Design and Implementation of a hot-potato Switch in a Network on Chip.

Master of Science thesis

Erland Nilsson
Supervisors: Axel Jantsch and Johnny Öberg

Master Thesis IMIT/LECS 2002-11
Laboratory of Electronics and Computer Systems,
Royal Institute of Technology (KTH),
LECS/IMIT/KTH-Electrum, Electrum 229, SE-164 40 Kista, Sweden
Email: {erlendn, axel, johnny}@ele.kth.se

17th June 2002
Abstract

As the number of transistor functions on a single chip increases, applied systems becomes more complex and the challenge of developing applications in short time is critical. The design of chips for applications used in, for example telecommunication demands high performance and reliability, while the time-to-market factor is kept to a minimum. To meet this challenge, a design methodology is required which allows reuse of previously made designs. Resources are identified from previous designs. A resource can be any kind of Intellectual Property, for example, a processor element, an I/O-element, or various memory types. Many resources can form a network mesh. The network model reflected is called Network on Chip, in short NoC. In the NoC, the computational resources are distributed all over the chip, organised in regions, connected together over a communication network. The network consists of a two-dimensional mesh of switch-resource pairs. This master thesis is an attempt of implementing the NoC switch in synthesisable VHDL.

The responsibility of a switch is to provide a resource with point-to-point or point-to-multipoint connections in a packet structure. The mesh size and the location of communicating resources clearly limits the number of packets that can possibly be traveling in the network simultaneously.

One way to implement a distributed system of resources, that oversees the communication on the network, is to design switches as cellular automata. These cellular switches would then communicate their status to their neighbouring switches. The use of stress values, which defines the switch status, can ease the decision situation for the switches and packets can be routed through the least congested path. The stress values are sent by every switch in all four directions and progresses in all directions in the manner of cellular automata. The main objective as a starting point is to keep the final switch very simple. Every little intelligence added in the switch results in a larger design, which is expensive in area, speed and not at least power.
Contents

1 Introduction ................................................. 9
  1.1 System on Chip ......................................... 9
  1.2 Network on Chip ....................................... 9
  1.3 Thesis layout ......................................... 10

2 The Network on Chip concept ............................ 11
  2.1 Basic model ........................................... 11
  2.2 Improved model ....................................... 13
  2.3 Objectives ............................................ 13

3 Architecture and layers .................................. 15
  3.1 The OSI reference model ............................... 16
  3.2 The OSI reference model adapted to the Network on Chip ... 16
  3.3 The NoC backbone standard and extended service layers ... 17
  3.4 Layer dependent clock relation ....................... 19

4 The switching policy in the NoC backbone .......... 21
  4.1 Fundamental limitations .............................. 22
  4.2 Switching decisions .................................. 22
    4.2.1 Priority ......................................... 22
    4.2.2 Routing ......................................... 23
  4.3 The buffer-less switching concept .................... 23
  4.4 Mesh boundary ........................................ 24
  4.5 Load distribution ..................................... 25
  4.6 Bandwidth reservation using virtual circuits .......... 27
  4.7 Addressing ........................................... 28
    4.7.1 Absolute addressing ............................ 28
    4.7.2 Relative addressing ............................ 28
  4.8 Error correction ..................................... 29

5 The hardware implications ............................... 31
  5.1 The switches ......................................... 31
  5.2 The network interface ................................ 31
  5.3 The resources ........................................ 32
    5.3.1 Processor elements ............................. 32
    5.3.2 Memory elements ................................ 33
    5.3.3 I/O-elements .................................... 33
    5.3.4 Statically dedicated hardware ................... 33
    5.3.5 Dynamically reconfigurable hardware .......... 34
  5.4 Regions .............................................. 34
6 Implementation
   6.1 Model presentation ........................................... 35
   6.2 Switching solution ........................................... 36
       6.2.1 With no stress value ................................. 38
       6.2.2 With stress value .................................... 39
       6.2.3 Averaged stress value ................................. 39
       6.2.4 Gate depth ............................................. 40
       6.2.5 Synthesis ............................................... 40

