signal link_w_vc0, link_w_vc1, link_n_vc0, link_n_vc1, link_e_vc0,
    link_e_vc1, link_s_vc0, link_s_vc1, link_a_vc0, link_a_vc1
    : vc_data;          -- VC data
  signal link_w_vc0_rot, link_w_vc1_rot, link_n_vc0_rot, link_n_vc1_rot,
    link_e_vc0_rot, link_e_vc1_rot, link_s_vc0_rot, link_s_vc1_rot,
    link_a_vc0_rot, link_a_vc1_rot : vc_data;
  signal link_w_vc0_dir, link_w_vc1_dir, link_n_vc0_dir, link_n_vc1_dir,
    link_e_vc0_dir, link_e_vc1_dir, link_s_vc0_dir, link_s_vc1_dir,
    link_a_vc0_dir, link_a_vc1_dir
    : rt_direction;   -- Translated routing direction
  signal link_w_vc0_empty, link_w_vc1_empty, link_n_vc0_empty,
    link_n_vc1_empty, link_e_vc0_empty, link_e_vc1_empty, link_s_vc0_empty,
    link_s_vc1_empty, link_a_vc0_empty, link_a_vc1_empty : std_logic;
  signal link_w_vc0_b_empty, link_w_vc1_b_empty, link_n_vc0_b_empty,
    link_n_vc1_b_empty, link_e_vc0_b_empty, link_e_vc1_b_empty,
    link_s_vc0_b_empty, link_s_vc1_b_empty, link_a_vc0_b_empty,
    link_a_vc1_b_empty : std_logic;
  signal link_w_vc0_full, link_w_vc1_full, link_n_vc0_full, link_n_vc1_full,
    link_e_vc0_full, link_e_vc1_full, link_s_vc0_full, link_s_vc1_full,
    link_a_vc0_full, link_a_vc1_full
    : std_logic;   -- Translated routing direction
  signal rot_w_vc0, rot_w_vc1, rot_n_vc0, rot_n_vc1, rot_e_vc0, rot_e_vc1,
    rot_s_vc0, rot_s_vc1, rot_a_vc0, rot_a_vc1 : std_logic;   -- Rotate header
  signal w_vc_old, s_vc_old, e_vc_old, n_vc_old, a_vc_old : std_logic;

begin  -- mesh_router

  -- West
  buf_w_vc0 : vc_buffer
    port map (
      data_i => link_w_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_w_vc0,
      clk    => clk,
      rst    => rst,
      rd     => w_vc0_rd,
      wr     => w_vc0_wr,
      full   => link_w_vc0_full,
      empty  => link_w_vc0_b_empty);

w_vc0_wr <= '1' when (link_w_fw_i(DV_bit) = '1'
                      and link_w_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "0") else '0';
w_vc0_rd <= (src_sel_a(0) or src_sel_n(0) or src_sel_e(0) or src_sel_s(0));
link_w_vc0_rot <= link_w_vc0 when rot_w_vc0 = '0' else link_w_vc0(EOP_BIT_POS)
                  & link_w_vc0(TYPE_BIT_POS)
                  & link_w_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_w_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_w_vc0(HDR_ADDR_OFFSET-1 downto 0);

  fh_w_vc0 : flit_handler
    generic map(
      INPUT => rt_w)
    port map(
      clk   => clk,
```

```
      rst   => rst,
      eop   => link_w_vc0(EOP_BIT_POS),
      empty => link_w_vc0_b_empty,
      rd    => w_vc0_rd,
      dir_i => link_w_vc0(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_w_vc0_dir,
      rot   => rot_w_vc0);       -- Rotate header (Indicates header flit)

    buf_w_vc1 : vc_buffer
    port map (
      data_i => link_w_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_w_vc1,
      clk    => clk,
      rst    => rst,
      rd     => w_vc1_rd,
      wr     => w_vc1_wr,
      full   => link_w_vc1_full,
      empty  => link_w_vc1_b_empty);

w_vc1_wr <= '1' when (link_w_fw_i(DV_bit) = '1'
                      and link_w_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "1") else '0';
w_vc1_rd <= (src_sel_a(4) or src_sel_n(4) or src_sel_e(4) or src_sel_s(4));
link_w_vc1_rot <= link_w_vc1 when rot_w_vc1 = '0' else link_w_vc1(EOP_BIT_POS)
                  & link_w_vc1(TYPE_BIT_POS)
                  & link_w_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_w_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_w_vc1(HDR_ADDR_OFFSET-1 downto 0);

    fh_w_vc1 : flit_handler
    generic map(
      INPUT => rt_w)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_w_vc1(FLIT_WIDTH-1),
      empty => link_w_vc1_b_empty,
      rd    => w_vc1_rd,
      dir_i => link_w_vc1(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_w_vc1_dir,
      rot   => rot_w_vc1);       -- Rotate header (Indicates header flit)

-- Determine VC order.
process (clk, rst)
begin  -- process
  if rst = '0' then                    -- asynchronous reset (active low)
    w_vc_old <= '0';
  elsif clk'event and clk = '1' then    -- rising clock edge
    if (link_w_vc0_b_empty xor link_w_vc1_b_empty) = '1' then
      w_vc_old <= link_w_vc0_b_empty;
    end if;
  end if;
end process;

link_w_vc0_empty <= '1' when ((link_w_vc1_dir = link_w_vc0_dir)
                              and w_vc_old = '1')
                    else link_w_vc0_b_empty;
link_w_vc1_empty <= '1' when ((link_w_vc1_dir = link_w_vc0_dir)
```

```
                            and w_vc_old = '0')
                    else link_w_vc1_b_empty;

  router_w : be_router
    generic map (
      DIR => rt_w)
    port map (
      clk         => clk,
      rst         => rst,
      src_0(2 downto 0)  => link_a_vc0_dir,
      src_0(3)    => link_a_vc0(EOP_BIT_POS),
      src_0(4)    => link_a_vc0_empty,
      src_1(2 downto 0)  => link_n_vc0_dir,
      src_1(3)    => link_n_vc0(EOP_BIT_POS),
      src_1(4)    => link_n_vc0_empty,
      src_2(2 downto 0)  => link_e_vc0_dir,
      src_2(3)    => link_e_vc0(EOP_BIT_POS),
      src_2(4)    => link_e_vc0_empty,
      src_3(2 downto 0)  => link_s_vc0_dir,
      src_3(3)    => link_s_vc0(EOP_BIT_POS),
      src_3(4)    => link_s_vc0_empty,
      src_4(2 downto 0)  => link_a_vc1_dir,
      src_4(3)    => link_a_vc1(EOP_BIT_POS),
      src_4(4)    => link_a_vc1_empty,
      src_5(2 downto 0)  => link_n_vc1_dir,
      src_5(3)    => link_n_vc1(EOP_BIT_POS),
      src_5(4)    => link_n_vc1_empty,
      src_6(2 downto 0)  => link_e_vc1_dir,
      src_6(3)    => link_e_vc1(EOP_BIT_POS),
      src_6(4)    => link_e_vc1_empty,
      src_7(2 downto 0)  => link_s_vc1_dir,
      src_7(3)    => link_s_vc1(EOP_BIT_POS),
      src_7(4)    => link_s_vc1_empty,
      vc_empty => link_w_bw_i,
      vc_sel   => link_w_fw_o(FLIT_WIDTH-1+VC_bit downto FLIT_WIDTH),
      src_sel  => src_sel_w);

link_w_bw_o <= not link_w_vc1_full & not link_w_vc0_full;
link_w_fw_o(FLIT_WIDTH-1 downto 0) <= link_a_vc0_rot when src_sel_w = X"01" else
                                      link_n_vc0_rot when src_sel_w = X"02" else
                                      link_e_vc0_rot when src_sel_w = X"04" else
                                      link_s_vc0_rot when src_sel_w = X"08" else
                                      link_a_vc1_rot when src_sel_w = X"10" else
                                      link_n_vc1_rot when src_sel_w = X"20" else
                                      link_e_vc1_rot when src_sel_w = X"40"
                                      else link_s_vc1_rot;
link_w_fw_o(FLIT_WIDTH+VC_bit) <= src_sel_w(0) or src_sel_w(1) or src_sel_w(2)
                                  or src_sel_w(3) or src_sel_w(4)
                                  or src_sel_w(5) or src_sel_w(6)
                                  or src_sel_w(7);

  -- North
  buf_n_vc0 : vc_buffer
    port map (
      data_i => link_n_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_n_vc0,
      clk    => clk,
      rst    => rst,
      rd     => n_vc0_rd,
      wr     => n_vc0_wr,
      full   => link_n_vc0_full,
      empty  => link_n_vc0_b_empty);
```

```
n_vc0_wr <= '1' when (link_n_fw_i(DV_bit) = '1'
                      and link_n_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "0") else '0';
n_vc0_rd <= (src_sel_a(1) or src_sel_w(1) or src_sel_e(1) or src_sel_s(1));
link_n_vc0_rot <= link_n_vc0 when rot_n_vc0 = '0' else link_n_vc0(EOP_BIT_POS)
                  & link_n_vc0(TYPE_BIT_POS)
                  & link_n_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_n_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_n_vc0(HDR_ADDR_OFFSET-1 downto 0);

    fh_n_vc0 : flit_handler
    generic map(
      INPUT => rt_n)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_n_vc0(EOP_BIT_POS),
      empty => link_n_vc0_b_empty,
      rd    => n_vc0_rd,
      dir_i => link_n_vc0(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_n_vc0_dir,
      rot   => rot_n_vc0);       -- Rotate header (Indicates header flit)

    buf_n_vc1 : vc_buffer
    port map (
      data_i => link_n_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_n_vc1,
      clk    => clk,
      rst    => rst,
      rd     => n_vc1_rd,
      wr     => n_vc1_wr,
      full   => link_n_vc1_full,
      empty  => link_n_vc1_b_empty);

n_vc1_wr <= '1' when (link_n_fw_i(DV_bit) = '1'
                      and link_n_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "1") else '0';
n_vc1_rd <= (src_sel_a(5) or src_sel_w(5) or src_sel_e(5) or src_sel_s(5));
link_n_vc1_rot <= link_n_vc1 when rot_n_vc1 = '0' else link_n_vc1(EOP_BIT_POS)
                  & link_n_vc1(TYPE_BIT_POS)
                  & link_n_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_n_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_n_vc1(HDR_ADDR_OFFSET-1 downto 0);

    fh_n_vc1 : flit_handler
    generic map(
      INPUT => rt_n)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_n_vc1(EOP_BIT_POS),
      empty => link_n_vc1_b_empty,
      rd    => n_vc1_rd,
      dir_i => link_n_vc1(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_n_vc1_dir,
```

```
      rot   => rot_n_vc1);       -- Rotate header (Indicates header flit)

-- Determine VC order.
process (clk, rst)
begin  -- process
  if rst = '0' then                     -- asynchronous reset (active low)
    n_vc_old <= '0';
  elsif clk'event and clk = '1' then    -- rising clock edge
    if (link_n_vc0_b_empty xor link_n_vc1_b_empty) = '1' then
      n_vc_old <= link_n_vc0_b_empty;
    end if;
  end if;
end process;

link_n_vc0_empty <= '1' when ((link_n_vc1_dir = link_n_vc0_dir)
                              and n_vc_old = '1')
                    else link_n_vc0_b_empty;
link_n_vc1_empty <= '1' when ((link_n_vc1_dir = link_n_vc0_dir)
                              and n_vc_old = '0')
                    else link_n_vc1_b_empty;

  router_n : be_router
    generic map (
      DIR => rt_n)
    port map (
      clk         => clk,
      rst         => rst,
      src_0(2 downto 0)    => link_w_vc0_dir,
      src_0(3)    => link_w_vc0(EOP_BIT_POS),
      src_0(4)    => link_w_vc0_empty,
      src_1(2 downto 0)    => link_a_vc0_dir,
      src_1(3)    => link_a_vc0(EOP_BIT_POS),
      src_1(4)    => link_a_vc0_empty,
      src_2(2 downto 0)    => link_e_vc0_dir,
      src_2(3)    => link_e_vc0(EOP_BIT_POS),
      src_2(4)    => link_e_vc0_empty,
      src_3(2 downto 0)    => link_s_vc0_dir,
      src_3(3)    => link_s_vc0(EOP_BIT_POS),
      src_3(4)    => link_s_vc0_empty,
      src_4(2 downto 0)    => link_w_vc1_dir,
      src_4(3)    => link_w_vc1(EOP_BIT_POS),
      src_4(4)    => link_w_vc1_empty,
      src_5(2 downto 0)    => link_a_vc1_dir,
      src_5(3)    => link_a_vc1(EOP_BIT_POS),
      src_5(4)    => link_a_vc1_empty,
      src_6(2 downto 0)    => link_e_vc1_dir,
      src_6(3)    => link_e_vc1(EOP_BIT_POS),
      src_6(4)    => link_e_vc1_empty,
      src_7(2 downto 0)    => link_s_vc1_dir,
      src_7(3)    => link_s_vc1(EOP_BIT_POS),
      src_7(4)    => link_s_vc1_empty,
      vc_empty => link_n_bw_i,
      vc_sel   => link_n_fw_o(FLIT_WIDTH-1+VC_bit downto FLIT_WIDTH),
      src_sel  => src_sel_n);

link_n_bw_o <= not link_n_vc1_full & not link_n_vc0_full;
link_n_fw_o(FLIT_WIDTH-1 downto 0) <= link_w_vc0_rot when src_sel_n = X"01" else
                                      link_a_vc0_rot when src_sel_n = X"02" else
                                      link_e_vc0_rot when src_sel_n = X"04" else
                                      link_s_vc0_rot when src_sel_n = X"08" else
                                      link_w_vc1_rot when src_sel_n = X"10" else
                                      link_a_vc1_rot when src_sel_n = X"20" else
```

```
                                    link_e_vc1_rot when src_sel_n = X"40"
                                    else link_s_vc1_rot;
link_n_fw_o(FLIT_WIDTH+VC_bit) <= src_sel_n(0) or src_sel_n(1) or src_sel_n(2)
                                  or src_sel_n(3) or src_sel_n(4)
                                  or src_sel_n(5) or src_sel_n(6)
                                  or src_sel_n(7);

  -- East
  buf_e_vc0 : vc_buffer
    port map (
      data_i => link_e_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_e_vc0,
      clk    => clk,
      rst    => rst,
      rd     => e_vc0_rd,
      wr     => e_vc0_wr,
      full   => link_e_vc0_full,
      empty  => link_e_vc0_b_empty);

e_vc0_wr <= '1' when (link_e_fw_i(DV_bit) = '1'
                      and link_e_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "0") else '0';
e_vc0_rd <= (src_sel_a(2) or src_sel_n(2) or src_sel_w(2) or src_sel_s(2));
link_e_vc0_rot <= link_e_vc0 when rot_e_vc0 = '0' else link_e_vc0(EOP_BIT_POS)
                  & link_e_vc0(TYPE_BIT_POS)
                  & link_e_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_e_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_e_vc0(HDR_ADDR_OFFSET-1 downto 0);

    fh_e_vc0 : flit_handler
    generic map(
      INPUT => rt_e)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_e_vc0(EOP_BIT_POS),
      empty => link_e_vc0_b_empty,
      rd    => e_vc0_rd,
      dir_i => link_e_vc0(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_e_vc0_dir,
      rot   => rot_e_vc0);       -- Rotate header (Indicates header flit)

    buf_e_vc1 : vc_buffer
    port map (
      data_i => link_e_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_e_vc1,
      clk    => clk,
      rst    => rst,
      rd     => e_vc1_rd,
      wr     => e_vc1_wr,
      full   => link_e_vc1_full,
      empty  => link_e_vc1_b_empty);

e_vc1_wr <= '1' when (link_e_fw_i(DV_bit) = '1'
                      and link_e_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "1") else '0';
e_vc1_rd <= (src_sel_a(6) or src_sel_n(6) or src_sel_w(6) or src_sel_s(6));
link_e_vc1_rot <= link_e_vc1 when rot_e_vc1 = '0' else link_e_vc1(EOP_BIT_POS)
                  & link_e_vc1(TYPE_BIT_POS)
```

```
                & link_e_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                             downto HDR_ADDR_OFFSET)
                & link_e_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                             downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                & link_e_vc1(HDR_ADDR_OFFSET-1 downto 0);

    fh_e_vc1 : flit_handler
    generic map(
      INPUT => rt_e)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_e_vc1(EOP_BIT_POS),
      empty => link_e_vc1_b_empty,
      rd    => e_vc1_rd,
      dir_i => link_e_vc1(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_e_vc1_dir,
      rot   => rot_e_vc1);       -- Rotate header (Indicates header flit)

-- Determine VC order.
process (clk, rst)
begin  -- process
  if rst = '0' then                     -- asynchronous reset (active low)
    e_vc_old <= '0';
  elsif clk'event and clk = '1' then    -- rising clock edge
    if (link_e_vc0_b_empty xor link_e_vc1_b_empty) = '1' then
      e_vc_old <= link_e_vc0_b_empty;
    end if;
  end if;
end process;

link_e_vc0_empty <= '1' when ((link_e_vc1_dir = link_e_vc0_dir)
                              and e_vc_old = '1')
                    else link_e_vc0_b_empty;
link_e_vc1_empty <= '1' when ((link_e_vc1_dir = link_e_vc0_dir)
                              and e_vc_old = '0')
                    else link_e_vc1_b_empty;

  router_e : be_router
    generic map (
      DIR => rt_e)
    port map (
      clk         => clk,
      rst         => rst,
      src_0(2 downto 0)    => link_w_vc0_dir,
      src_0(3)    => link_w_vc0(EOP_BIT_POS),
      src_0(4)    => link_w_vc0_empty,
      src_1(2 downto 0)    => link_n_vc0_dir,
      src_1(3)    => link_n_vc0(EOP_BIT_POS),
      src_1(4)    => link_n_vc0_empty,
      src_2(2 downto 0)    => link_a_vc0_dir,
      src_2(3)    => link_a_vc0(EOP_BIT_POS),
      src_2(4)    => link_a_vc0_empty,
      src_3(2 downto 0)    => link_s_vc0_dir,
      src_3(3)    => link_s_vc0(EOP_BIT_POS),
      src_3(4)    => link_s_vc0_empty,
      src_4(2 downto 0)    => link_w_vc1_dir,
      src_4(3)    => link_w_vc1(EOP_BIT_POS),
      src_4(4)    => link_w_vc1_empty,
      src_5(2 downto 0)    => link_n_vc1_dir,
      src_5(3)    => link_n_vc1(EOP_BIT_POS),
```