7 Simulation results ........................................... 43
   7.1 Maximum average distributed probability ...................... 43
       7.1.1 Theoretical values ................................... 44
   7.2 Delivery time ................................................. 46
   7.3 Hop-count study ............................................. 48
   7.4 FIFO-buffer study ........................................... 49
       7.4.1 With no stress value .................................. 50
       7.4.2 With stress value ..................................... 50
       7.4.3 Averaged stress value ................................ 50
   7.5 Stress value analysis ........................................ 52

8 Conclusion ..................................................... 53
   8.1 Future work .................................................. 54

A The final implementation ...................................... 57

B Flowchart of basic model ...................................... 59

C The control box, VHDL-code ................................... 61
List of abbreviations

BCD  Binary Coded Decimal, section 6.2
CLICHE Chip-Level Integration of Communicating Heterogeneous Elements, section 2.1
DSP  Digital Signal Processor, section 5.3.1
ECLIPSE Embedding into Chip-Level Integrated Parallel SupercomputEr, section 2.1
FIFO First In First Out, queue behaviour, section 4.3
FPGA Field Programmable Gate Array [1]
FtG  Furthest to Go, chapter 4
HC   Hop-Count, section 4.2.1
IEEE Institute of Electrical and Electronics Engineers, chapter 6
IP   Intellectual Property, section 1.2
LUT  Look Up Table, section 4.2.2
MID  Message IDentity, figure 3.3
MSN  Message Sequence Number, figure 3.3
NI   Network Interface, section 5.2
NoC  Network on Chip, section 1.2
NtG  Nearest to Go, chapter 4
OSI  Open Systems Interconnect, section 3.1
PDU  Protocol Data Unit, chapter 3
PE   Processor Element, section 5.3.1
PID  Process IDentity, chapter 3.3
SAP  Service Access Point, chapter 3
SoC  System on Chip, section 1.1
ToC  Time of Creation, chapter 4
ToD  Time of Departure, chapter 4
VHDL VHSIC Hardware Description Language [4]
VHSIC Very High Speed Integrated Circuit [4]
Chapter 1

Introduction

Today's growing need for high capacity, short development time, and low cost in electronic systems, leads to more complex systems with higher demands. To meet these requirements, it is more common to fit a whole system on one single chip. This is called System on Chip, SoC. More and more products such as mobile phones and portable computers are built on a single silicon chip. While the number of transistor functions on a single chip increases, the complexity of every system expands to a limit where development time and probability for errors becomes unacceptable. It is a great challenge to meet the requirements of low power consumption in relation to the increasing complexity together with high performance in the systems.

1.1 System on Chip

A System on Chip, SoC, can consist of for instance Processor elements, I/O-elements, and various memory types. Every element has its own specially designed interconnections, normally in direct connection to the processor core. Each unit can vary in connecting bus size, transmission speed, and transmission protocol. When talking about the architecture model, the SoC is normally considered. This may be the way the general layout of the model is made.

1.2 Network on Chip

A model of a SoC is a network of resources that eases the ability of reusing previous designs. This network model is called Network on Chip, in short NoC. Because of manufacturability and verification problems, future chips will most probably be organised as NoCs. The NoC architecture is an $M \times N$-mesh of switches with resources placed on the slots formed by the switches. Each switch is connected to one resource and four neighbouring switches. A resource can be a processor core, a memory, an FPGA [1], a custom made hardware block or any other intellectual property block, IP, which complies with the interface of the NoC.