```
      src_5(4)    => link_n_vc1_empty,
      src_6(2 downto 0)    => link_a_vc1_dir,
      src_6(3)    => link_a_vc1(EOP_BIT_POS),
      src_6(4)    => link_a_vc1_empty,
      src_7(2 downto 0)    => link_s_vc1_dir,
      src_7(3)    => link_s_vc1(EOP_BIT_POS),
      src_7(4)    => link_s_vc1_empty,
      vc_empty => link_e_bw_i,
      vc_sel   => link_e_fw_o(FLIT_WIDTH-1+VC_bit downto FLIT_WIDTH),
      src_sel  => src_sel_e);

link_e_bw_o <= not link_e_vc1_full & not link_e_vc0_full;
link_e_fw_o(FLIT_WIDTH-1 downto 0) <= link_w_vc0_rot when src_sel_e = X"01" else
                                      link_n_vc0_rot when src_sel_e = X"02" else
                                      link_a_vc0_rot when src_sel_e = X"04" else
                                      link_s_vc0_rot when src_sel_e = X"08" else
                                      link_w_vc1_rot when src_sel_e = X"10" else
                                      link_n_vc1_rot when src_sel_e = X"20" else
                                      link_a_vc1_rot when src_sel_e = X"40"
                                      else link_s_vc1_rot;
link_e_fw_o(FLIT_WIDTH+VC_bit) <= src_sel_e(0) or src_sel_e(1) or src_sel_e(2)
                                  or src_sel_e(3) or src_sel_e(4)
                                  or src_sel_e(5) or src_sel_e(6)
                                  or src_sel_e(7);

  -- South
  buf_s_vc0 : vc_buffer
    port map (
      data_i => link_s_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_s_vc0,
      clk    => clk,
      rst    => rst,
      rd     => s_vc0_rd,
      wr     => s_vc0_wr,
      full  => link_s_vc0_full,
      empty  => link_s_vc0_b_empty);

s_vc0_wr <= '1' when (link_s_fw_i(DV_bit) = '1'
                      and link_s_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "0") else '0';
s_vc0_rd <= (src_sel_a(3) or src_sel_n(3) or src_sel_e(3) or src_sel_w(3));
link_s_vc0_rot <= link_s_vc0 when rot_s_vc0 = '0' else link_s_vc0(EOP_BIT_POS)
                  & link_s_vc0(TYPE_BIT_POS)
                  & link_s_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_s_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_s_vc0(HDR_ADDR_OFFSET-1 downto 0);


    fh_s_vc0 : flit_handler
    generic map(
      INPUT => rt_s)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_s_vc0(EOP_BIT_POS),
      empty => link_s_vc0_b_empty,
      rd    => s_vc0_rd,
      dir_i => link_s_vc0(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_s_vc0_dir,
```

```
      rot   => rot_s_vc0);       -- Rotate header (Indicates header flit)

    buf_s_vc1 : vc_buffer
    port map (
      data_i => link_s_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_s_vc1,
      clk    => clk,
      rst    => rst,
      rd     => s_vc1_rd,
      wr     => s_vc1_wr,
      full   => link_s_vc1_full,
      empty  => link_s_vc1_b_empty);

s_vc1_wr <= '1' when (link_s_fw_i(DV_bit) = '1'
                      and link_s_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "1") else '0';
s_vc1_rd <= (src_sel_a(7) or src_sel_n(7) or src_sel_e(7) or src_sel_w(7));
link_s_vc1_rot <= link_s_vc1 when rot_s_vc1 = '0' else link_s_vc1(EOP_BIT_POS)
                  & link_s_vc1(TYPE_BIT_POS)
                  & link_s_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                               downto HDR_ADDR_OFFSET)
                  & link_s_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                               downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                  & link_s_vc1(HDR_ADDR_OFFSET-1 downto 0);

    fh_s_vc1 : flit_handler
    generic map(
      INPUT => rt_s)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_s_vc1(EOP_BIT_POS),
      empty => link_s_vc1_b_empty,
      rd    => s_vc1_rd,
      dir_i => link_s_vc1(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_s_vc1_dir,
      rot   => rot_s_vc1);       -- Rotate header (Indicates header flit)

-- Determine VC order.
process (clk, rst)
begin  -- process
  if rst = '0' then                     -- asynchronous reset (active low)
    s_vc_old <= '0';
  elsif clk'event and clk = '1' then    -- rising clock edge
    if (link_s_vc0_b_empty xor link_s_vc1_b_empty) = '1' then
      s_vc_old <= link_s_vc0_b_empty;
    end if;
  end if;
end process;

link_s_vc0_empty <= '1' when ((link_s_vc1_dir = link_s_vc0_dir)
                              and s_vc_old = '1')
                    else link_s_vc0_b_empty;
link_s_vc1_empty <= '1' when ((link_s_vc1_dir = link_s_vc0_dir)
                              and s_vc_old = '0')
                    else link_s_vc1_b_empty;

  router_s : be_router
    generic map (
      DIR => rt_s)
    port map (
```

```
      clk         => clk,
      rst         => rst,
      src_0(2 downto 0)    => link_w_vc0_dir,
      src_0(3)    => link_w_vc0(EOP_BIT_POS),
      src_0(4)    => link_w_vc0_empty,
      src_1(2 downto 0)    => link_n_vc0_dir,
      src_1(3)    => link_n_vc0(EOP_BIT_POS),
      src_1(4)    => link_n_vc0_empty,
      src_2(2 downto 0)    => link_e_vc0_dir,
      src_2(3)    => link_e_vc0(EOP_BIT_POS),
      src_2(4)    => link_e_vc0_empty,
      src_3(2 downto 0)    => link_a_vc0_dir,
      src_3(3)    => link_a_vc0(EOP_BIT_POS),
      src_3(4)    => link_a_vc0_empty,
      src_4(2 downto 0)    => link_w_vc1_dir,
      src_4(3)    => link_w_vc1(EOP_BIT_POS),
      src_4(4)    => link_w_vc1_empty,
      src_5(2 downto 0)    => link_n_vc1_dir,
      src_5(3)    => link_n_vc1(EOP_BIT_POS),
      src_5(4)    => link_n_vc1_empty,
      src_6(2 downto 0)    => link_e_vc1_dir,
      src_6(3)    => link_e_vc1(EOP_BIT_POS),
      src_6(4)    => link_e_vc1_empty,
      src_7(2 downto 0)    => link_a_vc1_dir,
      src_7(3)    => link_a_vc1(EOP_BIT_POS),
      src_7(4)    => link_a_vc1_empty,
      vc_empty => link_s_bw_i,
      vc_sel   => link_s_fw_o(FLIT_WIDTH-1+VC_bit downto FLIT_WIDTH),
      src_sel  => src_sel_s);

link_s_bw_o <= not link_s_vc1_full & not link_s_vc0_full;
link_s_fw_o(FLIT_WIDTH-1 downto 0) <= link_w_vc0_rot when src_sel_s = X"01" else
                                      link_n_vc0_rot when src_sel_s = X"02" else
                                      link_e_vc0_rot when src_sel_s = X"04" else
                                      link_a_vc0_rot when src_sel_s = X"08" else
                                      link_w_vc1_rot when src_sel_s = X"10" else
                                      link_n_vc1_rot when src_sel_s = X"20" else
                                      link_e_vc1_rot when src_sel_s = X"40"
                                      else link_a_vc1_rot;
link_s_fw_o(FLIT_WIDTH+VC_bit) <= src_sel_s(0) or src_sel_s(1) or src_sel_s(2)
                                  or src_sel_s(3) or src_sel_s(4)
                                  or src_sel_s(5) or src_sel_s(6)
                                  or src_sel_s(7);

  -- Adapter
  buf_a_vc0 : vc_buffer
    port map (
      data_i => link_a_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_a_vc0,
      clk    => clk,
      rst    => rst,
      rd     => a_vc0_rd,
      wr     => a_vc0_wr,
      full   => link_a_vc0_full,
      empty  => link_a_vc0_b_empty);

a_vc0_wr <= '1' when (link_a_fw_i(DV_bit) = '1'
                      and link_a_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "0") else '0';
a_vc0_rd <= (src_sel_w(0) or src_sel_n(1) or src_sel_e(2) or src_sel_s(3));
link_a_vc0_rot <= link_a_vc0 when rot_a_vc0 = '0' else link_a_vc0(EOP_BIT_POS)
                  & link_a_vc0(TYPE_BIT_POS)
```

```
                & link_a_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                             downto HDR_ADDR_OFFSET)
                & link_a_vc0(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                             downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                & link_a_vc0(HDR_ADDR_OFFSET-1 downto 0);

    fh_a_vc0 : flit_handler
    generic map(
      INPUT => rt_a)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_a_vc0(EOP_BIT_POS),
      empty => link_a_vc0_b_empty,
      rd    => a_vc0_rd,
      dir_i => link_a_vc0(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_a_vc0_dir,
      rot   => rot_a_vc0);       -- Rotate header (Indicates header flit)

    buf_a_vc1 : vc_buffer
    port map (
      data_i => link_a_fw_i(FLIT_WIDTH-1 downto 0),
      data_o => link_a_vc1,
      clk    => clk,
      rst    => rst,
      rd     => a_vc1_rd,
      wr     => a_vc1_wr,
      full   => link_a_vc1_full,
      empty  => link_a_vc1_b_empty);

a_vc1_wr <= '1' when (link_a_fw_i(DV_bit) = '1'
                      and link_a_fw_i(FLIT_WIDTH-1 + VC_bit downto FLIT_WIDTH)
                      = "1") else '0';
a_vc1_rd <= (src_sel_w(4) or src_sel_n(5) or src_sel_e(6) or src_sel_s(7));
link_a_vc1_rot <= link_a_vc1 when rot_a_vc1 = '0' else link_a_vc1(EOP_BIT_POS)
                & link_a_vc1(TYPE_BIT_POS)
                & link_a_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1-HOP_BITS
                             downto HDR_ADDR_OFFSET)
                & link_a_vc1(HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-1
                             downto HDR_ADDR_OFFSET+HDR_ADDR_WIDTH-HOP_BITS)
                & link_a_vc1(HDR_ADDR_OFFSET-1 downto 0);

    fh_a_vc1 : flit_handler
    generic map(
      INPUT => rt_a)
    port map(
      clk   => clk,
      rst   => rst,
      eop   => link_a_vc1(EOP_BIT_POS),
      empty => link_a_vc1_b_empty,
      rd    => a_vc1_rd,
      dir_i => link_a_vc1(HDR_ADDR_WIDTH + HDR_ADDR_OFFSET -1
                          downto HDR_ADDR_WIDTH + HDR_ADDR_OFFSET - 2),
      dir_o => link_a_vc1_dir,
      rot   => rot_a_vc1);       -- Rotate header (Indicates header flit)

-- Determine VC order.
process (clk, rst)
begin  -- process
  if rst = '0' then                    -- asynchronous reset (active low)
    a_vc_old <= '0';
```

```
  elsif clk'event and clk = '1' then    -- rising clock edge
    if (link_a_vc0_b_empty xor link_a_vc1_b_empty) = '1' then
      a_vc_old <= link_a_vc0_b_empty;
    end if;
  end if;
end process;

link_a_vc0_empty <= '1' when ((link_a_vc1_dir = link_a_vc0_dir)
                              and a_vc_old = '1')
                    else link_a_vc0_b_empty;
link_a_vc1_empty <= '1' when ((link_a_vc1_dir = link_a_vc0_dir)
                              and a_vc_old = '0')
                    else link_a_vc1_b_empty;

  router_a : be_router
    generic map (
      DIR => rt_a)
    port map (
      clk         => clk,
      rst         => rst,
      src_0(2 downto 0)    => link_w_vc0_dir,
      src_0(3)    => link_w_vc0(EOP_BIT_POS),
      src_0(4)    => link_w_vc0_empty,
      src_1(2 downto 0)    => link_n_vc0_dir,
      src_1(3)    => link_n_vc0(EOP_BIT_POS),
      src_1(4)    => link_n_vc0_empty,
      src_2(2 downto 0)    => link_e_vc0_dir,
      src_2(3)    => link_e_vc0(EOP_BIT_POS),
      src_2(4)    => link_e_vc0_empty,
      src_3(2 downto 0)    => link_s_vc0_dir,
      src_3(3)    => link_s_vc0(EOP_BIT_POS),
      src_3(4)    => link_s_vc0_empty,
      src_4(2 downto 0)    => link_w_vc1_dir,
      src_4(3)    => link_w_vc1(EOP_BIT_POS),
      src_4(4)    => link_w_vc1_empty,
      src_5(2 downto 0)    => link_n_vc1_dir,
      src_5(3)    => link_n_vc1(EOP_BIT_POS),
      src_5(4)    => link_n_vc1_empty,
      src_6(2 downto 0)    => link_e_vc1_dir,
      src_6(3)    => link_e_vc1(EOP_BIT_POS),
      src_6(4)    => link_e_vc1_empty,
      src_7(2 downto 0)    => link_s_vc1_dir,
      src_7(3)    => link_s_vc1(EOP_BIT_POS),
      src_7(4)    => link_s_vc1_empty,
      vc_empty => link_a_bw_i,
      vc_sel   => link_a_fw_o(FLIT_WIDTH-1+VC_bit downto FLIT_WIDTH),
      src_sel  => src_sel_a);

link_a_bw_o <= not link_a_vc1_full & not link_a_vc0_full;
link_a_fw_o(FLIT_WIDTH-1 downto 0) <= link_w_vc0_rot when src_sel_a = X"01" else
                                      link_n_vc0_rot when src_sel_a = X"02" else
                                      link_e_vc0_rot when src_sel_a = X"04" else
                                      link_s_vc0_rot when src_sel_a = X"08" else
                                      link_w_vc1_rot when src_sel_a = X"10" else
                                      link_n_vc1_rot when src_sel_a = X"20" else
                                      link_e_vc1_rot when src_sel_a = X"40"
                                      else link_s_vc1_rot;
link_a_fw_o(FLIT_WIDTH+VC_bit) <= src_sel_a(0) or src_sel_a(1) or src_sel_a(2)
                                  or src_sel_a(3) or src_sel_a(4)
                                  or src_sel_a(5) or src_sel_a(6)
                                  or src_sel_a(7);
```

```
end mesh_router;
```

C.1.11 Traffic generator

```
--------------------------------------------------------------------------------
-- Generic traffic - Traffic generator
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use std.textio.all;
use work.types.all;

entity gen_traffic is
  generic (
    conf_file   : string;
    dump_file   : string;
    period      : integer;
    init_delay  : integer := 50;
    ADDR_BITS   : integer;
    DATA_WIDTH_I : integer;
    DATA_WIDTH_O : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_o : out std_logic_vector(ADDR_BITS-1 downto 0);
    type_o : out std_logic_vector(TYPE_BITS-1 downto 0);
    req_i  : in  std_logic;
    req_o  : out std_logic;
    ack_i  : in  std_logic;
    ack_o  : out std_logic);
end gen_traffic;

architecture arch of gen_traffic is

component file_log_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

  signal clock_count, init_count : integer := -1;

begin

process (clk, rst)
begin  -- process
  if rst = '0' then                     -- asynchronous reset (active low)
    clock_count <= -1;
    init_count <= -1;
  elsif clk'event and clk = '1' then    -- rising clock edge
    if clock_count = period-1 then
```

```
      clock_count <= 0;
    else
      clock_count <= clock_count + 1;
    end if;
    if init_count < init_delay then
      init_count <=  init_count + 1;
    end if;
  end if;
end process;

dump_log : file_log_std_logic
  generic map(
    filename   => dump_file,
    DATA_WIDTH => DATA_WIDTH_I)
  port map(
    clk    => clk,
    rst    => rst,
    dv     => req_i,
    data_i => data_i);

ack_o <= '1';


process (clock_count, rst)
    variable l : line;
    variable cc : integer;
    variable v : std_logic_vector(DATA_WIDTH_O+ADDR_BITS+TYPE_BITS-1 downto 0);
    variable w : integer := 0;
    file input : text is conf_file;
  begin  -- process
--    assert endfile(input) report "End of file" severity note;
    if clock_count'event and rst = '1' then
      if init_count = init_delay then
      if clock_count = 0 then
        -- Open conf_file
        file_close(input);
        file_open(input, external_name => conf_file, open_kind => read_mode);
        w := 0;
      end if;

      if w = 0  then
        if not endfile(input) then
          readline(input, l);
          read(l,cc);
          read(l,v);
        end if;
        w := 1;
      end if;

      if w = 0 and ack_i = '0' then
        req_o <= '1';
      else
        req_o <= '0';
      end if;

      if clock_count = cc then
        data_o <= v(DATA_WIDTH_O-1 downto 0);
        addr_o <= v(DATA_WIDTH_O+ADDR_BITS-1 downto DATA_WIDTH_O);
        type_o <= v(DATA_WIDTH_O+ADDR_BITS+TYPE_BITS-1
                    downto DATA_WIDTH_O+ADDR_BITS);
        req_o <= '1';
        w := 0;
```

```
      end if;
      end if;
    end if;
  end process;

end arch;
```

C.2 Optimized NoC

C.2.1 Optimized NoC

```
--------------------------------------------------------------------------------
-- R Mesh - Optimized NoC
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity r_mesh is
  port (
    clk : in std_logic;                 -- Clock
    rst : in std_logic;                 -- Reset

-- Adapter 00a
na00a_data_i : in  std_logic_vector (23 downto 0);
na00a_data_o : out std_logic_vector (23 downto 0);
na00a_addr_i : in  std_logic_vector (1 downto 0);
na00a_type_i : in  std_logic_vector (0 downto 0);
na00a_req_i  : in  std_logic;
na00a_req_o  : out std_logic;
na00a_ack_i  : in  std_logic;
na00a_ack_o  : out std_logic;
-- Adapter 00w
na00w_data_i : in  std_logic_vector (23 downto 0);
na00w_data_o : out std_logic_vector (23 downto 0);
na00w_addr_i : in  std_logic_vector (1 downto 0);
na00w_type_i : in  std_logic_vector (0 downto 0);
na00w_req_i  : in  std_logic;
na00w_req_o  : out std_logic;
na00w_ack_i  : in  std_logic;
na00w_ack_o  : out std_logic;
-- Adapter 01a
na01a_data_i : in  std_logic_vector (23 downto 0);
na01a_data_o : out std_logic_vector (23 downto 0);
na01a_addr_i : in  std_logic_vector (1 downto 0);
na01a_type_i : in  std_logic_vector (0 downto 0);
na01a_req_i  : in  std_logic;
na01a_req_o  : out std_logic;
na01a_ack_i  : in  std_logic;
na01a_ack_o  : out std_logic;
-- Adapter 01w
na01w_data_i : in  std_logic_vector (23 downto 0);
na01w_data_o : out std_logic_vector (23 downto 0);
na01w_addr_i : in  std_logic_vector (1 downto 0);
na01w_type_i : in  std_logic_vector (0 downto 0);
na01w_req_i  : in  std_logic;
na01w_req_o  : out std_logic;
na01w_ack_i  : in  std_logic;
```

```
na01w_ack_o  : out std_logic;
-- Adapter 01n
na01n_data_i : in  std_logic_vector (23 downto 0);
na01n_data_o : out std_logic_vector (23 downto 0);
na01n_addr_i : in  std_logic_vector (1 downto 0);
na01n_type_i : in  std_logic_vector (0 downto 0);
na01n_req_i  : in  std_logic;
na01n_req_o  : out std_logic;
na01n_ack_i  : in  std_logic;
na01n_ack_o  : out std_logic;
-- Adapter 10a
na10a_data_i : in  std_logic_vector (23 downto 0);
na10a_data_o : out std_logic_vector (23 downto 0);
na10a_addr_i : in  std_logic_vector (1 downto 0);
na10a_type_i : in  std_logic_vector (0 downto 0);
na10a_req_i  : in  std_logic;
na10a_req_o  : out std_logic;
na10a_ack_i  : in  std_logic;
na10a_ack_o  : out std_logic;
-- Adapter 10s
na10s_data_i : in  std_logic_vector (23 downto 0);
na10s_data_o : out std_logic_vector (23 downto 0);
na10s_addr_i : in  std_logic_vector (1 downto 0);
na10s_type_i : in  std_logic_vector (0 downto 0);
na10s_req_i  : in  std_logic;
na10s_req_o  : out std_logic;
na10s_ack_i  : in  std_logic;
na10s_ack_o  : out std_logic;
-- Adapter 11a
na11a_data_i : in  std_logic_vector (23 downto 0);
na11a_data_o : out std_logic_vector (23 downto 0);
na11a_addr_i : in  std_logic_vector (1 downto 0);
na11a_type_i : in  std_logic_vector (0 downto 0);
na11a_req_i  : in  std_logic;
na11a_req_o  : out std_logic;
na11a_ack_i  : in  std_logic;
na11a_ack_o  : out std_logic;
-- Adapter 11n
na11n_data_i : in  std_logic_vector (23 downto 0);
na11n_data_o : out std_logic_vector (23 downto 0);
na11n_addr_i : in  std_logic_vector (1 downto 0);
na11n_type_i : in  std_logic_vector (0 downto 0);
na11n_req_i  : in  std_logic;
na11n_req_o  : out std_logic;
na11n_ack_i  : in  std_logic;
na11n_ack_o  : out std_logic;
-- Adapter 20a
na20a_data_i : in  std_logic_vector (23 downto 0);
na20a_data_o : out std_logic_vector (23 downto 0);
na20a_addr_i : in  std_logic_vector (1 downto 0);
na20a_type_i : in  std_logic_vector (0 downto 0);
na20a_req_i  : in  std_logic;
na20a_req_o  : out std_logic;
na20a_ack_i  : in  std_logic;
na20a_ack_o  : out std_logic;
-- Adapter 20s
na20s_data_i : in  std_logic_vector (23 downto 0);
na20s_data_o : out std_logic_vector (23 downto 0);
na20s_addr_i : in  std_logic_vector (1 downto 0);
na20s_type_i : in  std_logic_vector (0 downto 0);
na20s_req_i  : in  std_logic;
na20s_req_o  : out std_logic;
```

```
na20s_ack_i  : in  std_logic;
na20s_ack_o  : out std_logic;
-- Adapter 21a
na21a_data_i : in  std_logic_vector (23 downto 0);
na21a_data_o : out std_logic_vector (23 downto 0);
na21a_addr_i : in  std_logic_vector (1 downto 0);
na21a_type_i : in  std_logic_vector (0 downto 0);
na21a_req_i  : in  std_logic;
na21a_req_o  : out std_logic;
na21a_ack_i  : in  std_logic;
na21a_ack_o  : out std_logic;
-- Adapter 21n
na21n_data_i : in  std_logic_vector (23 downto 0);
na21n_data_o : out std_logic_vector (23 downto 0);
na21n_addr_i : in  std_logic_vector (1 downto 0);
na21n_type_i : in  std_logic_vector (0 downto 0);
na21n_req_i  : in  std_logic;
na21n_req_o  : out std_logic;
na21n_ack_i  : in  std_logic;
na21n_ack_o  : out std_logic;
-- PGM Unit 21e
link_fw_21_e_o : out noc_link_fw;
link_bw_21_e_i : in  noc_link_bw;
link_fw_e_21_i : in  noc_link_fw;
link_bw_e_21_o : out noc_link_bw;
-- Adapter 30n
na30n_data_i : in  std_logic_vector (23 downto 0);
na30n_data_o : out std_logic_vector (23 downto 0);
na30n_addr_i : in  std_logic_vector (1 downto 0);
na30n_type_i : in  std_logic_vector (0 downto 0);
na30n_req_i  : in  std_logic;
na30n_req_o  : out std_logic;
na30n_ack_i  : in  std_logic;
na30n_ack_o  : out std_logic;
-- Adapter 30e
na30e_data_i : in  std_logic_vector (23 downto 0);
na30e_data_o : out std_logic_vector (23 downto 0);
na30e_addr_i : in  std_logic_vector (1 downto 0);
na30e_type_i : in  std_logic_vector (0 downto 0);
na30e_req_i  : in  std_logic;
na30e_req_o  : out std_logic;
na30e_ack_i  : in  std_logic;
na30e_ack_o  : out std_logic;
-- Adapter 30s
na30s_data_i : in  std_logic_vector (23 downto 0);
na30s_data_o : out std_logic_vector (23 downto 0);
na30s_addr_i : in  std_logic_vector (1 downto 0);
na30s_type_i : in  std_logic_vector (0 downto 0);
na30s_req_i  : in  std_logic;
na30s_req_o  : out std_logic;
na30s_ack_i  : in  std_logic;
na30s_ack_o  : out std_logic);
end r_mesh;


architecture r_arch of r_mesh is

component mesh_router
  port (
    clk         : in  std_logic;        -- Clock
    rst         : in  std_logic;        -- Reset
    -- Input links
```

```
    link_w_fw_i : in  noc_link_fw;      -- Input link west data
    link_w_bw_o : out noc_link_bw;      -- Input link west ack
    link_n_fw_i : in  noc_link_fw;      -- Input link north data
    link_n_bw_o : out noc_link_bw;      -- Input link north ack
    link_e_fw_i : in  noc_link_fw;      -- Input link east data
    link_e_bw_o : out noc_link_bw;      -- Input link east ack
    link_s_fw_i : in  noc_link_fw;      -- Input link south data
    link_s_bw_o : out noc_link_bw;      -- Input link south ack
    link_a_fw_i : in  noc_link_fw;      -- Input link adapter data
    link_a_bw_o : out noc_link_bw;      -- Input link adapter ack
    -- Output links
    link_w_fw_o : out noc_link_fw;      -- Input link west data
    link_w_bw_i : in  noc_link_bw;      -- Input link west ack
    link_n_fw_o : out noc_link_fw;      -- Input link north data
    link_n_bw_i : in  noc_link_bw;      -- Input link north ack
    link_e_fw_o : out noc_link_fw;      -- Input link east data
    link_e_bw_i : in  noc_link_bw;      -- Input link east ack
    link_s_fw_o : out noc_link_fw;      -- Input link south data
    link_s_bw_i : in  noc_link_bw;      -- Input link south ack
    link_a_fw_o : out noc_link_fw;      -- Input link adapter data
    link_a_bw_i : in  noc_link_bw);     -- Input link adapter ack
end component;

component mesh_na
  generic (
    ADDR_BITS    : integer := 4;
    DATA_WIDTH_I : integer := 32;
    DATA_WIDTH_O : integer := 32);
  port (
    noc_fw_i : in  noc_link_fw;
    noc_bw_o : out noc_link_bw;
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic;
    data_i   : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o   : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_i   : in  std_logic_vector(ADDR_BITS-1 downto 0);
    type_i   : in  std_logic_vector(TYPE_BITS-1 downto 0);
    req_i    : in  std_logic;
    req_o    : out std_logic;
    ack_i    : in  std_logic;
    ack_o    : out std_logic);
end component;

component stat_log
  generic (
    filename   : string;
    DATA_WIDTH : integer;
    PERIOD     : integer;
    INIT_DELAY : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

signal link_fw_00_w, link_fw_w_00, link_fw_00_01, link_fw_00_10, link_fw_00_s,
  link_fw_s_00, link_fw_00_na, link_fw_na_00, link_fw_10_00, link_fw_10_11,
  link_fw_10_20, link_fw_10_s, link_fw_s_10, link_fw_10_na, link_fw_na_10,
  link_fw_20_10, link_fw_20_21, link_fw_20_30, link_fw_20_s, link_fw_s_20,
  link_fw_20_na, link_fw_na_20, link_fw_30_20, link_fw_30_n, link_fw_n_30,
```

```
  link_fw_30_e, link_fw_e_30, link_fw_30_s, link_fw_s_30, link_fw_30_na,
  link_fw_na_30, link_fw_01_w, link_fw_w_01, link_fw_01_n, link_fw_n_01,
  link_fw_01_11, link_fw_01_00, link_fw_01_na, link_fw_na_01, link_fw_11_01,
  link_fw_11_n, link_fw_n_11, link_fw_11_21, link_fw_11_10, link_fw_11_na,
  link_fw_na_11, link_fw_21_11, link_fw_21_n, link_fw_n_21, link_fw_21_20,
  link_fw_21_na, link_fw_na_21  : noc_link_fw;
signal link_bw_00_w, link_bw_w_00, link_bw_00_01, link_bw_00_10, link_bw_00_s,
  link_bw_s_00, link_bw_00_na, link_bw_na_00, link_bw_10_00, link_bw_10_11,
  link_bw_10_20, link_bw_10_s, link_bw_s_10, link_bw_10_na, link_bw_na_10,
  link_bw_20_10, link_bw_20_21, link_bw_20_30, link_bw_20_s, link_bw_s_20,
  link_bw_20_na, link_bw_na_20, link_bw_30_20, link_bw_n_30, link_bw_30_n,
  link_bw_30_e, link_bw_e_30, link_bw_30_s, link_bw_s_30, link_bw_30_na,
  link_bw_na_30, link_bw_01_w, link_bw_w_01, link_bw_01_n, link_bw_n_01,
  link_bw_01_11, link_bw_01_00, link_bw_01_na, link_bw_na_01, link_bw_11_01,
  link_bw_11_n, link_bw_n_11, link_bw_11_21, link_bw_11_10, link_bw_11_na,
  link_bw_na_11, link_bw_21_11, link_bw_21_n, link_bw_n_21, link_bw_21_20,
  link_bw_21_na, link_bw_na_21 : noc_link_bw;

signal link_fw_gnd : noc_link_fw;
signal link_bw_high : noc_link_bw;

begin

link_fw_gnd <= (others => '0');
link_bw_high <= (others => '1');

node_00 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_w_00,
    link_w_bw_o => link_bw_w_00,
    link_n_fw_i => link_fw_01_00,
    link_n_bw_o => link_bw_01_00,
    link_e_fw_i => link_fw_10_00,
    link_e_bw_o => link_bw_10_00,
    link_s_fw_i => link_fw_gnd,
    link_s_bw_o => open,
    link_a_fw_i => link_fw_na_00,
    link_a_bw_o => link_bw_na_00,
    -- Output links
    link_w_fw_o => link_fw_00_w,
    link_w_bw_i => link_bw_00_w,
    link_n_fw_o => link_fw_00_01,
    link_n_bw_i => link_bw_00_01,
    link_e_fw_o => link_fw_00_10,
    link_e_bw_i => link_bw_00_10,
    link_s_fw_o => open,
    link_s_bw_i => link_bw_high,
    link_a_fw_o => link_fw_00_na,
    link_a_bw_i => link_bw_00_na);

na_00a : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_00_na,
    noc_bw_o => link_bw_00_na,
    noc_fw_o => link_fw_na_00,
```

```
    noc_bw_i => link_bw_na_00,
    clk      => clk,
    rst      => rst,
    data_i   => na00a_data_i,
    data_o   => na00a_data_o,
    addr_i   => na00a_addr_i,
    type_i   => na00a_type_i,
    req_i    => na00a_req_i,
    req_o    => na00a_req_o,
    ack_i    => na00a_ack_i,
    ack_o    => na00a_ack_o);

na_00w : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_00_w,
    noc_bw_o => link_bw_00_w,
    noc_fw_o => link_fw_w_00,
    noc_bw_i => link_bw_w_00,
    clk      => clk,
    rst      => rst,
    data_i   => na00w_data_i,
    data_o   => na00w_data_o,
    addr_i   => na00w_addr_i,
    type_i   => na00w_type_i,
    req_i    => na00w_req_i,
    req_o    => na00w_req_o,
    ack_i    => na00w_ack_i,
    ack_o    => na00w_ack_o);

node_10 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_00_10,
    link_w_bw_o => link_bw_00_10,
    link_n_fw_i => link_fw_11_10,
    link_n_bw_o => link_bw_11_10,
    link_e_fw_i => link_fw_20_10,
    link_e_bw_o => link_bw_20_10,
    link_s_fw_i => link_fw_s_10,
    link_s_bw_o => link_bw_s_10,
    link_a_fw_i => link_fw_na_10,
    link_a_bw_o => link_bw_na_10,
    -- Output links
    link_w_fw_o => link_fw_10_00,
    link_w_bw_i => link_bw_10_00,
    link_n_fw_o => link_fw_10_11,
    link_n_bw_i => link_bw_10_11,
    link_e_fw_o => link_fw_10_20,
    link_e_bw_i => link_bw_10_20,
    link_s_fw_o => link_fw_10_s,
    link_s_bw_i => link_bw_10_s,
    link_a_fw_o => link_fw_10_na,
    link_a_bw_i => link_bw_10_na);

na_10a : mesh_na
  generic map(
```

```
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_10_na,
    noc_bw_o => link_bw_10_na,
    noc_fw_o => link_fw_na_10,
    noc_bw_i => link_bw_na_10,
    clk      => clk,
    rst      => rst,
    data_i   => na10a_data_i,
    data_o   => na10a_data_o,
    addr_i   => na10a_addr_i,
    type_i   => na10a_type_i,
    req_i    => na10a_req_i,
    req_o    => na10a_req_o,
    ack_i    => na10a_ack_i,
    ack_o    => na10a_ack_o);

na_10s : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_10_s,
    noc_bw_o => link_bw_10_s,
    noc_fw_o => link_fw_s_10,
    noc_bw_i => link_bw_s_10,
    clk      => clk,
    rst      => rst,
    data_i   => na10s_data_i,
    data_o   => na10s_data_o,
    addr_i   => na10s_addr_i,
    type_i   => na10s_type_i,
    req_i    => na10s_req_i,
    req_o    => na10s_req_o,
    ack_i    => na10s_ack_i,
    ack_o    => na10s_ack_o);

node_20 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_10_20,
    link_w_bw_o => link_bw_10_20,
    link_n_fw_i => link_fw_21_20,
    link_n_bw_o => link_bw_21_20,
    link_e_fw_i => link_fw_30_20,
    link_e_bw_o => link_bw_30_20,
    link_s_fw_i => link_fw_s_20,
    link_s_bw_o => link_bw_s_20,
    link_a_fw_i => link_fw_na_20,
    link_a_bw_o => link_bw_na_20,
    -- Output links
    link_w_fw_o => link_fw_20_10,
    link_w_bw_i => link_bw_20_10,
    link_n_fw_o => link_fw_20_21,
    link_n_bw_i => link_bw_20_21,
    link_e_fw_o => link_fw_20_30,
    link_e_bw_i => link_bw_20_30,
```

```
    link_s_fw_o => link_fw_20_s,
    link_s_bw_i => link_bw_20_s,
    link_a_fw_o => link_fw_20_na,
    link_a_bw_i => link_bw_20_na);

na_20a : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_20_na,
    noc_bw_o => link_bw_20_na,
    noc_fw_o => link_fw_na_20,
    noc_bw_i => link_bw_na_20,
    clk      => clk,
    rst      => rst,
    data_i   => na20a_data_i,
    data_o   => na20a_data_o,
    addr_i   => na20a_addr_i,
    type_i   => na20a_type_i,
    req_i    => na20a_req_i,
    req_o    => na20a_req_o,
    ack_i    => na20a_ack_i,
    ack_o    => na20a_ack_o);

na_20s : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_20_s,
    noc_bw_o => link_bw_20_s,
    noc_fw_o => link_fw_s_20,
    noc_bw_i => link_bw_s_20,
    clk      => clk,
    rst      => rst,
    data_i   => na20s_data_i,
    data_o   => na20s_data_o,
    addr_i   => na20s_addr_i,
    type_i   => na20s_type_i,
    req_i    => na20s_req_i,
    req_o    => na20s_req_o,
    ack_i    => na20s_ack_i,
    ack_o    => na20s_ack_o);

node_30 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_20_30,
    link_w_bw_o => link_bw_20_30,
    link_n_fw_i => link_fw_n_30,
    link_n_bw_o => link_bw_n_30,
    link_e_fw_i => link_fw_e_30,
    link_e_bw_o => link_bw_e_30,
    link_s_fw_i => link_fw_s_30,
    link_s_bw_o => link_bw_s_30,
    link_a_fw_i => link_fw_gnd,
    link_a_bw_o => open,
```

```
    -- Output links
    link_w_fw_o => link_fw_30_20,
    link_w_bw_i => link_bw_30_20,
    link_n_fw_o => link_fw_30_n,
    link_n_bw_i => link_bw_30_n,
    link_e_fw_o => link_fw_30_e,
    link_e_bw_i => link_bw_30_e,
    link_s_fw_o => link_fw_30_s,
    link_s_bw_i => link_bw_30_s,
    link_a_fw_o => open,
    link_a_bw_i => link_bw_high);

na_30n : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_30_n,
    noc_bw_o => link_bw_30_n,
    noc_fw_o => link_fw_n_30,
    noc_bw_i => link_bw_n_30,
    clk      => clk,
    rst      => rst,
    data_i   => na30n_data_i,
    data_o   => na30n_data_o,
    addr_i   => na30n_addr_i,
    type_i   => na30n_type_i,
    req_i    => na30n_req_i,
    req_o    => na30n_req_o,
    ack_i    => na30n_ack_i,
    ack_o    => na30n_ack_o);

na_30e : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_30_e,
    noc_bw_o => link_bw_30_e,
    noc_fw_o => link_fw_e_30,
    noc_bw_i => link_bw_e_30,
    clk      => clk,
    rst      => rst,
    data_i   => na30e_data_i,
    data_o   => na30e_data_o,
    addr_i   => na30e_addr_i,
    type_i   => na30e_type_i,
    req_i    => na30e_req_i,
    req_o    => na30e_req_o,
    ack_i    => na30e_ack_i,
    ack_o    => na30e_ack_o);

na_30s : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_30_s,
    noc_bw_o => link_bw_30_s,
```