As a result of several research groups, the same or similar architecture models have different names. For instance, Network on Chip, on-chip network [2], and micro network [3].
1.3 Thesis layout

This master thesis is concentrated on making a switch to which network resources are connected. A model of the switch should be implemented in the hardware description language VHDL. A good book where VHDL is described is [4]. During the time of the thesis work, the protocols used by the switch are in continuous development and are not fully established, this results in varying terms and protocol layout. To test the switch performance a network of switches are to be built in the test environment. Much time should be dedicated in investigating the relation between different parameters for the network. These parameters are for instance, mesh size, buffer size, and routing decisions. The switch will also have a feature that is planned to enhance the routing decision. The idea is to use cellular automata technology for switch status information distribution.

The master thesis is carried out in the following steps:

1. Study desired switch behaviour, chapter 2-5
2. Implement switch in VHDL, chapter 6
3. Build test environment, chapter 6
4. Analyse early stage switch performance, chapter 7
5. Include tool for load distribution, chapter 6
6. Analyse improved switch performance, chapter 7
Chapter 2
The Network on Chip concept

2.1 Basic model

Today, for the designer of a System on Chip short development time is one of the greatest constraints to achieve, to decrease the time-to-market factor. Creating building blocks of modules by dividing larger elements in a design into smaller units with clear and separate tasks, the design gets a clear overview. Each unit is a resource. The resource can be any kind of Intellectual Property, IP, element in a SoC, among others processor cores, I/O-elements, and memories. Each resource is specially designed for its particular purpose. When designing a SoC, instead of starting from scratch every time, the resources can be reused from previous designs with no or few changes. For example, if a design needs an I/O-unit, the designer can pick a suitable and reliable element from a resource library or database and put into the new design. Normally, each element has its own interconnection. Since the intention is to use the elements as building blocks in a larger structure, the interconnections have to remain the same independent of its particular purpose.

The resources are connected in a network. Any kind of information is sent to the destination over the network in packets. Every packet has the same predefined structure and can be routed any way from source to destination. This network structure is called Network on Chip, NoC. A NoC is normally a mesh of switches. It can be found natural that a symmetric mesh must be the most efficient topology in two dimensions, for further motivation see section 7.2.

The network contains of resources and switches where each resource is connected to one switch. On the other hand, depending on the architecture model chosen, each switch is not necessarily connected to a resource. In the model used in this master thesis, the number of switches and the number of resources are the same, they build switch-resource pairs. This model is called CLICHE which is short for Chip-Level Integration of Communicating Heterogeneous Elements, see figure 2.1. The second model is called ECLIPSE which is an abbreviation for Embedding into Chip-Level Integrated Parallel SupercomputEr [5], figure 2.2. In the ECLIPSE model, the number of switches is greater than the number of Resources. This model increases the number of possible paths between two resources. However, this structure increases the distance between two resources, the number of gates in the design, and the physical size of the final layout in relation to the number of resources.

In the topology suggested each switch has a resource connected to it. The resource transmits and receives data packets to and from the network through
the switches. On the output from the resource, a FIFO buffer is applied, the size of this buffer depends on for instance the network load, transmission probability or packet insertion rate, and the physical location in the network. Even if the average packet probability is low over a long time it can still fill the FIFO buffer if many packets are sent in bursts. This is the reason for a limited insertion rate, which results in a restriction for the resource so that it is not allowed to transmit more often than for example every second possible transmit period. The aim of the network is that every packet in the network should be immediately forwarded to another switch every clock cycle and never buffered; this switching policy is called hot-potato routing. An advantage of using hot-potato policy is that there are no output queues in the switches on the regular switch interconnections, which keeps the size of the switch to a minimum. Instead, a packet may be routed in a loop to surrounding switches when waiting for its turn to reach its destination [6].