```
    noc_fw_o => link_fw_s_30,
    noc_bw_i => link_bw_s_30,
    clk      => clk,
    rst      => rst,
    data_i   => na30s_data_i,
    data_o   => na30s_data_o,
    addr_i   => na30s_addr_i,
    type_i   => na30s_type_i,
    req_i    => na30s_req_i,
    req_o    => na30s_req_o,
    ack_i    => na30s_ack_i,
    ack_o    => na30s_ack_o);

node_01 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_w_01,
    link_w_bw_o => link_bw_w_01,
    link_n_fw_i => link_fw_n_01,
    link_n_bw_o => link_bw_n_01,
    link_e_fw_i => link_fw_11_01,
    link_e_bw_o => link_bw_11_01,
    link_s_fw_i => link_fw_00_01,
    link_s_bw_o => link_bw_00_01,
    link_a_fw_i => link_fw_na_01,
    link_a_bw_o => link_bw_na_01,
    -- Output links
    link_w_fw_o => link_fw_01_w,
    link_w_bw_i => link_bw_01_w,
    link_n_fw_o => link_fw_01_n,
    link_n_bw_i => link_bw_01_n,
    link_e_fw_o => link_fw_01_11,
    link_e_bw_i => link_bw_01_11,
    link_s_fw_o => link_fw_01_00,
    link_s_bw_i => link_bw_01_00,
    link_a_fw_o => link_fw_01_na,
    link_a_bw_i => link_bw_01_na);

na_01n : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_01_n,
    noc_bw_o => link_bw_01_n,
    noc_fw_o => link_fw_n_01,
    noc_bw_i => link_bw_n_01,
    clk      => clk,
    rst      => rst,
    data_i   => na01n_data_i,
    data_o   => na01n_data_o,
    addr_i   => na01n_addr_i,
    type_i   => na01n_type_i,
    req_i    => na01n_req_i,
    req_o    => na01n_req_o,
    ack_i    => na01n_ack_i,
    ack_o    => na01n_ack_o);

na_01w : mesh_na
```

```
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_01_w,
    noc_bw_o => link_bw_01_w,
    noc_fw_o => link_fw_w_01,
    noc_bw_i => link_bw_w_01,
    clk      => clk,
    rst      => rst,
    data_i   => na01w_data_i,
    data_o   => na01w_data_o,
    addr_i   => na01w_addr_i,
    type_i   => na01w_type_i,
    req_i    => na01w_req_i,
    req_o    => na01w_req_o,
    ack_i    => na01w_ack_i,
    ack_o    => na01w_ack_o);

na_01a : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_01_na,
    noc_bw_o => link_bw_01_na,
    noc_fw_o => link_fw_na_01,
    noc_bw_i => link_bw_na_01,
    clk      => clk,
    rst      => rst,
    data_i   => na01a_data_i,
    data_o   => na01a_data_o,
    addr_i   => na01a_addr_i,
    type_i   => na01a_type_i,
    req_i    => na01a_req_i,
    req_o    => na01a_req_o,
    ack_i    => na01a_ack_i,
    ack_o    => na01a_ack_o);

node_11 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_01_11,
    link_w_bw_o => link_bw_01_11,
    link_n_fw_i => link_fw_n_11,
    link_n_bw_o => link_bw_n_11,
    link_e_fw_i => link_fw_21_11,
    link_e_bw_o => link_bw_21_11,
    link_s_fw_i => link_fw_10_11,
    link_s_bw_o => link_bw_10_11,
    link_a_fw_i => link_fw_na_11,
    link_a_bw_o => link_bw_na_11,
    -- Output links
    link_w_fw_o => link_fw_11_01,
    link_w_bw_i => link_bw_11_01,
    link_n_fw_o => link_fw_11_n,
    link_n_bw_i => link_bw_11_n,
    link_e_fw_o => link_fw_11_21,
```

```
    link_e_bw_i => link_bw_11_21,
    link_s_fw_o => link_fw_11_10,
    link_s_bw_i => link_bw_11_10,
    link_a_fw_o => link_fw_11_na,
    link_a_bw_i => link_bw_11_na);

na_11n : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_11_n,
    noc_bw_o => link_bw_11_n,
    noc_fw_o => link_fw_n_11,
    noc_bw_i => link_bw_n_11,
    clk      => clk,
    rst      => rst,
    data_i   => na11n_data_i,
    data_o   => na11n_data_o,
    addr_i   => na11n_addr_i,
    type_i   => na11n_type_i,
    req_i    => na11n_req_i,
    req_o    => na11n_req_o,
    ack_i    => na11n_ack_i,
    ack_o    => na11n_ack_o);

na_11a : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_11_na,
    noc_bw_o => link_bw_11_na,
    noc_fw_o => link_fw_na_11,
    noc_bw_i => link_bw_na_11,
    clk      => clk,
    rst      => rst,
    data_i   => na11a_data_i,
    data_o   => na11a_data_o,
    addr_i   => na11a_addr_i,
    type_i   => na11a_type_i,
    req_i    => na11a_req_i,
    req_o    => na11a_req_o,
    ack_i    => na11a_ack_i,
    ack_o    => na11a_ack_o);

node_21 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_11_21,
    link_w_bw_o => link_bw_11_21,
    link_n_fw_i => link_fw_n_21,
    link_n_bw_o => link_bw_n_21,
    link_e_fw_i => link_fw_e_21_i,
    link_e_bw_o => link_bw_e_21_o,
    link_s_fw_i => link_fw_20_21,
    link_s_bw_o => link_bw_20_21,
    link_a_fw_i => link_fw_na_21,
```

```
    link_a_bw_o => link_bw_na_21,
    -- Output links
    link_w_fw_o => link_fw_21_11,
    link_w_bw_i => link_bw_21_11,
    link_n_fw_o => link_fw_21_n,
    link_n_bw_i => link_bw_21_n,
    link_e_fw_o => link_fw_21_e_o,
    link_e_bw_i => link_bw_21_e_i,
    link_s_fw_o => link_fw_21_20,
    link_s_bw_i => link_bw_21_20,
    link_a_fw_o => link_fw_21_na,
    link_a_bw_i => link_bw_21_na);

na_21n : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_21_n,
    noc_bw_o => link_bw_21_n,
    noc_fw_o => link_fw_n_21,
    noc_bw_i => link_bw_n_21,
    clk      => clk,
    rst      => rst,
    data_i   => na21n_data_i,
    data_o   => na21n_data_o,
    addr_i   => na21n_addr_i,
    type_i   => na21n_type_i,
    req_i    => na21n_req_i,
    req_o    => na21n_req_o,
    ack_i    => na21n_ack_i,
    ack_o    => na21n_ack_o);

na_21a : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_21_na,
    noc_bw_o => link_bw_21_na,
    noc_fw_o => link_fw_na_21,
    noc_bw_i => link_bw_na_21,
    clk      => clk,
    rst      => rst,
    data_i   => na21a_data_i,
    data_o   => na21a_data_o,
    addr_i   => na21a_addr_i,
    type_i   => na21a_type_i,
    req_i    => na21a_req_i,
    req_o    => na21a_req_o,
    ack_i    => na21a_ack_i,
    ack_o    => na21a_ack_o);

end r_arch;
```

C.2.2 Global settings

```
--------------------------------------------------------------------------------
-- Types - Global definitions - Optimized NoC
```

```
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

package types is
  constant FLIT_WIDTH : integer := 36;  -- Flit width
  constant EOP_BIT_POS : integer := 35;  -- EOP bit position
  constant ADDR_HOPS : integer := 5;    -- Number of hops in header
  constant HOP_BITS : integer := 2;     -- Bit pr. hop
  constant TYPE_BITS : integer := 1;    -- Bits indicating type
  constant TYPE_BIT_POS : integer := 34;    -- Bits indicating type
  constant FLIT_DATA_WIDTH : integer := 34;    -- Data width
  constant HDR_ADDR_WIDTH : integer := ADDR_HOPS * HOP_BITS;
                                        -- Header bits for routing information
  constant HDR_ADDR_OFFSET : integer := 24;  -- Address position
  constant VC : integer := 2;           -- Number of virtual channels
  constant VC_bit : integer := 1;       -- log2(VC)
  constant DV_bit : integer := 37;      -- Data valid bit position

  subtype noc_link_fw is std_logic_vector((FLIT_WIDTH + VC_bit) downto 0);
  subtype noc_link_bw is std_logic_vector (VC-1 downto 0);
                                        -- Network acknowledge link
  subtype vc_data is std_logic_vector(FLIT_WIDTH-1 downto 0);  -- VC data

  subtype rt_direction is std_logic_vector(2 downto 0);  -- Routing direction

  constant rt_n : rt_direction := "000";  -- North
  constant rt_e : rt_direction := "001";  -- East
  constant rt_s : rt_direction := "011";  -- South
  constant rt_w : rt_direction := "010";  -- West
  constant rt_a : rt_direction := "100";  -- Adapter
  constant rt_x : rt_direction := "111";  -- Nowhere

end types;
```

C.2.3 Main testbench

```
--------------------------------------------------------------------------------
-- TB R Mesh clean - Optimized NoC main testbench
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity tb_r_mesh_clean is

end tb_r_mesh_clean;

architecture tb_clean_arch of tb_r_mesh_clean is

component r_mesh
  port (
    clk : in std_logic;                 -- Clock
    rst : in std_logic;                 -- Reset

-- Adapter 00a
na00a_data_i : in  std_logic_vector (23 downto 0);
```

```
na00a_data_o : out std_logic_vector (23 downto 0);
na00a_addr_i : in  std_logic_vector (1 downto 0);
na00a_type_i : in  std_logic_vector (0 downto 0);
na00a_req_i  : in  std_logic;
na00a_req_o  : out std_logic;
na00a_ack_i  : in  std_logic;
na00a_ack_o  : out std_logic;
-- Adapter 00w
na00w_data_i : in  std_logic_vector (23 downto 0);
na00w_data_o : out std_logic_vector (23 downto 0);
na00w_addr_i : in  std_logic_vector (1 downto 0);
na00w_type_i : in  std_logic_vector (0 downto 0);
na00w_req_i  : in  std_logic;
na00w_req_o  : out std_logic;
na00w_ack_i  : in  std_logic;
na00w_ack_o  : out std_logic;
-- Adapter 01a
na01a_data_i : in  std_logic_vector (23 downto 0);
na01a_data_o : out std_logic_vector (23 downto 0);
na01a_addr_i : in  std_logic_vector (1 downto 0);
na01a_type_i : in  std_logic_vector (0 downto 0);
na01a_req_i  : in  std_logic;
na01a_req_o  : out std_logic;
na01a_ack_i  : in  std_logic;
na01a_ack_o  : out std_logic;
-- Adapter 01w
na01w_data_i : in  std_logic_vector (23 downto 0);
na01w_data_o : out std_logic_vector (23 downto 0);
na01w_addr_i : in  std_logic_vector (1 downto 0);
na01w_type_i : in  std_logic_vector (0 downto 0);
na01w_req_i  : in  std_logic;
na01w_req_o  : out std_logic;
na01w_ack_i  : in  std_logic;
na01w_ack_o  : out std_logic;
-- Adapter 01n
na01n_data_i : in  std_logic_vector (23 downto 0);
na01n_data_o : out std_logic_vector (23 downto 0);
na01n_addr_i : in  std_logic_vector (1 downto 0);
na01n_type_i : in  std_logic_vector (0 downto 0);
na01n_req_i  : in  std_logic;
na01n_req_o  : out std_logic;
na01n_ack_i  : in  std_logic;
na01n_ack_o  : out std_logic;
-- Adapter 10a
na10a_data_i : in  std_logic_vector (23 downto 0);
na10a_data_o : out std_logic_vector (23 downto 0);
na10a_addr_i : in  std_logic_vector (1 downto 0);
na10a_type_i : in  std_logic_vector (0 downto 0);
na10a_req_i  : in  std_logic;
na10a_req_o  : out std_logic;
na10a_ack_i  : in  std_logic;
na10a_ack_o  : out std_logic;
-- Adapter 10s
na10s_data_i : in  std_logic_vector (23 downto 0);
na10s_data_o : out std_logic_vector (23 downto 0);
na10s_addr_i : in  std_logic_vector (1 downto 0);
na10s_type_i : in  std_logic_vector (0 downto 0);
na10s_req_i  : in  std_logic;
na10s_req_o  : out std_logic;
na10s_ack_i  : in  std_logic;
na10s_ack_o  : out std_logic;
-- Adapter 11a
```

```
na11a_data_i : in  std_logic_vector (23 downto 0);
na11a_data_o : out std_logic_vector (23 downto 0);
na11a_addr_i : in  std_logic_vector (1 downto 0);
na11a_type_i : in  std_logic_vector (0 downto 0);
na11a_req_i  : in  std_logic;
na11a_req_o  : out std_logic;
na11a_ack_i  : in  std_logic;
na11a_ack_o  : out std_logic;
-- Adapter 11n
na11n_data_i : in  std_logic_vector (23 downto 0);
na11n_data_o : out std_logic_vector (23 downto 0);
na11n_addr_i : in  std_logic_vector (1 downto 0);
na11n_type_i : in  std_logic_vector (0 downto 0);
na11n_req_i  : in  std_logic;
na11n_req_o  : out std_logic;
na11n_ack_i  : in  std_logic;
na11n_ack_o  : out std_logic;
-- Adapter 20a
na20a_data_i : in  std_logic_vector (23 downto 0);
na20a_data_o : out std_logic_vector (23 downto 0);
na20a_addr_i : in  std_logic_vector (1 downto 0);
na20a_type_i : in  std_logic_vector (0 downto 0);
na20a_req_i  : in  std_logic;
na20a_req_o  : out std_logic;
na20a_ack_i  : in  std_logic;
na20a_ack_o  : out std_logic;
-- Adapter 20s
na20s_data_i : in  std_logic_vector (23 downto 0);
na20s_data_o : out std_logic_vector (23 downto 0);
na20s_addr_i : in  std_logic_vector (1 downto 0);
na20s_type_i : in  std_logic_vector (0 downto 0);
na20s_req_i  : in  std_logic;
na20s_req_o  : out std_logic;
na20s_ack_i  : in  std_logic;
na20s_ack_o  : out std_logic;
-- Adapter 21a
na21a_data_i : in  std_logic_vector (23 downto 0);
na21a_data_o : out std_logic_vector (23 downto 0);
na21a_addr_i : in  std_logic_vector (1 downto 0);
na21a_type_i : in  std_logic_vector (0 downto 0);
na21a_req_i  : in  std_logic;
na21a_req_o  : out std_logic;
na21a_ack_i  : in  std_logic;
na21a_ack_o  : out std_logic;
-- Adapter 21n
na21n_data_i : in  std_logic_vector (23 downto 0);
na21n_data_o : out std_logic_vector (23 downto 0);
na21n_addr_i : in  std_logic_vector (1 downto 0);
na21n_type_i : in  std_logic_vector (0 downto 0);
na21n_req_i  : in  std_logic;
na21n_req_o  : out std_logic;
na21n_ack_i  : in  std_logic;
na21n_ack_o  : out std_logic;
-- PGM Unit 21e
link_fw_21_e_o : out noc_link_fw;
link_bw_21_e_i : in  noc_link_bw;
link_fw_e_21_i : in  noc_link_fw;
link_bw_e_21_o : out noc_link_bw;
-- Adapter 30n
na30n_data_i : in  std_logic_vector (23 downto 0);
na30n_data_o : out std_logic_vector (23 downto 0);
na30n_addr_i : in  std_logic_vector (1 downto 0);
```

```
na30n_type_i : in  std_logic_vector (0 downto 0);
na30n_req_i  : in  std_logic;
na30n_req_o  : out std_logic;
na30n_ack_i  : in  std_logic;
na30n_ack_o  : out std_logic;
-- Adapter 30e
na30e_data_i : in  std_logic_vector (23 downto 0);
na30e_data_o : out std_logic_vector (23 downto 0);
na30e_addr_i : in  std_logic_vector (1 downto 0);
na30e_type_i : in  std_logic_vector (0 downto 0);
na30e_req_i  : in  std_logic;
na30e_req_o  : out std_logic;
na30e_ack_i  : in  std_logic;
na30e_ack_o  : out std_logic;
-- Adapter 30s
na30s_data_i : in  std_logic_vector (23 downto 0);
na30s_data_o : out std_logic_vector (23 downto 0);
na30s_addr_i : in  std_logic_vector (1 downto 0);
na30s_type_i : in  std_logic_vector (0 downto 0);
na30s_req_i  : in  std_logic;
na30s_req_o  : out std_logic;
na30s_ack_i  : in  std_logic;
na30s_ack_o  : out std_logic);
end component;


component file_log_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH downto 0));
end component;

component file_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

component file_cc_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

component pgm_unit
  generic (
    filename   : string);
```

```
  port (
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic);
end component;

component gen_traffic
  generic (
    conf_file   : string;
    dump_file   : string;
    period      : integer;
    ADDR_BITS   : integer;
    DATA_WIDTH_I : integer;
    DATA_WIDTH_O : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_o : out std_logic_vector(ADDR_BITS-1 downto 0);
    type_o : out std_logic_vector(TYPE_BITS-1 downto 0);
    req_i  : in  std_logic;
    req_o  : out std_logic;
    ack_i  : in  std_logic;
    ack_o  : out std_logic);
end component;

component stat_log
  generic (
    filename   : string;
    DATA_WIDTH : integer;
    PERIOD     : integer;
    INIT_DELAY : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

signal clk, rst : std_logic := '0';

signal na00a_data_i, na00w_data_i, na01a_data_i, na01n_data_i, na01w_data_i,
  na10a_data_i, na10s_data_i, na11a_data_i, na11n_data_i, na20a_data_i,
  na20s_data_i, na21a_data_i, na21n_data_i, na21e_data_i, na30n_data_i,
  na30e_data_i, na30s_data_i, na00a_data_o, na00w_data_o, na01a_data_o,
  na01n_data_o, na01w_data_o, na10a_data_o, na10s_data_o, na11a_data_o,
  na11n_data_o, na20a_data_o, na20s_data_o, na21a_data_o, na21n_data_o,
  na21e_data_o, na30n_data_o, na30e_data_o, na30s_data_o
  : std_logic_vector(23 downto 0);
signal na00a_addr_i, na00w_addr_i, na01a_addr_i, na01n_addr_i, na01w_addr_i,
  na10a_addr_i, na10s_addr_i, na11a_addr_i, na11n_addr_i, na20a_addr_i,
  na20s_addr_i, na21a_addr_i, na21n_addr_i, na21e_addr_i, na30n_addr_i,
  na30e_addr_i, na30s_addr_i : std_logic_vector(1 downto 0);
signal na00a_type_i, na00w_type_i, na01a_type_i, na01n_type_i, na01w_type_i,
  na10a_type_i, na10s_type_i, na11a_type_i, na11n_type_i, na20a_type_i,
  na20s_type_i, na21a_type_i, na21n_type_i, na21e_type_i, na30n_type_i,
  na30e_type_i, na30s_type_i : std_logic_vector(TYPE_BITS-1 downto 0);
signal na00a_req_i, na00w_req_i, na01a_req_i, na01n_req_i, na01w_req_i,
  na10a_req_i, na10s_req_i, na11a_req_i, na11n_req_i, na20a_req_i,
  na20s_req_i, na21a_req_i, na21n_req_i, na21e_req_i, na30n_req_i,
  na30e_req_i, na30s_req_i, na00a_req_o, na00w_req_o, na01a_req_o,
```

```
  na01n_req_o, na01w_req_o, na10a_req_o, na10s_req_o, na11a_req_o,
  na11n_req_o, na20a_req_o, na20s_req_o, na21a_req_o, na21n_req_o,
  na21e_req_o, na30n_req_o, na30e_req_o, na30s_req_o : std_logic;
signal na00a_ack_i, na00w_ack_i, na01a_ack_i, na01n_ack_i, na01w_ack_i,
  na10a_ack_i, na10s_ack_i, na11a_ack_i, na11n_ack_i, na20a_ack_i,
  na20s_ack_i, na21a_ack_i, na21n_ack_i, na21e_ack_i, na30n_ack_i,
  na30e_ack_i, na30s_ack_i, na00a_ack_o, na00w_ack_o, na01a_ack_o,
  na01n_ack_o, na01w_ack_o, na10a_ack_o, na10s_ack_o, na11a_ack_o,
  na11n_ack_o, na20a_ack_o, na20s_ack_o, na21a_ack_o, na21n_ack_o,
  na21e_ack_o, na30n_ack_o, na30e_ack_o, na30s_ack_o : std_logic;

signal link_fw_e_21_i : noc_link_fw;
signal link_bw_e_21_o : noc_link_bw;

begin  -- test

  clk <= not clk after 5 ns;
  rst <= '1' after 17 ns;

noc : r_mesh
  port map(
    clk => clk,
    rst => rst,

na00a_data_i => na00a_data_i,
na00a_data_o => na00a_data_o,
na00a_addr_i => na00a_addr_i,
na00a_type_i => na00a_type_i,
na00a_req_i  => na00a_req_i,
na00a_req_o  => na00a_req_o,
na00a_ack_i  => na00a_ack_i,
na00a_ack_o  => na00a_ack_o,
na00w_data_i => na00w_data_i,
na00w_data_o => na00w_data_o,
na00w_addr_i => na00w_addr_i,
na00w_type_i => na00w_type_i,
na00w_req_i  => na00w_req_i,
na00w_req_o  => na00w_req_o,
na00w_ack_i  => na00w_ack_i,
na00w_ack_o  => na00w_ack_o,
na01a_data_i => na01a_data_i,
na01a_data_o => na01a_data_o,
na01a_addr_i => na01a_addr_i,
na01a_type_i => na01a_type_i,
na01a_req_i  => na01a_req_i,
na01a_req_o  => na01a_req_o,
na01a_ack_i  => na01a_ack_i,
na01a_ack_o  => na01a_ack_o,
na01w_data_i => na01w_data_i,
na01w_data_o => na01w_data_o,
na01w_addr_i => na01w_addr_i,
na01w_type_i => na01w_type_i,
na01w_req_i  => na01w_req_i,
na01w_req_o  => na01w_req_o,
na01w_ack_i  => na01w_ack_i,
na01w_ack_o  => na01w_ack_o,
na01n_data_i => na01n_data_i,
na01n_data_o => na01n_data_o,
na01n_addr_i => na01n_addr_i,
na01n_type_i => na01n_type_i,
na01n_req_i  => na01n_req_i,
na01n_req_o  => na01n_req_o,
```

```
na01n_ack_i  => na01n_ack_i,
na01n_ack_o  => na01n_ack_o,
na10a_data_i => na10a_data_i,
na10a_data_o => na10a_data_o,
na10a_addr_i => na10a_addr_i,
na10a_type_i => na10a_type_i,
na10a_req_i  => na10a_req_i,
na10a_req_o  => na10a_req_o,
na10a_ack_i  => na10a_ack_i,
na10a_ack_o  => na10a_ack_o,
na10s_data_i => na10s_data_i,
na10s_data_o => na10s_data_o,
na10s_addr_i => na10s_addr_i,
na10s_type_i => na10s_type_i,
na10s_req_i  => na10s_req_i,
na10s_req_o  => na10s_req_o,
na10s_ack_i  => na10s_ack_i,
na10s_ack_o  => na10s_ack_o,
na11a_data_i => na11a_data_i,
na11a_data_o => na11a_data_o,
na11a_addr_i => na11a_addr_i,
na11a_type_i => na11a_type_i,
na11a_req_i  => na11a_req_i,
na11a_req_o  => na11a_req_o,
na11a_ack_i  => na11a_ack_i,
na11a_ack_o  => na11a_ack_o,
na11n_data_i => na11n_data_i,
na11n_data_o => na11n_data_o,
na11n_addr_i => na11n_addr_i,
na11n_type_i => na11n_type_i,
na11n_req_i  => na11n_req_i,
na11n_req_o  => na11n_req_o,
na11n_ack_i  => na11n_ack_i,
na11n_ack_o  => na11n_ack_o,
na20a_data_i => na20a_data_i,
na20a_data_o => na20a_data_o,
na20a_addr_i => na20a_addr_i,
na20a_type_i => na20a_type_i,
na20a_req_i  => na20a_req_i,
na20a_req_o  => na20a_req_o,
na20a_ack_i  => na20a_ack_i,
na20a_ack_o  => na20a_ack_o,
na20s_data_i => na20s_data_i,
na20s_data_o => na20s_data_o,
na20s_addr_i => na20s_addr_i,
na20s_type_i => na20s_type_i,
na20s_req_i  => na20s_req_i,
na20s_req_o  => na20s_req_o,
na20s_ack_i  => na20s_ack_i,
na20s_ack_o  => na20s_ack_o,
na21a_data_i => na21a_data_i,
na21a_data_o => na21a_data_o,
na21a_addr_i => na21a_addr_i,
na21a_type_i => na21a_type_i,
na21a_req_i  => na21a_req_i,
na21a_req_o  => na21a_req_o,
na21a_ack_i  => na21a_ack_i,
na21a_ack_o  => na21a_ack_o,
na21n_data_i => na21n_data_i,
na21n_data_o => na21n_data_o,
na21n_addr_i => na21n_addr_i,
na21n_type_i => na21n_type_i,
```

```
na21n_req_i  => na21n_req_i,
na21n_req_o  => na21n_req_o,
na21n_ack_i  => na21n_ack_i,
na21n_ack_o  => na21n_ack_o,
link_fw_21_e_o => open,
link_bw_21_e_i => (others => '1'),
link_fw_e_21_i => link_fw_e_21_i,
link_bw_e_21_o => link_bw_e_21_o,
na30n_data_i => na30n_data_i,
na30n_data_o => na30n_data_o,
na30n_addr_i => na30n_addr_i,
na30n_type_i => na30n_type_i,
na30n_req_i  => na30n_req_i,
na30n_req_o  => na30n_req_o,
na30n_ack_i  => na30n_ack_i,
na30n_ack_o  => na30n_ack_o,
na30e_data_i => na30e_data_i,
na30e_data_o => na30e_data_o,
na30e_addr_i => na30e_addr_i,
na30e_type_i => na30e_type_i,
na30e_req_i  => na30e_req_i,
na30e_req_o  => na30e_req_o,
na30e_ack_i  => na30e_ack_i,
na30e_ack_o  => na30e_ack_o,
na30s_data_i => na30s_data_i,
na30s_data_o => na30s_data_o,
na30s_addr_i => na30s_addr_i,
na30s_type_i => na30s_type_i,
na30s_req_i  => na30s_req_i,
na30s_req_o  => na30s_req_o,
na30s_ack_i  => na30s_ack_i,
na30s_ack_o  => na30s_ack_o);


traffic00a : gen_traffic
  generic map(
    conf_file   => "1a.txt",
    dump_file   => "1a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na00a_data_o,
    data_o => na00a_data_i,
    addr_o => na00a_addr_i,
    type_o => na00a_type_i,
    req_i  => na00a_req_o,
    req_o  => na00a_req_i,
    ack_i  => na00a_ack_o,
    ack_o  => na00a_ack_i);

traffic00w : gen_traffic
  generic map(
    conf_file   => "5.txt",
    dump_file   => "5_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
```

```
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na00w_data_o,
    data_o => na00w_data_i,
    addr_o => na00w_addr_i,
    type_o => na00w_type_i,
    req_i  => na00w_req_o,
    req_o  => na00w_req_i,
    ack_i  => na00w_ack_o,
    ack_o  => na00w_ack_i);


traffic01a : gen_traffic
  generic map(
    conf_file   => "6.txt",
    dump_file   => "6_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na01a_data_o,
    data_o => na01a_data_i,
    addr_o => na01a_addr_i,
    type_o => na01a_type_i,
    req_i  => na01a_req_o,
    req_o  => na01a_req_i,
    ack_i  => na01a_ack_o,
    ack_o  => na01a_ack_i);

traffic01w : gen_traffic
  generic map(
    conf_file   => "1b.txt",
    dump_file   => "1b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na01w_data_o,
    data_o => na01w_data_i,
    addr_o => na01w_addr_i,
    type_o => na01w_type_i,
    req_i  => na01w_req_o,
    req_o  => na01w_req_i,
    ack_i  => na01w_ack_o,
    ack_o  => na01w_ack_i);

traffic01n : gen_traffic
  generic map(
    conf_file   => "7.txt",
    dump_file   => "7_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
```

```
    clk    => clk,
    rst    => rst,
    data_i => na01n_data_o,
    data_o => na01n_data_i,
    addr_o => na01n_addr_i,
    type_o => na01n_type_i,
    req_i  => na01n_req_o,
    req_o  => na01n_req_i,
    ack_i  => na01n_ack_o,
    ack_o  => na01n_ack_i);

traffic10a : gen_traffic
  generic map(
    conf_file   => "2b.txt",
    dump_file   => "2b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na10a_data_o,
    data_o => na10a_data_i,
    addr_o => na10a_addr_i,
    type_o => na10a_type_i,
    req_i  => na10a_req_o,
    req_o  => na10a_req_i,
    ack_i  => na10a_ack_o,
    ack_o  => na10a_ack_i);

traffic10s : gen_traffic
  generic map(
    conf_file   => "4b.txt",
    dump_file   => "4b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na10s_data_o,
    data_o => na10s_data_i,
    addr_o => na10s_addr_i,
    type_o => na10s_type_i,
    req_i  => na10s_req_o,
    req_o  => na10s_req_i,
    ack_i  => na10s_ack_o,
    ack_o  => na10s_ack_i);

traffic11a : gen_traffic
  generic map(
    conf_file   => "2a.txt",
    dump_file   => "2a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
```

```
    data_i => na11a_data_o,
    data_o => na11a_data_i,
    addr_o => na11a_addr_i,
    type_o => na11a_type_i,
    req_i  => na11a_req_o,
    req_o  => na11a_req_i,
    ack_i  => na11a_ack_o,
    ack_o  => na11a_ack_i);

traffic11n : gen_traffic
  generic map(
    conf_file   => "3a.txt",
    dump_file   => "3a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na11n_data_o,
    data_o => na11n_data_i,
    addr_o => na11n_addr_i,
    type_o => na11n_type_i,
    req_i  => na11n_req_o,
    req_o  => na11n_req_i,
    ack_i  => na11n_ack_o,
    ack_o  => na11n_ack_i);

traffic20a : gen_traffic
  generic map(
    conf_file   => "9.txt",
    dump_file   => "9_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na20a_data_o,
    data_o => na20a_data_i,
    addr_o => na20a_addr_i,
    type_o => na20a_type_i,
    req_i  => na20a_req_o,
    req_o  => na20a_req_i,
    ack_i  => na20a_ack_o,
    ack_o  => na20a_ack_i);

traffic20s : gen_traffic
  generic map(
    conf_file   => "10.txt",
    dump_file   => "10_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na20s_data_o,
    data_o => na20s_data_i,
```

```
    addr_o => na20s_addr_i,
    type_o => na20s_type_i,
    req_i  => na20s_req_o,
    req_o  => na20s_req_i,
    ack_i  => na20s_ack_o,
    ack_o  => na20s_ack_i);

traffic21a : gen_traffic
  generic map(
    conf_file   => "4a.txt",
    dump_file   => "4a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na21a_data_o,
    data_o => na21a_data_i,
    addr_o => na21a_addr_i,
    type_o => na21a_type_i,
    req_i  => na21a_req_o,
    req_o  => na21a_req_i,
    ack_i  => na21a_ack_o,
    ack_o  => na21a_ack_i);

traffic21n : gen_traffic
  generic map(
    conf_file   => "3b.txt",
    dump_file   => "3b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na21n_data_o,
    data_o => na21n_data_i,
    addr_o => na21n_addr_i,
    type_o => na21n_type_i,
    req_i  => na21n_req_o,
    req_o  => na21n_req_i,
    ack_i  => na21n_ack_o,
    ack_o  => na21n_ack_i);

traffic30n : gen_traffic
  generic map(
    conf_file   => "empty.txt",
    dump_file   => "12_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na30n_data_o,
    data_o => na30n_data_i,
    addr_o => na30n_addr_i,
    type_o => na30n_type_i,
```

```
    req_i  => na30n_req_o,
    req_o  => na30n_req_i,
    ack_i  => na30n_ack_o,
    ack_o  => na30n_ack_i);

traffic30e : gen_traffic
  generic map(
    conf_file   => "11.txt",
    dump_file   => "11_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na30e_data_o,
    data_o => na30e_data_i,
    addr_o => na30e_addr_i,
    type_o => na30e_type_i,
    req_i  => na30e_req_o,
    req_o  => na30e_req_i,
    ack_i  => na30e_ack_o,
    ack_o  => na30e_ack_i);

traffic30s : gen_traffic
  generic map(
    conf_file   => "8.txt",
    dump_file   => "8_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na30s_data_o,
    data_o => na30s_data_i,
    addr_o => na30s_addr_i,
    type_o => na30s_type_i,
    req_i  => na30s_req_o,
    req_o  => na30s_req_i,
    ack_i  => na30s_ack_o,
    ack_o  => na30s_ack_i);

pgm : pgm_unit
  generic map(
    filename   => "pgm.txt")
  port map(
    noc_fw_o => link_fw_e_21_i,
    noc_bw_i => link_bw_e_21_o,
    clk      => clk,
    rst      => rst);

end tb_clean_arch;
```

C.3 Mesh NoC

C.3.1 Mesh NoC

```
--------------------------------------------------------------------------------
-- Mesh 4x4 - Mesh NoC
-- by Morten Sleth Rasmussen, s011295
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity mesh_4x4 is
  port (
    clk : in std_logic;                 -- Clock
    rst : in std_logic;                 -- Reset

      -- Test enable pin, that will be inserted by DC
      -- RTL_SYNTHESIS OFF
      -- pragma synthesis_off
      test_se    : in  std_logic;   -- syncheck_off
      -- pragma synthesis_on
      -- RTL_SYNTHESIS ON
-- Programming unit
pgm_data_i : in  noc_link_fw;
pgm_data_o : out  noc_link_bw;

-- Adapter 00
na00_data_i : in  std_logic_vector (23 downto 0);
na00_data_o : out std_logic_vector (23 downto 0);
na00_addr_i : in  std_logic_vector (1 downto 0);
na00_type_i : in  std_logic_vector (0 downto 0);
na00_req_i  : in  std_logic;
na00_req_o  : out std_logic;
na00_ack_i  : in  std_logic;
na00_ack_o  : out std_logic;
-- Adapter 01
na01_data_i : in  std_logic_vector (23 downto 0);
na01_data_o : out std_logic_vector (23 downto 0);
na01_addr_i : in  std_logic_vector (1 downto 0);
na01_type_i : in  std_logic_vector (0 downto 0);
na01_req_i  : in  std_logic;
na01_req_o  : out std_logic;
na01_ack_i  : in  std_logic;
na01_ack_o  : out std_logic;
-- Adapter 02
na02_data_i : in  std_logic_vector (23 downto 0);
na02_data_o : out std_logic_vector (23 downto 0);
na02_addr_i : in  std_logic_vector (1 downto 0);
na02_type_i : in  std_logic_vector (0 downto 0);
na02_req_i  : in  std_logic;
na02_req_o  : out std_logic;
na02_ack_i  : in  std_logic;
na02_ack_o  : out std_logic;
-- Adapter 03
na03_data_i : in  std_logic_vector (23 downto 0);
na03_data_o : out std_logic_vector (23 downto 0);
na03_addr_i : in  std_logic_vector (1 downto 0);
na03_type_i : in  std_logic_vector (0 downto 0);
na03_req_i  : in  std_logic;
na03_req_o  : out std_logic;
```

```
na03_ack_i  : in  std_logic;
na03_ack_o  : out std_logic;
-- Adapter 10
na10_data_i : in  std_logic_vector (23 downto 0);
na10_data_o : out std_logic_vector (23 downto 0);
na10_addr_i : in  std_logic_vector (1 downto 0);
na10_type_i : in  std_logic_vector (0 downto 0);
na10_req_i  : in  std_logic;
na10_req_o  : out std_logic;
na10_ack_i  : in  std_logic;
na10_ack_o  : out std_logic;
-- Adapter 11
na11_data_i : in  std_logic_vector (23 downto 0);
na11_data_o : out std_logic_vector (23 downto 0);
na11_addr_i : in  std_logic_vector (1 downto 0);
na11_type_i : in  std_logic_vector (0 downto 0);
na11_req_i  : in  std_logic;
na11_req_o  : out std_logic;
na11_ack_i  : in  std_logic;
na11_ack_o  : out std_logic;
-- Adapter 12
na12_data_i : in  std_logic_vector (23 downto 0);
na12_data_o : out std_logic_vector (23 downto 0);
na12_addr_i : in  std_logic_vector (1 downto 0);
na12_type_i : in  std_logic_vector (0 downto 0);
na12_req_i  : in  std_logic;
na12_req_o  : out std_logic;
na12_ack_i  : in  std_logic;
na12_ack_o  : out std_logic;
-- Adapter 13
na13_data_i : in  std_logic_vector (23 downto 0);
na13_data_o : out std_logic_vector (23 downto 0);
na13_addr_i : in  std_logic_vector (1 downto 0);
na13_type_i : in  std_logic_vector (0 downto 0);
na13_req_i  : in  std_logic;
na13_req_o  : out std_logic;
na13_ack_i  : in  std_logic;
na13_ack_o  : out std_logic;
-- Adapter 20
na20_data_i : in  std_logic_vector (23 downto 0);
na20_data_o : out std_logic_vector (23 downto 0);
na20_addr_i : in  std_logic_vector (1 downto 0);
na20_type_i : in  std_logic_vector (0 downto 0);
na20_req_i  : in  std_logic;
na20_req_o  : out std_logic;
na20_ack_i  : in  std_logic;
na20_ack_o  : out std_logic;
-- Adapter 21
na21_data_i : in  std_logic_vector (23 downto 0);
na21_data_o : out std_logic_vector (23 downto 0);
na21_addr_i : in  std_logic_vector (1 downto 0);
na21_type_i : in  std_logic_vector (0 downto 0);
na21_req_i  : in  std_logic;
na21_req_o  : out std_logic;
na21_ack_i  : in  std_logic;
na21_ack_o  : out std_logic;
-- Adapter 22
na22_data_i : in  std_logic_vector (23 downto 0);
na22_data_o : out std_logic_vector (23 downto 0);
na22_addr_i : in  std_logic_vector (1 downto 0);
na22_type_i : in  std_logic_vector (0 downto 0);
na22_req_i  : in  std_logic;
```