To increase the locality in a NoC, a number of resources can form an operating block. Most of the data traffic created here are assumed to be inside this block of resources. This block is called region. A region may act as a large resource which occupies an area of more than the size of one standard switch-resource pair. To guarantee a certain bandwidth for the resources-to-resource communication within a region, packets with a destination outside the region may not be allowed to enter the region. Using regions hereby restricts the generic layout model. For further explanations in section 5.4.
2.2 Improved model

Consider a case in a symmetric network, $N \times N$-mesh, where the source and destination of a packet is stochastic. The probability for transmission is equal everywhere in the mesh. It is natural to expect that there will be higher traffic of messages passing through the middle section of the mesh. Thus the centre will become a hot-spot. It is shown by simulation that the centre switches undergoes much higher load compared to one on the border. The maximum network throughput is limited by the load distribution in the switches. A bottleneck behaviour is being introduced. This is however not likely since the network design should have an inherit locality of resources. A memory often used by the processor element is expected to reside in its immediate neighbourhood. By spreading the load in the network by routing packets around the worst congested area, more uniform load distribution can be achieved, and the network throughput increased.

2.3 Objectives

In order to measure the performance of the NoC, the information of interest are among other the following:

- Network delay
- Average waiting time
- Number of packets in FIFO
- Network throughput

This can be achieved by studying the average number of packets in the queues, the number of hops until the destination is reached in relation to the optimal distance, investigating the maximum throughput while remaining stability, reliability, and robustness. The performance is always a trade-off from the switch speed and physical area used by the switch. The aim is to find an acceptable switch size counted in number of NAND-equivalents and a packet switch frequency high enough to meet todays demands. The optimal switch frequency is based on the ability of switching packets in all four directions and to and from the resource in one clock period only. With the technology that is intended to be used, this results in a factor of the time it takes for a gate to switch output depending on input. If the constraints of making a switch with gate depth small enough to meet this, pipelining will have to be used to keep the clock frequency at the same rate.

The time it takes for the signals to progress over the interconnections is also an issue that limits the maximum clock frequency possible.
Chapter 3

Architecture and layers

The format of communication needs to be strictly defined, in order to make data transfer between different processes. In order to ensure that various processes understand each other, definitions of protocols are set that explain how the communication should be carried out. To prevent misunderstanding, a few terms must be explained.

The information exchanged by processes are called messages. A message can be any kind of information, for the network itself, the data in the messages are of no importance. Instead, the characteristics of the data flow can be of greater interest, for instance if the exchanged messages are sent in bursts or in a steady stream, data transfer rate, and acceptable delays.

A whole communication system easily becomes very complex when a number of protocols are used. To simplify this, an abstract model of a vertical set of layers are introduced, where each layer provides layers above with services. A service is an agreed functionality a particular layer has to provide. Data exchange is only made between two layers of the same level, they speak by definition the same language.

Figure 3.1: The layered approach using service access points and entities.

In the layered structure approach, each layer includes two fields of information, header and payload. What is a whole string of information in one layer is only regarded as anonymous payload in a lower layer. Advantages of using the layered model is the possibility to change the implementation of one layer without affecting the others. To be able to provide the service promised, a number of functions in each layer are grouped as entities. Entities within the same layer communicates together using agreed protocols. Every layer groups the information sent from an upper layer with a header created by the layer itself, see figure 3.1. This header includes essential information needed in that particular layer to be able to fulfill its services. The header together with the anonymous payload in a layer is called Protocol Data Unit, PDU. Examples of well known
PDUs are packet, frame, and datagram [7]. The interface between the upper and lower layer is referred to as Service Access Point, SAP. The basis for the layered structure used in the NoC is the well known OSI reference model [8].

3.1 The OSI reference model

The original and well accepted Open Systems Interconnect, OSI, model is used as a basis for many communication systems [7]. The functions are partitioned into a vertical set of layers, where each layer has one or more responsibilities for the layers above. The layers in the seven layer model are described briefly below. The seven layers are shown in figure 3.2 and are together called the protocol stack.

**Physical layer:** Concerned with transmission of unstructured bit stream over physical medium; deals with the mechanical, electrical, functional, and procedural characteristics to access the physical medium.

**Data link layer:** Provides for the reliable transfer of information across the physical link; sends blocks of data (frames) with the necessary synchronisation, error control, and flow control.

**Network layer:** Provides upper layers with independence from the data transmission and switching technologies used to connect systems; responsible for establishing, maintaining, and terminating connections.