```
na22_req_o  : out std_logic;
na22_ack_i  : in  std_logic;
na22_ack_o  : out std_logic;
-- Adapter 23
na23_data_i : in  std_logic_vector (23 downto 0);
na23_data_o : out std_logic_vector (23 downto 0);
na23_addr_i : in  std_logic_vector (1 downto 0);
na23_type_i : in  std_logic_vector (0 downto 0);
na23_req_i  : in  std_logic;
na23_req_o  : out std_logic;
na23_ack_i  : in  std_logic;
na23_ack_o  : out std_logic;
-- Adapter 30
na30_data_i : in  std_logic_vector (23 downto 0);
na30_data_o : out std_logic_vector (23 downto 0);
na30_addr_i : in  std_logic_vector (1 downto 0);
na30_type_i : in  std_logic_vector (0 downto 0);
na30_req_i  : in  std_logic;
na30_req_o  : out std_logic;
na30_ack_i  : in  std_logic;
na30_ack_o  : out std_logic;
-- Adapter 31
na31_data_i : in  std_logic_vector (23 downto 0);
na31_data_o : out std_logic_vector (23 downto 0);
na31_addr_i : in  std_logic_vector (1 downto 0);
na31_type_i : in  std_logic_vector (0 downto 0);
na31_req_i  : in  std_logic;
na31_req_o  : out std_logic;
na31_ack_i  : in  std_logic;
na31_ack_o  : out std_logic;
-- Adapter 32
na32_data_i : in  std_logic_vector (23 downto 0);
na32_data_o : out std_logic_vector (23 downto 0);
na32_addr_i : in  std_logic_vector (1 downto 0);
na32_type_i : in  std_logic_vector (0 downto 0);
na32_req_i  : in  std_logic;
na32_req_o  : out std_logic;
na32_ack_i  : in  std_logic;
na32_ack_o  : out std_logic;
-- Adapter 33
na33_data_i : in  std_logic_vector (23 downto 0);
na33_data_o : out std_logic_vector (23 downto 0);
na33_addr_i : in  std_logic_vector (1 downto 0);
na33_type_i : in  std_logic_vector (0 downto 0);
na33_req_i  : in  std_logic;
na33_req_o  : out std_logic;
na33_ack_i  : in  std_logic;
na33_ack_o  : out std_logic);

end mesh_4x4;

architecture arch_4x4 of mesh_4x4 is

component mesh_router
  port (
    clk         : in  std_logic;        -- Clock
    rst         : in  std_logic;        -- Reset
    -- Input links
    link_w_fw_i : in  noc_link_fw;      -- Input link west data
    link_w_bw_o : out noc_link_bw;      -- Input link west ack
    link_n_fw_i : in  noc_link_fw;      -- Input link north data
    link_n_bw_o : out noc_link_bw;      -- Input link north ack
```

```
    link_e_fw_i : in  noc_link_fw;      -- Input link east data
    link_e_bw_o : out noc_link_bw;      -- Input link east ack
    link_s_fw_i : in  noc_link_fw;      -- Input link south data
    link_s_bw_o : out noc_link_bw;      -- Input link south ack
    link_a_fw_i : in  noc_link_fw;      -- Input link adapter data
    link_a_bw_o : out noc_link_bw;      -- Input link adapter ack
    -- Output links
    link_w_fw_o : out noc_link_fw;      -- Input link west data
    link_w_bw_i : in  noc_link_bw;      -- Input link west ack
    link_n_fw_o : out noc_link_fw;      -- Input link north data
    link_n_bw_i : in  noc_link_bw;      -- Input link north ack
    link_e_fw_o : out noc_link_fw;      -- Input link east data
    link_e_bw_i : in  noc_link_bw;      -- Input link east ack
    link_s_fw_o : out noc_link_fw;      -- Input link south data
    link_s_bw_i : in  noc_link_bw;      -- Input link south ack
    link_a_fw_o : out noc_link_fw;      -- Input link adapter data
    link_a_bw_i : in  noc_link_bw);     -- Input link adapter ack
end component;

component mesh_na
  generic (
    ADDR_BITS    : integer := 4;
    DATA_WIDTH_I : integer := 32;
    DATA_WIDTH_O : integer := 32);
  port (
    noc_fw_i : in  noc_link_fw;
    noc_bw_o : out noc_link_bw;
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic;
    data_i   : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o   : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_i   : in  std_logic_vector(ADDR_BITS-1 downto 0);
    type_i   : in  std_logic_vector(TYPE_BITS-1 downto 0);
    req_i    : in  std_logic;
    req_o    : out std_logic;
    ack_i    : in  std_logic;
    ack_o    : out std_logic);
end component;

component file_log_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH downto 0));
end component;

component file_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;
```

```
component file_cc_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

component pgm_unit
  generic (
    filename   : string);
  port (
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic);
end component;

component gen_traffic
  generic (
    conf_file   : string;
    dump_file   : string;
    period      : integer;
    ADDR_BITS   : integer;
    DATA_WIDTH_I : integer;
    DATA_WIDTH_O : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_o : out std_logic_vector(ADDR_BITS-1 downto 0);
    type_o : out std_logic_vector(TYPE_BITS-1 downto 0);
    req_i  : in  std_logic;
    req_o  : out std_logic;
    ack_i  : in  std_logic;
    ack_o  : out std_logic);
end component;

component stat_log
  generic (
    filename   : string;
    DATA_WIDTH : integer;
    PERIOD     : integer;
    INIT_DELAY : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

--signal clk, rst : std_logic := '0';

signal link_fw_00_01, link_fw_00_10, link_fw_00_na, link_fw_na_00,
  link_fw_10_00, link_fw_10_11, link_fw_10_20, link_fw_10_na,
  link_fw_na_10, link_fw_20_10, link_fw_20_21, link_fw_20_30,
  link_fw_20_na, link_fw_na_20, link_fw_30_20, link_fw_30_31,
  link_fw_30_na, link_fw_na_30, link_fw_01_02, link_fw_01_11,
```

```
  link_fw_01_00, link_fw_01_na, link_fw_na_01, link_fw_11_01,
  link_fw_11_12, link_fw_11_21, link_fw_11_10, link_fw_11_na,
  link_fw_na_11, link_fw_21_11, link_fw_21_22, link_fw_21_31,
  link_fw_21_20, link_fw_21_na, link_fw_na_21, link_fw_31_21,
  link_fw_31_32, link_fw_31_30, link_fw_31_na, link_fw_na_31,
  link_fw_02_03, link_fw_02_12, link_fw_02_01, link_fw_02_na,
  link_fw_na_02, link_fw_12_02, link_fw_12_13, link_fw_12_22,
  link_fw_12_11, link_fw_12_na, link_fw_na_12, link_fw_22_12,
  link_fw_22_23, link_fw_22_32, link_fw_22_21, link_fw_22_na,
  link_fw_na_22, link_fw_32_22, link_fw_32_33, link_fw_32_31,
  link_fw_32_na, link_fw_na_32, link_fw_03_13, link_fw_03_02,
  link_fw_03_na, link_fw_na_03, link_fw_13_03, link_fw_13_23,
  link_fw_13_12, link_fw_13_na, link_fw_na_13, link_fw_23_13,
  link_fw_23_33, link_fw_23_22, link_fw_23_na, link_fw_na_23,
  link_fw_33_23, link_fw_33_32, link_fw_33_na, link_fw_na_33 : noc_link_fw;
signal link_bw_00_01, link_bw_00_10, link_bw_00_na, link_bw_na_00,
  link_bw_10_00, link_bw_10_11, link_bw_10_20, link_bw_10_na, link_bw_na_10,
  link_bw_20_10, link_bw_20_21, link_bw_20_30, link_bw_20_na, link_bw_na_20,
  link_bw_30_20, link_bw_30_31, link_bw_30_na, link_bw_na_30, link_bw_01_02,
  link_bw_01_11, link_bw_01_00, link_bw_01_na, link_bw_na_01, link_bw_11_01,
  link_bw_11_12, link_bw_11_21, link_bw_11_10, link_bw_11_na, link_bw_na_11,
  link_bw_21_11, link_bw_21_22, link_bw_21_31, link_bw_21_20, link_bw_21_na,
  link_bw_na_21, link_bw_31_21, link_bw_31_32, link_bw_31_30, link_bw_31_na,
  link_bw_na_31, link_bw_02_03, link_bw_02_12, link_bw_02_01, link_bw_02_na,
  link_bw_na_02, link_bw_12_02, link_bw_12_13, link_bw_12_22, link_bw_12_11,
  link_bw_12_na, link_bw_na_12, link_bw_22_12, link_bw_22_23, link_bw_22_32,
  link_bw_22_21, link_bw_22_na, link_bw_na_22, link_bw_32_22, link_bw_32_33,
  link_bw_32_31, link_bw_32_na, link_bw_na_32, link_bw_03_13, link_bw_03_02,
  link_bw_03_na, link_bw_na_03, link_bw_13_03, link_bw_13_23, link_bw_13_12,
  link_bw_13_na, link_bw_na_13, link_bw_23_13, link_bw_23_33, link_bw_23_22,
  link_bw_23_na, link_bw_na_23, link_bw_33_23, link_bw_33_32, link_bw_33_na,
  link_bw_na_33 : noc_link_bw;

signal link_fw_gnd : noc_link_fw;
signal link_bw_high : noc_link_bw;

begin  -- test

link_fw_gnd <= (others => '0');
link_bw_high <= (others => '1');

node_00 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => pgm_data_i,
    link_w_bw_o => pgm_data_o,
    link_n_fw_i => link_fw_01_00,
    link_n_bw_o => link_bw_01_00,
    link_e_fw_i => link_fw_10_00,
    link_e_bw_o => link_bw_10_00,
    link_s_fw_i => link_fw_gnd,
    link_s_bw_o => open,
    link_a_fw_i => link_fw_na_00,
    link_a_bw_o => link_bw_na_00,
    -- Output links
    link_w_fw_o => open,
    link_w_bw_i => link_bw_high,
    link_n_fw_o => link_fw_00_01,
    link_n_bw_i => link_bw_00_01,
    link_e_fw_o => link_fw_00_10,
```

```
    link_e_bw_i => link_bw_00_10,
    link_s_fw_o => open,
    link_s_bw_i => link_bw_high,
    link_a_fw_o => link_fw_00_na,
    link_a_bw_i => link_bw_00_na);

na_00 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_00_na,
    noc_bw_o => link_bw_00_na,
    noc_fw_o => link_fw_na_00,
    noc_bw_i => link_bw_na_00,
    clk      => clk,
    rst      => rst,
    data_i   => na00_data_i,
    data_o   => na00_data_o,
    addr_i   => na00_addr_i,
    type_i   => na00_type_i,
    req_i    => na00_req_i,
    req_o    => na00_req_o,
    ack_i    => na00_ack_i,
    ack_o    => na00_ack_o);

node_10 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_00_10,
    link_w_bw_o => link_bw_00_10,
    link_n_fw_i => link_fw_11_10,
    link_n_bw_o => link_bw_11_10,
    link_e_fw_i => link_fw_20_10,
    link_e_bw_o => link_bw_20_10,
    link_s_fw_i => link_fw_gnd,
    link_s_bw_o => open,
    link_a_fw_i => link_fw_na_10,
    link_a_bw_o => link_bw_na_10,
    -- Output links
    link_w_fw_o => link_fw_10_00,
    link_w_bw_i => link_bw_10_00,
    link_n_fw_o => link_fw_10_11,
    link_n_bw_i => link_bw_10_11,
    link_e_fw_o => link_fw_10_20,
    link_e_bw_i => link_bw_10_20,
    link_s_fw_o => open,
    link_s_bw_i => link_bw_high,
    link_a_fw_o => link_fw_10_na,
    link_a_bw_i => link_bw_10_na);

na_10 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_10_na,
    noc_bw_o => link_bw_10_na,
```

```
    noc_fw_o => link_fw_na_10,
    noc_bw_i => link_bw_na_10,
    clk      => clk,
    rst      => rst,
    data_i   => na10_data_i,
    data_o   => na10_data_o,
    addr_i   => na10_addr_i,
    type_i   => na10_type_i,
    req_i    => na10_req_i,
    req_o    => na10_req_o,
    ack_i    => na10_ack_i,
    ack_o    => na10_ack_o);

node_20 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_10_20,
    link_w_bw_o => link_bw_10_20,
    link_n_fw_i => link_fw_21_20,
    link_n_bw_o => link_bw_21_20,
    link_e_fw_i => link_fw_30_20,
    link_e_bw_o => link_bw_30_20,
    link_s_fw_i => link_fw_gnd,
    link_s_bw_o => open,
    link_a_fw_i => link_fw_na_20,
    link_a_bw_o => link_bw_na_20,
    -- Output links
    link_w_fw_o => link_fw_20_10,
    link_w_bw_i => link_bw_20_10,
    link_n_fw_o => link_fw_20_21,
    link_n_bw_i => link_bw_20_21,
    link_e_fw_o => link_fw_20_30,
    link_e_bw_i => link_bw_20_30,
    link_s_fw_o => open,
    link_s_bw_i => link_bw_high,
    link_a_fw_o => link_fw_20_na,
    link_a_bw_i => link_bw_20_na);

na_20 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_20_na,
    noc_bw_o => link_bw_20_na,
    noc_fw_o => link_fw_na_20,
    noc_bw_i => link_bw_na_20,
    clk      => clk,
    rst      => rst,
    data_i   => na20_data_i,
    data_o   => na20_data_o,
    addr_i   => na20_addr_i,
    type_i   => na20_type_i,
    req_i    => na20_req_i,
    req_o    => na20_req_o,
    ack_i    => na20_ack_i,
    ack_o    => na20_ack_o);

node_30 : mesh_router
```

```
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_20_30,
    link_w_bw_o => link_bw_20_30,
    link_n_fw_i => link_fw_31_30,
    link_n_bw_o => link_bw_31_30,
    link_e_fw_i => link_fw_gnd,
    link_e_bw_o => open,
    link_s_fw_i => link_fw_gnd,
    link_s_bw_o => open,
    link_a_fw_i => link_fw_na_30,
    link_a_bw_o => link_bw_na_30,
    -- Output links
    link_w_fw_o => link_fw_30_20,
    link_w_bw_i => link_bw_30_20,
    link_n_fw_o => link_fw_30_31,
    link_n_bw_i => link_bw_30_31,
    link_e_fw_o => open,
    link_e_bw_i => link_bw_high,
    link_s_fw_o => open,
    link_s_bw_i => link_bw_high,
    link_a_fw_o => link_fw_30_na,
    link_a_bw_i => link_bw_30_na);

na_30 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_30_na,
    noc_bw_o => link_bw_30_na,
    noc_fw_o => link_fw_na_30,
    noc_bw_i => link_bw_na_30,
    clk      => clk,
    rst      => rst,
    data_i   => na30_data_i,
    data_o   => na30_data_o,
    addr_i   => na30_addr_i,
    type_i   => na30_type_i,
    req_i    => na30_req_i,
    req_o    => na30_req_o,
    ack_i    => na30_ack_i,
    ack_o    => na30_ack_o);

node_01 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_gnd,
    link_w_bw_o => open,
    link_n_fw_i => link_fw_02_01,
    link_n_bw_o => link_bw_02_01,
    link_e_fw_i => link_fw_11_01,
    link_e_bw_o => link_bw_11_01,
    link_s_fw_i => link_fw_00_01,
    link_s_bw_o => link_bw_00_01,
    link_a_fw_i => link_fw_na_01,
    link_a_bw_o => link_bw_na_01,
```

```
    -- Output links
    link_w_fw_o => open,
    link_w_bw_i => link_bw_high,
    link_n_fw_o => link_fw_01_02,
    link_n_bw_i => link_bw_01_02,
    link_e_fw_o => link_fw_01_11,
    link_e_bw_i => link_bw_01_11,
    link_s_fw_o => link_fw_01_00,
    link_s_bw_i => link_bw_01_00,
    link_a_fw_o => link_fw_01_na,
    link_a_bw_i => link_bw_01_na);

na_01 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_01_na,
    noc_bw_o => link_bw_01_na,
    noc_fw_o => link_fw_na_01,
    noc_bw_i => link_bw_na_01,
    clk      => clk,
    rst      => rst,
    data_i   => na01_data_i,
    data_o   => na01_data_o,
    addr_i   => na01_addr_i,
    type_i   => na01_type_i,
    req_i    => na01_req_i,
    req_o    => na01_req_o,
    ack_i    => na01_ack_i,
    ack_o    => na01_ack_o);

node_11 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_01_11,
    link_w_bw_o => link_bw_01_11,
    link_n_fw_i => link_fw_12_11,
    link_n_bw_o => link_bw_12_11,
    link_e_fw_i => link_fw_21_11,
    link_e_bw_o => link_bw_21_11,
    link_s_fw_i => link_fw_10_11,
    link_s_bw_o => link_bw_10_11,
    link_a_fw_i => link_fw_na_11,
    link_a_bw_o => link_bw_na_11,
    -- Output links
    link_w_fw_o => link_fw_11_01,
    link_w_bw_i => link_bw_11_01,
    link_n_fw_o => link_fw_11_12,
    link_n_bw_i => link_bw_11_12,
    link_e_fw_o => link_fw_11_21,
    link_e_bw_i => link_bw_11_21,
    link_s_fw_o => link_fw_11_10,
    link_s_bw_i => link_bw_11_10,
    link_a_fw_o => link_fw_11_na,
    link_a_bw_i => link_bw_11_na);

na_11 : mesh_na
  generic map(
```

```
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_11_na,
    noc_bw_o => link_bw_11_na,
    noc_fw_o => link_fw_na_11,
    noc_bw_i => link_bw_na_11,
    clk      => clk,
    rst      => rst,
    data_i   => na11_data_i,
    data_o   => na11_data_o,
    addr_i   => na11_addr_i,
    type_i   => na11_type_i,
    req_i    => na11_req_i,
    req_o    => na11_req_o,
    ack_i    => na11_ack_i,
    ack_o    => na11_ack_o);

node_21 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_11_21,
    link_w_bw_o => link_bw_11_21,
    link_n_fw_i => link_fw_22_21,
    link_n_bw_o => link_bw_22_21,
    link_e_fw_i => link_fw_31_21,
    link_e_bw_o => link_bw_31_21,
    link_s_fw_i => link_fw_20_21,
    link_s_bw_o => link_bw_20_21,
    link_a_fw_i => link_fw_na_21,
    link_a_bw_o => link_bw_na_21,
    -- Output links
    link_w_fw_o => link_fw_21_11,
    link_w_bw_i => link_bw_21_11,
    link_n_fw_o => link_fw_21_22,
    link_n_bw_i => link_bw_21_22,
    link_e_fw_o => link_fw_21_31,
    link_e_bw_i => link_bw_21_31,
    link_s_fw_o => link_fw_21_20,
    link_s_bw_i => link_bw_21_20,
    link_a_fw_o => link_fw_21_na,
    link_a_bw_i => link_bw_21_na);

na_21 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_21_na,
    noc_bw_o => link_bw_21_na,
    noc_fw_o => link_fw_na_21,
    noc_bw_i => link_bw_na_21,
    clk      => clk,
    rst      => rst,
    data_i   => na21_data_i,
    data_o   => na21_data_o,
    addr_i   => na21_addr_i,
    type_i   => na21_type_i,
```

```
    req_i    => na21_req_i,
    req_o    => na21_req_o,
    ack_i    => na21_ack_i,
    ack_o    => na21_ack_o);

node_31 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_21_31,
    link_w_bw_o => link_bw_21_31,
    link_n_fw_i => link_fw_32_31,
    link_n_bw_o => link_bw_32_31,
    link_e_fw_i => link_fw_gnd,
    link_e_bw_o => open,
    link_s_fw_i => link_fw_30_31,
    link_s_bw_o => link_bw_30_31,
    link_a_fw_i => link_fw_na_31,
    link_a_bw_o => link_bw_na_31,
    -- Output links
    link_w_fw_o => link_fw_31_21,
    link_w_bw_i => link_bw_31_21,
    link_n_fw_o => link_fw_31_32,
    link_n_bw_i => link_bw_31_32,
    link_e_fw_o => open,
    link_e_bw_i => link_bw_high,
    link_s_fw_o => link_fw_31_30,
    link_s_bw_i => link_bw_31_30,
    link_a_fw_o => link_fw_31_na,
    link_a_bw_i => link_bw_31_na);

na_31 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_31_na,
    noc_bw_o => link_bw_31_na,
    noc_fw_o => link_fw_na_31,
    noc_bw_i => link_bw_na_31,
    clk      => clk,
    rst      => rst,
    data_i   => na31_data_i,
    data_o   => na31_data_o,
    addr_i   => na31_addr_i,
    type_i   => na31_type_i,
    req_i    => na31_req_i,
    req_o    => na31_req_o,
    ack_i    => na31_ack_i,
    ack_o    => na31_ack_o);

node_02 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_gnd,
    link_w_bw_o => open,
    link_n_fw_i => link_fw_03_02,
    link_n_bw_o => link_bw_03_02,
```

```
    link_e_fw_i => link_fw_12_02,
    link_e_bw_o => link_bw_12_02,
    link_s_fw_i => link_fw_01_02,
    link_s_bw_o => link_bw_01_02,
    link_a_fw_i => link_fw_na_02,
    link_a_bw_o => link_bw_na_02,
    -- Output links
    link_w_fw_o => open,
    link_w_bw_i => link_bw_high,
    link_n_fw_o => link_fw_02_03,
    link_n_bw_i => link_bw_02_03,
    link_e_fw_o => link_fw_02_12,
    link_e_bw_i => link_bw_02_12,
    link_s_fw_o => link_fw_02_01,
    link_s_bw_i => link_bw_02_01,
    link_a_fw_o => link_fw_02_na,
    link_a_bw_i => link_bw_02_na);

na_02 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_02_na,
    noc_bw_o => link_bw_02_na,
    noc_fw_o => link_fw_na_02,
    noc_bw_i => link_bw_na_02,
    clk      => clk,
    rst      => rst,
    data_i   => na02_data_i,
    data_o   => na02_data_o,
    addr_i   => na02_addr_i,
    type_i   => na02_type_i,
    req_i    => na02_req_i,
    req_o    => na02_req_o,
    ack_i    => na02_ack_i,
    ack_o    => na02_ack_o);

node_12 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_02_12,
    link_w_bw_o => link_bw_02_12,
    link_n_fw_i => link_fw_13_12,
    link_n_bw_o => link_bw_13_12,
    link_e_fw_i => link_fw_22_12,
    link_e_bw_o => link_bw_22_12,
    link_s_fw_i => link_fw_11_12,
    link_s_bw_o => link_bw_11_12,
    link_a_fw_i => link_fw_na_12,
    link_a_bw_o => link_bw_na_12,
    -- Output links
    link_w_fw_o => link_fw_12_02,
    link_w_bw_i => link_bw_12_02,
    link_n_fw_o => link_fw_12_13,
    link_n_bw_i => link_bw_12_13,
    link_e_fw_o => link_fw_12_22,
    link_e_bw_i => link_bw_12_22,
    link_s_fw_o => link_fw_12_11,
```

```
    link_s_bw_i => link_bw_12_11,
    link_a_fw_o => link_fw_12_na,
    link_a_bw_i => link_bw_12_na);

na_12 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_12_na,
    noc_bw_o => link_bw_12_na,
    noc_fw_o => link_fw_na_12,
    noc_bw_i => link_bw_na_12,
    clk      => clk,
    rst      => rst,
    data_i   => na12_data_i,
    data_o   => na12_data_o,
    addr_i   => na12_addr_i,
    type_i   => na12_type_i,
    req_i    => na12_req_i,
    req_o    => na12_req_o,
    ack_i    => na12_ack_i,
    ack_o    => na12_ack_o);

node_22 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_12_22,
    link_w_bw_o => link_bw_12_22,
    link_n_fw_i => link_fw_23_22,
    link_n_bw_o => link_bw_23_22,
    link_e_fw_i => link_fw_32_22,
    link_e_bw_o => link_bw_32_22,
    link_s_fw_i => link_fw_21_22,
    link_s_bw_o => link_bw_21_22,
    link_a_fw_i => link_fw_na_22,
    link_a_bw_o => link_bw_na_22,
    -- Output links
    link_w_fw_o => link_fw_22_12,
    link_w_bw_i => link_bw_22_12,
    link_n_fw_o => link_fw_22_23,
    link_n_bw_i => link_bw_22_23,
    link_e_fw_o => link_fw_22_32,
    link_e_bw_i => link_bw_22_32,
    link_s_fw_o => link_fw_22_21,
    link_s_bw_i => link_bw_22_21,
    link_a_fw_o => link_fw_22_na,
    link_a_bw_i => link_bw_22_na);

na_22 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_22_na,
    noc_bw_o => link_bw_22_na,
    noc_fw_o => link_fw_na_22,
    noc_bw_i => link_bw_na_22,
```

```
    clk      => clk,
    rst      => rst,
    data_i   => na22_data_i,
    data_o   => na22_data_o,
    addr_i   => na22_addr_i,
    type_i   => na22_type_i,
    req_i    => na22_req_i,
    req_o    => na22_req_o,
    ack_i    => na22_ack_i,
    ack_o    => na22_ack_o);

node_32 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_22_32,
    link_w_bw_o => link_bw_22_32,
    link_n_fw_i => link_fw_33_32,
    link_n_bw_o => link_bw_33_32,
    link_e_fw_i => link_fw_gnd,
    link_e_bw_o => open,
    link_s_fw_i => link_fw_31_32,
    link_s_bw_o => link_bw_31_32,
    link_a_fw_i => link_fw_na_32,
    link_a_bw_o => link_bw_na_32,
    -- Output links
    link_w_fw_o => link_fw_32_22,
    link_w_bw_i => link_bw_32_22,
    link_n_fw_o => link_fw_32_33,
    link_n_bw_i => link_bw_32_33,
    link_e_fw_o => open,
    link_e_bw_i => link_bw_high,
    link_s_fw_o => link_fw_32_31,
    link_s_bw_i => link_bw_32_31,
    link_a_fw_o => link_fw_32_na,
    link_a_bw_i => link_bw_32_na);

na_32 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_32_na,
    noc_bw_o => link_bw_32_na,
    noc_fw_o => link_fw_na_32,
    noc_bw_i => link_bw_na_32,
    clk      => clk,
    rst      => rst,
    data_i   => na32_data_i,
    data_o   => na32_data_o,
    addr_i   => na32_addr_i,
    type_i   => na32_type_i,
    req_i    => na32_req_i,
    req_o    => na32_req_o,
    ack_i    => na32_ack_i,
    ack_o    => na32_ack_o);

node_03 : mesh_router
  port map (
    clk         => clk,
```

```
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_gnd,
    link_w_bw_o => open,
    link_n_fw_i => link_fw_gnd,
    link_n_bw_o => open,
    link_e_fw_i => link_fw_13_03,
    link_e_bw_o => link_bw_13_03,
    link_s_fw_i => link_fw_02_03,
    link_s_bw_o => link_bw_02_03,
    link_a_fw_i => link_fw_na_03,
    link_a_bw_o => link_bw_na_03,
    -- Output links
    link_w_fw_o => open,
    link_w_bw_i => link_bw_high,
    link_n_fw_o => open,
    link_n_bw_i => link_bw_high,
    link_e_fw_o => link_fw_03_13,
    link_e_bw_i => link_bw_03_13,
    link_s_fw_o => link_fw_03_02,
    link_s_bw_i => link_bw_03_02,
    link_a_fw_o => link_fw_03_na,
    link_a_bw_i => link_bw_03_na);

na_03 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_03_na,
    noc_bw_o => link_bw_03_na,
    noc_fw_o => link_fw_na_03,
    noc_bw_i => link_bw_na_03,
    clk      => clk,
    rst      => rst,
    data_i   => na03_data_i,
    data_o   => na03_data_o,
    addr_i   => na03_addr_i,
    type_i   => na03_type_i,
    req_i    => na03_req_i,
    req_o    => na03_req_o,
    ack_i    => na03_ack_i,
    ack_o    => na03_ack_o);

node_13 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_03_13,
    link_w_bw_o => link_bw_03_13,
    link_n_fw_i => link_fw_gnd,
    link_n_bw_o => open,
    link_e_fw_i => link_fw_23_13,
    link_e_bw_o => link_bw_23_13,
    link_s_fw_i => link_fw_12_13,
    link_s_bw_o => link_bw_12_13,
    link_a_fw_i => link_fw_na_13,
    link_a_bw_o => link_bw_na_13,
    -- Output links
    link_w_fw_o => link_fw_13_03,
```

```
    link_w_bw_i => link_bw_13_03,
    link_n_fw_o => open,
    link_n_bw_i => link_bw_high,
    link_e_fw_o => link_fw_13_23,
    link_e_bw_i => link_bw_13_23,
    link_s_fw_o => link_fw_13_12,
    link_s_bw_i => link_bw_13_12,
    link_a_fw_o => link_fw_13_na,
    link_a_bw_i => link_bw_13_na);

na_13 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_13_na,
    noc_bw_o => link_bw_13_na,
    noc_fw_o => link_fw_na_13,
    noc_bw_i => link_bw_na_13,
    clk      => clk,
    rst      => rst,
    data_i   => na13_data_i,
    data_o   => na13_data_o,
    addr_i   => na13_addr_i,
    type_i   => na13_type_i,
    req_i    => na13_req_i,
    req_o    => na13_req_o,
    ack_i    => na13_ack_i,
    ack_o    => na13_ack_o);

node_23 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_13_23,
    link_w_bw_o => link_bw_13_23,
    link_n_fw_i => link_fw_gnd,
    link_n_bw_o => open,
    link_e_fw_i => link_fw_33_23,
    link_e_bw_o => link_bw_33_23,
    link_s_fw_i => link_fw_22_23,
    link_s_bw_o => link_bw_22_23,
    link_a_fw_i => link_fw_na_23,
    link_a_bw_o => link_bw_na_23,
    -- Output links
    link_w_fw_o => link_fw_23_13,
    link_w_bw_i => link_bw_23_13,
    link_n_fw_o => open,
    link_n_bw_i => link_bw_high,
    link_e_fw_o => link_fw_23_33,
    link_e_bw_i => link_bw_23_33,
    link_s_fw_o => link_fw_23_22,
    link_s_bw_i => link_bw_23_22,
    link_a_fw_o => link_fw_23_na,
    link_a_bw_i => link_bw_23_na);

na_23 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
```

```
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_23_na,
    noc_bw_o => link_bw_23_na,
    noc_fw_o => link_fw_na_23,
    noc_bw_i => link_bw_na_23,
    clk      => clk,
    rst      => rst,
    data_i   => na23_data_i,
    data_o   => na23_data_o,
    addr_i   => na23_addr_i,
    type_i   => na23_type_i,
    req_i    => na23_req_i,
    req_o    => na23_req_o,
    ack_i    => na23_ack_i,
    ack_o    => na23_ack_o);

node_33 : mesh_router
  port map (
    clk         => clk,
    rst         => rst,
    -- Input links
    link_w_fw_i => link_fw_23_33,
    link_w_bw_o => link_bw_23_33,
    link_n_fw_i => link_fw_gnd,
    link_n_bw_o => open,
    link_e_fw_i => link_fw_gnd,
    link_e_bw_o => open,
    link_s_fw_i => link_fw_32_33,
    link_s_bw_o => link_bw_32_33,
    link_a_fw_i => link_fw_na_33,
    link_a_bw_o => link_bw_na_33,
    -- Output links
    link_w_fw_o => link_fw_33_23,
    link_w_bw_i => link_bw_33_23,
    link_n_fw_o => open,
    link_n_bw_i => link_bw_high,
    link_e_fw_o => open,
    link_e_bw_i => link_bw_high,
    link_s_fw_o => link_fw_33_32,
    link_s_bw_i => link_bw_33_32,
    link_a_fw_o => link_fw_33_na,
    link_a_bw_i => link_bw_33_na);

na_33 : mesh_na
  generic map(
    ADDR_BITS    =>  2,
    DATA_WIDTH_I =>  24,
    DATA_WIDTH_O =>  24)
  port map(
    noc_fw_i => link_fw_33_na,
    noc_bw_o => link_bw_33_na,
    noc_fw_o => link_fw_na_33,
    noc_bw_i => link_bw_na_33,
    clk      => clk,
    rst      => rst,
    data_i   => na33_data_i,
    data_o   => na33_data_o,
    addr_i   => na33_addr_i,
    type_i   => na33_type_i,
    req_i    => na33_req_i,
    req_o    => na33_req_o,
```

```
    ack_i    => na33_ack_i,
    ack_o    => na33_ack_o);

end arch_4x4;
```

C.3.2 Global settings

```
-------------------------------------------------------------------------------
-- Types - Global definitions - Mesh NoC
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

package types is
  constant FLIT_WIDTH : integer := 40;  -- Flit width
  constant EOP_BIT_POS : integer := 39;  -- EOP bit position
  constant ADDR_HOPS : integer := 7;    -- Number of hops in header
  constant HOP_BITS : integer := 2;     -- Bit pr. hop
  constant TYPE_BITS : integer := 1;    -- Bits indicating type
  constant TYPE_BIT_POS : integer := 38;    -- Bits indicating type
  constant FLIT_DATA_WIDTH : integer := 38;    -- Data width
  constant HDR_ADDR_WIDTH : integer := ADDR_HOPS * HOP_BITS;
                                        -- Header bits for routing information
  constant HDR_ADDR_OFFSET : integer := 24;  -- Address position
  constant VC : integer := 2;           -- Number of virtual channels
  constant VC_bit : integer := 1;       -- log2(VC)
  constant DV_bit : integer := 41;      -- Data valid bit position

  subtype noc_link_fw is std_logic_vector((FLIT_WIDTH + VC_bit) downto 0);
  subtype noc_link_bw is std_logic_vector (VC-1 downto 0);
                                        -- Network acknowledge link
  subtype vc_data is std_logic_vector(FLIT_WIDTH-1 downto 0);  -- VC data

  subtype rt_direction is std_logic_vector(2 downto 0);  -- Routing direction

  constant rt_n : rt_direction := "000";  -- North
  constant rt_e : rt_direction := "001";  -- East
  constant rt_s : rt_direction := "011";  -- South
  constant rt_w : rt_direction := "010";  -- West
  constant rt_a : rt_direction := "100";  -- Adapter
  constant rt_x : rt_direction := "111";  -- Nowhere

end types;
```

C.3.3 Main testbench

```
-------------------------------------------------------------------------------
-- TB Mesh 4x4 clean - Mesh NoC main testbench
-- by Morten Sleth Rasmussen, s011295
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.types.all;

entity tb_mesh_4x4_clean is

end tb_mesh_4x4_clean;
```

```
architecture clean_4x4 of tb_mesh_4x4_clean is

component mesh_4x4
  port (
    clk : in std_logic;                 -- Clock
    rst : in std_logic;                 -- Reset

      -- Test enable pin, that will be inserted by DC
      -- RTL_SYNTHESIS OFF
      -- pragma synthesis_off
      test_se    : in  std_logic;   -- syncheck_off
      -- pragma synthesis_on
      -- RTL_SYNTHESIS ON


-- Programming unit
pgm_data_i : in  noc_link_fw;
pgm_data_o : out  noc_link_bw;

-- Adapter 00
na00_data_i : in  std_logic_vector (23 downto 0);
na00_data_o : out std_logic_vector (23 downto 0);
na00_addr_i : in  std_logic_vector (1 downto 0);
na00_type_i : in  std_logic_vector (0 downto 0);
na00_req_i  : in  std_logic;
na00_req_o  : out std_logic;
na00_ack_i  : in  std_logic;
na00_ack_o  : out std_logic;
-- Adapter 01
na01_data_i : in  std_logic_vector (23 downto 0);
na01_data_o : out std_logic_vector (23 downto 0);
na01_addr_i : in  std_logic_vector (1 downto 0);
na01_type_i : in  std_logic_vector (0 downto 0);
na01_req_i  : in  std_logic;
na01_req_o  : out std_logic;
na01_ack_i  : in  std_logic;
na01_ack_o  : out std_logic;
-- Adapter 02
na02_data_i : in  std_logic_vector (23 downto 0);
na02_data_o : out std_logic_vector (23 downto 0);
na02_addr_i : in  std_logic_vector (1 downto 0);
na02_type_i : in  std_logic_vector (0 downto 0);
na02_req_i  : in  std_logic;
na02_req_o  : out std_logic;
na02_ack_i  : in  std_logic;
na02_ack_o  : out std_logic;
-- Adapter 03
na03_data_i : in  std_logic_vector (23 downto 0);
na03_data_o : out std_logic_vector (23 downto 0);
na03_addr_i : in  std_logic_vector (1 downto 0);
na03_type_i : in  std_logic_vector (0 downto 0);
na03_req_i  : in  std_logic;
na03_req_o  : out std_logic;
na03_ack_i  : in  std_logic;
na03_ack_o  : out std_logic;
-- Adapter 10
na10_data_i : in  std_logic_vector (23 downto 0);
na10_data_o : out std_logic_vector (23 downto 0);
na10_addr_i : in  std_logic_vector (1 downto 0);
na10_type_i : in  std_logic_vector (0 downto 0);
na10_req_i  : in  std_logic;
```

```
na10_req_o  : out std_logic;
na10_ack_i  : in  std_logic;
na10_ack_o  : out std_logic;
-- Adapter 11
na11_data_i : in  std_logic_vector (23 downto 0);
na11_data_o : out std_logic_vector (23 downto 0);
na11_addr_i : in  std_logic_vector (1 downto 0);
na11_type_i : in  std_logic_vector (0 downto 0);
na11_req_i  : in  std_logic;
na11_req_o  : out std_logic;
na11_ack_i  : in  std_logic;
na11_ack_o  : out std_logic;
-- Adapter 12
na12_data_i : in  std_logic_vector (23 downto 0);
na12_data_o : out std_logic_vector (23 downto 0);
na12_addr_i : in  std_logic_vector (1 downto 0);
na12_type_i : in  std_logic_vector (0 downto 0);
na12_req_i  : in  std_logic;
na12_req_o  : out std_logic;
na12_ack_i  : in  std_logic;
na12_ack_o  : out std_logic;
-- Adapter 13
na13_data_i : in  std_logic_vector (23 downto 0);
na13_data_o : out std_logic_vector (23 downto 0);
na13_addr_i : in  std_logic_vector (1 downto 0);
na13_type_i : in  std_logic_vector (0 downto 0);
na13_req_i  : in  std_logic;
na13_req_o  : out std_logic;
na13_ack_i  : in  std_logic;
na13_ack_o  : out std_logic;
-- Adapter 20
na20_data_i : in  std_logic_vector (23 downto 0);
na20_data_o : out std_logic_vector (23 downto 0);
na20_addr_i : in  std_logic_vector (1 downto 0);
na20_type_i : in  std_logic_vector (0 downto 0);
na20_req_i  : in  std_logic;
na20_req_o  : out std_logic;
na20_ack_i  : in  std_logic;
na20_ack_o  : out std_logic;
-- Adapter 21
na21_data_i : in  std_logic_vector (23 downto 0);
na21_data_o : out std_logic_vector (23 downto 0);
na21_addr_i : in  std_logic_vector (1 downto 0);
na21_type_i : in  std_logic_vector (0 downto 0);
na21_req_i  : in  std_logic;
na21_req_o  : out std_logic;
na21_ack_i  : in  std_logic;
na21_ack_o  : out std_logic;
-- Adapter 22
na22_data_i : in  std_logic_vector (23 downto 0);
na22_data_o : out std_logic_vector (23 downto 0);
na22_addr_i : in  std_logic_vector (1 downto 0);
na22_type_i : in  std_logic_vector (0 downto 0);
na22_req_i  : in  std_logic;
na22_req_o  : out std_logic;
na22_ack_i  : in  std_logic;
na22_ack_o  : out std_logic;
-- Adapter 23
na23_data_i : in  std_logic_vector (23 downto 0);
na23_data_o : out std_logic_vector (23 downto 0);
na23_addr_i : in  std_logic_vector (1 downto 0);
na23_type_i : in  std_logic_vector (0 downto 0);
```

```
na23_req_i  : in  std_logic;
na23_req_o  : out std_logic;
na23_ack_i  : in  std_logic;
na23_ack_o  : out std_logic;
-- Adapter 30
na30_data_i : in  std_logic_vector (23 downto 0);
na30_data_o : out std_logic_vector (23 downto 0);
na30_addr_i : in  std_logic_vector (1 downto 0);
na30_type_i : in  std_logic_vector (0 downto 0);
na30_req_i  : in  std_logic;
na30_req_o  : out std_logic;
na30_ack_i  : in  std_logic;
na30_ack_o  : out std_logic;
-- Adapter 31
na31_data_i : in  std_logic_vector (23 downto 0);
na31_data_o : out std_logic_vector (23 downto 0);
na31_addr_i : in  std_logic_vector (1 downto 0);
na31_type_i : in  std_logic_vector (0 downto 0);
na31_req_i  : in  std_logic;
na31_req_o  : out std_logic;
na31_ack_i  : in  std_logic;
na31_ack_o  : out std_logic;
-- Adapter 32
na32_data_i : in  std_logic_vector (23 downto 0);
na32_data_o : out std_logic_vector (23 downto 0);
na32_addr_i : in  std_logic_vector (1 downto 0);
na32_type_i : in  std_logic_vector (0 downto 0);
na32_req_i  : in  std_logic;
na32_req_o  : out std_logic;
na32_ack_i  : in  std_logic;
na32_ack_o  : out std_logic;
-- Adapter 33
na33_data_i : in  std_logic_vector (23 downto 0);
na33_data_o : out std_logic_vector (23 downto 0);
na33_addr_i : in  std_logic_vector (1 downto 0);
na33_type_i : in  std_logic_vector (0 downto 0);
na33_req_i  : in  std_logic;
na33_req_o  : out std_logic;
na33_ack_i  : in  std_logic;
na33_ack_o  : out std_logic);

end component;

component file_log_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH downto 0));
end component;

component file_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
```

```
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

component file_cc_read_std_logic
  generic (
    filename   : string;
    DATA_WIDTH : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    dv     : out std_logic;
    data_o : out std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

component pgm_unit
  generic (
    filename   : string);
  port (
    noc_fw_o : out noc_link_fw;
    noc_bw_i : in  noc_link_bw;
    clk      : in  std_logic;
    rst      : in  std_logic);
end component;

component gen_traffic
  generic (
    conf_file   : string;
    dump_file   : string;
    period      : integer;
    ADDR_BITS   : integer;
    DATA_WIDTH_I : integer;
    DATA_WIDTH_O : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in  std_logic_vector(DATA_WIDTH_I-1 downto 0);
    data_o : out std_logic_vector(DATA_WIDTH_O-1 downto 0);
    addr_o : out std_logic_vector(ADDR_BITS-1 downto 0);
    type_o : out std_logic_vector(TYPE_BITS-1 downto 0);
    req_i  : in  std_logic;
    req_o  : out std_logic;
    ack_i  : in  std_logic;
    ack_o  : out std_logic);
end component;

component stat_log
  generic (
    filename   : string;
    DATA_WIDTH : integer;
    PERIOD     : integer;
    INIT_DELAY : integer);
  port (
    clk    : in std_logic;
    rst    : in std_logic;
    data_i : in std_logic_vector(DATA_WIDTH-1 downto 0));
end component;

signal clk, rst : std_logic := '0';
signal test_se : std_logic;

signal na00_data_i, na10_data_i, na20_data_i, na30_data_i, na01_data_i,
  na11_data_i, na21_data_i, na31_data_i, na22_data_i, na32_data_i,
```

```
  na03_data_i, na13_data_i, na23_data_i, na33_data_i, na00_data_o,
  na20_data_o, na30_data_o, na01_data_o, na11_data_o, na21_data_o,
  na31_data_o, na22_data_o, na32_data_o,  na03_data_o, na13_data_o,
  na23_data_o, na33_data_o : std_logic_vector(23 downto 0);
signal na10_data_o : std_logic_vector(23 downto 0);
signal na02_data_i, na12_data_i : std_logic_vector(23 downto 0);
signal na02_data_o, na12_data_o : std_logic_vector(23 downto 0);
signal na00_addr_i, na10_addr_i, na20_addr_i, na30_addr_i, na01_addr_i,
  na11_addr_i, na21_addr_i, na31_addr_i, na02_addr_i, na12_addr_i,
  na22_addr_i, na32_addr_i, na03_addr_i, na13_addr_i, na23_addr_i,
  na33_addr_i : std_logic_vector(1 downto 0);
signal na00_type_i, na10_type_i, na20_type_i, na30_type_i, na01_type_i,
  na11_type_i, na21_type_i, na31_type_i, na02_type_i, na12_type_i,
  na22_type_i, na32_type_i, na03_type_i, na13_type_i, na23_type_i,
  na33_type_i : std_logic_vector(TYPE_BITS-1 downto 0);
signal na00_req_i, na10_req_i, na20_req_i, na30_req_i, na01_req_i,
  na11_req_i, na21_req_i, na31_req_i, na02_req_i, na12_req_i, na22_req_i,
  na32_req_i, na03_req_i, na13_req_i, na23_req_i, na33_req_i, na00_req_o,
  na10_req_o, na20_req_o, na30_req_o, na01_req_o, na11_req_o, na21_req_o,
  na31_req_o, na02_req_o, na12_req_o, na22_req_o, na32_req_o, na03_req_o,
  na13_req_o, na23_req_o, na33_req_o : std_logic;
signal na00_ack_i, na10_ack_i, na20_ack_i, na30_ack_i, na01_ack_i,
  na11_ack_i, na21_ack_i, na31_ack_i, na02_ack_i, na12_ack_i, na22_ack_i,
  na32_ack_i, na03_ack_i, na13_ack_i, na23_ack_i, na33_ack_i, na00_ack_o,
  na10_ack_o, na20_ack_o, na30_ack_o, na01_ack_o, na11_ack_o, na21_ack_o,
  na31_ack_o, na02_ack_o, na12_ack_o, na22_ack_o, na32_ack_o, na03_ack_o,
  na13_ack_o, na23_ack_o, na33_ack_o : std_logic;

signal link_input_0_fw : noc_link_fw;
signal link_input_0_bw : noc_link_bw;

begin  -- test

  test_se <= '0';
  clk <= not clk after 250 ns;
  rst <= '1' after 2000 ns;

noc :  mesh_4x4
  port map(
    clk => clk,
    rst => rst,
pgm_data_i => link_input_0_fw,
pgm_data_o => link_input_0_bw,
test_se => test_se,

na00_data_i => na00_data_i,
na00_data_o => na00_data_o,
na00_addr_i => na00_addr_i,
na00_type_i => na00_type_i,
na00_req_i  => na00_req_i ,
na00_req_o  => na00_req_o ,
na00_ack_i  => na00_ack_i ,
na00_ack_o  => na00_ack_o ,
na01_data_i => na01_data_i,
na01_data_o => na01_data_o,
na01_addr_i => na01_addr_i,
na01_type_i => na01_type_i,
na01_req_i  => na01_req_i ,
na01_req_o  => na01_req_o ,
na01_ack_i  => na01_ack_i ,
na01_ack_o  => na01_ack_o ,
na02_data_i => na02_data_i,
```

```
na02_data_o => na02_data_o,
na02_addr_i => na02_addr_i,
na02_type_i => na02_type_i,
na02_req_i  => na02_req_i ,
na02_req_o  => na02_req_o ,
na02_ack_i  => na02_ack_i ,
na02_ack_o  => na02_ack_o ,
na03_data_i => na03_data_i,
na03_data_o => na03_data_o,
na03_addr_i => na03_addr_i,
na03_type_i => na03_type_i,
na03_req_i  => na03_req_i ,
na03_req_o  => na03_req_o ,
na03_ack_i  => na03_ack_i ,
na03_ack_o  => na03_ack_o ,
na10_data_i => na10_data_i,
na10_data_o => na10_data_o,
na10_addr_i => na10_addr_i,
na10_type_i => na10_type_i,
na10_req_i  => na10_req_i ,
na10_req_o  => na10_req_o ,
na10_ack_i  => na10_ack_i ,
na10_ack_o  => na10_ack_o ,
na11_data_i => na11_data_i,
na11_data_o => na11_data_o,
na11_addr_i => na11_addr_i,
na11_type_i => na11_type_i,
na11_req_i  => na11_req_i ,
na11_req_o  => na11_req_o ,
na11_ack_i  => na11_ack_i ,
na11_ack_o  => na11_ack_o ,
na12_data_i => na12_data_i,
na12_data_o => na12_data_o,
na12_addr_i => na12_addr_i,
na12_type_i => na12_type_i,
na12_req_i  => na12_req_i ,
na12_req_o  => na12_req_o ,
na12_ack_i  => na12_ack_i ,
na12_ack_o  => na12_ack_o ,
na13_data_i => na13_data_i,
na13_data_o => na13_data_o,
na13_addr_i => na13_addr_i,
na13_type_i => na13_type_i,
na13_req_i  => na13_req_i ,
na13_req_o  => na13_req_o ,
na13_ack_i  => na13_ack_i ,
na13_ack_o  => na13_ack_o ,
na20_data_i => na20_data_i,
na20_data_o => na20_data_o,
na20_addr_i => na20_addr_i,
na20_type_i => na20_type_i,
na20_req_i  => na20_req_i ,
na20_req_o  => na20_req_o ,
na20_ack_i  => na20_ack_i ,
na20_ack_o  => na20_ack_o ,
na21_data_i => na21_data_i,
na21_data_o => na21_data_o,
na21_addr_i => na21_addr_i,
na21_type_i => na21_type_i,
na21_req_i  => na21_req_i ,
na21_req_o  => na21_req_o ,
na21_ack_i  => na21_ack_i ,
```

```
na21_ack_o  => na21_ack_o ,
na22_data_i => na22_data_i,
na22_data_o => na22_data_o,
na22_addr_i => na22_addr_i,
na22_type_i => na22_type_i,
na22_req_i  => na22_req_i ,
na22_req_o  => na22_req_o ,
na22_ack_i  => na22_ack_i ,
na22_ack_o  => na22_ack_o ,
na23_data_i => na23_data_i,
na23_data_o => na23_data_o,
na23_addr_i => na23_addr_i,
na23_type_i => na23_type_i,
na23_req_i  => na23_req_i ,
na23_req_o  => na23_req_o ,
na23_ack_i  => na23_ack_i ,
na23_ack_o  => na23_ack_o ,
na30_data_i => na30_data_i,
na30_data_o => na30_data_o,
na30_addr_i => na30_addr_i,
na30_type_i => na30_type_i,
na30_req_i  => na30_req_i ,
na30_req_o  => na30_req_o ,
na30_ack_i  => na30_ack_i ,
na30_ack_o  => na30_ack_o ,
na31_data_i => na31_data_i,
na31_data_o => na31_data_o,
na31_addr_i => na31_addr_i,
na31_type_i => na31_type_i,
na31_req_i  => na31_req_i ,
na31_req_o  => na31_req_o ,
na31_ack_i  => na31_ack_i ,
na31_ack_o  => na31_ack_o ,
na32_data_i => na32_data_i,
na32_data_o => na32_data_o,
na32_addr_i => na32_addr_i,
na32_type_i => na32_type_i,
na32_req_i  => na32_req_i ,
na32_req_o  => na32_req_o ,
na32_ack_i  => na32_ack_i ,
na32_ack_o  => na32_ack_o ,
na33_data_i => na33_data_i,
na33_data_o => na33_data_o,
na33_addr_i => na33_addr_i,
na33_type_i => na33_type_i,
na33_req_i  => na33_req_i ,
na33_req_o  => na33_req_o ,
na33_ack_i  => na33_ack_i ,
na33_ack_o  => na33_ack_o);

traffic00 : gen_traffic
  generic map(
    conf_file   => "8.txt",
    dump_file   => "8_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na00_data_o,
```

```
    data_o => na00_data_i,
    addr_o => na00_addr_i,
    type_o => na00_type_i,
    req_i  => na00_req_o,
    req_o  => na00_req_i,
    ack_i  => na00_ack_o,
    ack_o  => na00_ack_i);

traffic10 : gen_traffic
  generic map(
    conf_file   => "empty.txt",
    dump_file   => "12_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na10_data_o,
    data_o => na10_data_i,
    addr_o => na10_addr_i,
    type_o => na10_type_i,
    req_i  => na10_req_o,
    req_o  => na10_req_i,
    ack_i  => na10_ack_o,
    ack_o  => na10_ack_i);


traffic20 : gen_traffic
  generic map(
    conf_file   => "11.txt",
    dump_file   => "11_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na20_data_o,
    data_o => na20_data_i,
    addr_o => na20_addr_i,
    type_o => na20_type_i,
    req_i  => na20_req_o,
    req_o  => na20_req_i,
    ack_i  => na20_ack_o,
    ack_o  => na20_ack_i);

traffic30 : gen_traffic
  generic map(
    conf_file   => "10.txt",
    dump_file   => "10_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na30_data_o,
    data_o => na30_data_i,
```

```
    addr_o => na30_addr_i,
    type_o => na30_type_i,
    req_i  => na30_req_o,
    req_o  => na30_req_i,
    ack_i  => na30_ack_o,
    ack_o  => na30_ack_i);

traffic01 : gen_traffic
  generic map(
    conf_file   => "5.txt",
    dump_file   => "5_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na01_data_o,
    data_o => na01_data_i,
    addr_o => na01_addr_i,
    type_o => na01_type_i,
    req_i  => na01_req_o,
    req_o  => na01_req_i,
    ack_i  => na01_ack_o,
    ack_o  => na01_ack_i);

traffic11 : gen_traffic
  generic map(
    conf_file   => "1a.txt",
    dump_file   => "1a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na11_data_o,
    data_o => na11_data_i,
    addr_o => na11_addr_i,
    type_o => na11_type_i,
    req_i  => na11_req_o,
    req_o  => na11_req_i,
    ack_i  => na11_ack_o,
    ack_o  => na11_ack_i);

traffic21 : gen_traffic
  generic map(
    conf_file   => "4b.txt",
    dump_file   => "4b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na21_data_o,
    data_o => na21_data_i,
    addr_o => na21_addr_i,
    type_o => na21_type_i,
```

```
    req_i  => na21_req_o,
    req_o  => na21_req_i,
    ack_i  => na21_ack_o,
    ack_o  => na21_ack_i);

traffic31 : gen_traffic
  generic map(
    conf_file   => "9.txt",
    dump_file   => "9_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na31_data_o,
    data_o => na31_data_i,
    addr_o => na31_addr_i,
    type_o => na31_type_i,
    req_i  => na31_req_o,
    req_o  => na31_req_i,
    ack_i  => na31_ack_o,
    ack_o  => na31_ack_i);

traffic02 : gen_traffic
  generic map(
    conf_file   => "6.txt",
    dump_file   => "6_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na02_data_o,
    data_o => na02_data_i,
    addr_o => na02_addr_i,
    type_o => na02_type_i,
    req_i  => na02_req_o,
    req_o  => na02_req_i,
    ack_i  => na02_ack_o,
    ack_o  => na02_ack_i);

traffic12 : gen_traffic
  generic map(
    conf_file   => "1b.txt",
    dump_file   => "1b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na12_data_o,
    data_o => na12_data_i,
    addr_o => na12_addr_i,
    type_o => na12_type_i,
    req_i  => na12_req_o,
    req_o  => na12_req_i,
```

```
    ack_i  => na12_ack_o,
    ack_o  => na12_ack_i);

traffic22 : gen_traffic
  generic map(
    conf_file   => "2b.txt",
    dump_file   => "2b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na22_data_o,
    data_o => na22_data_i,
    addr_o => na22_addr_i,
    type_o => na22_type_i,
    req_i  => na22_req_o,
    req_o  => na22_req_i,
    ack_i  => na22_ack_o,
    ack_o  => na22_ack_i);

traffic32 : gen_traffic
  generic map(
    conf_file   => "4a.txt",
    dump_file   => "4a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na32_data_o,
    data_o => na32_data_i,
    addr_o => na32_addr_i,
    type_o => na32_type_i,
    req_i  => na32_req_o,
    req_o  => na32_req_i,
    ack_i  => na32_ack_o,
    ack_o  => na32_ack_i);

traffic03 : gen_traffic
  generic map(
    conf_file   => "7.txt",
    dump_file   => "7_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na03_data_o,
    data_o => na03_data_i,
    addr_o => na03_addr_i,
    type_o => na03_type_i,
    req_i  => na03_req_o,
    req_o  => na03_req_i,
    ack_i  => na03_ack_o,
    ack_o  => na03_ack_i);
```

```
traffic13 : gen_traffic
  generic map(
    conf_file   => "2a.txt",
    dump_file   => "2a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na13_data_o,
    data_o => na13_data_i,
    addr_o => na13_addr_i,
    type_o => na13_type_i,
    req_i  => na13_req_o,
    req_o  => na13_req_i,
    ack_i  => na13_ack_o,
    ack_o  => na13_ack_i);

traffic23 : gen_traffic
  generic map(
    conf_file   => "3a.txt",
    dump_file   => "3a_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na23_data_o,
    data_o => na23_data_i,
    addr_o => na23_addr_i,
    type_o => na23_type_i,
    req_i  => na23_req_o,
    req_o  => na23_req_i,
    ack_i  => na23_ack_o,
    ack_o  => na23_ack_i);

traffic33 : gen_traffic
  generic map(
    conf_file   => "3b.txt",
    dump_file   => "3b_log.txt",
    period      => 64,
    ADDR_BITS   => 2,
    DATA_WIDTH_I => 24,
    DATA_WIDTH_O => 24)
  port map(
    clk    => clk,
    rst    => rst,
    data_i => na33_data_o,
    data_o => na33_data_i,
    addr_o => na33_addr_i,
    type_o => na33_type_i,
    req_i  => na33_req_o,
    req_o  => na33_req_i,
    ack_i  => na33_ack_o,
    ack_o  => na33_ack_i);

pgm : pgm_unit
```

```
  generic map(
    filename   => "pgm.txt")
  port map(
    noc_fw_o => link_input_0_fw,
    noc_bw_i => link_input_0_bw,
    clk      => clk,
    rst      => rst);

end clean_4x4;
```