**Transport layer:** Provides reliable, transparent transfer of data between end points; provides end-to-end error recovery and flow control.

**Session layer:** Provides the control structure for communication between applications; establishes, manages, and terminates connections (sessions) between cooperating applications;

**Presentation layer:** Provides independence to the application processes from differences in data representation (syntax).

**Application layer:** Provides access to the OSI environment for users and also provides distributed information services.

![Diagram of the OSI reference model](image)

*Figure 3.2: A layered model of the network structure.*

3.2 The OSI reference model adapted to the Network on Chip

Several advantages comes along when using a protocol stack.
- Services are defined, each layer has well defined responsibilities, easy to identify operation of every layer

- Easy to perform simulations and implementations of the different layers independent from others

- Possible to change implementation of one layer without affecting others

Compared to a standardised protocol stack, such as TCP/IP, this network will in general only communicate with known resources. The design of one whole network is made simultaneously. This leaves room for changes if the designer may want to, as a result of the nature of the current design requirements. The exact implementation of services can be changed from design to design depending on the specification.

The physical layer is responsible for signaling voltages, timing issues, and pulse shapes. The physical layer identifies each signal as a one or a nought and is presented as either or for the data link layer. The data link layer works with frames that is sent in a peer to peer way between two switches, possible error control is made at bit level, such as parity check. For more sophisticated services such as better error control, an extended set service layers can be used. The standard service layers are the basic layers that are essential for the network’s functionality. To enhance the reliability and also include more services, an extra set of layers that are working in parallel to the standard layers is added. For instance, error control in different layers, packet loss discovery, load distribution, broad- and multi-casting. The only layers needed to route packets through the network are the three lowest layers, up and to the network layer [9].

3.3 The NoC backbone standard and extended service layers

This thesis is concentrated mainly on the behaviour of the switch and the network in general. The network layer and below are the levels of operation for the switch, the upper layers are only briefly described and suggested options of a NoC. After all, the final NoC is in the end determined by the designer.

The standard service definition at the data link layer provides peer to peer communication between two switch pairs, the layer is responsible only for moving one frame from the output on one switch to the input on the next switch with the synchronisation necessary. In a perfect world or in a simulation environment, there is normally no need for error correction, this might however be needed in an implementation of a real application. The NoC backbone is fully usable without the extended services, but the extended service at the same layer are to provide reliable transfer of information across the physical link using error and flow control. The error control can be in three modes.

- No detection

- Detection of errors in the header

- Detection of errors in a full frame

An error in the header is more dangerous to the network, miss-routed packet increases the load and may cause damage if interfering with critical applications. For example, take collision in a virtual circuit. If the packet is of the type where packet loss is not crucial for the application, the packet with a detected error can simply just be dropped without further notice. A service for a certain layer can automatically request a new packet, or simply just notify upper layers that
a packet loss has occurred. Otherwise, upper layers may expect a packet, the receiving node will request for packets that has not arrived after a fixed time. The sending node could also expect acknowledges for the packets in a sliding window manner [7]. Sending continuous acknowledgements will increase the number of packets that are being transferred in the network which results in increased network load.

At the network layer, the service is to provide error free transport of payload from the transmitting resource through the network to the resource of the intended destination.

The network layer is the highest layer of the switch operation. Services of the network layer defines how a packet should be routed. This is made by local decisions dynamically in the switches. The decision is based on the hop-counter and destination address, there are also options to extend this with priority and pre-defined routing. Packet priority is used in order to make good routing decision in case of a challenge of the same direction. In the end, in case of a tie, one input will finally always be preferred.

![Diagram of PDUs for various layers](image)

**Figure 3.3: Overview of PDUs for various layers.**

The transport layer serves for splitting and assembling messages into packets to and from the resource and adding headers needed to accomplish the services of that particular layer. The standard service divides large messages from the network layer into packets and tag each packet with a packet sequence number. To identify the last packet of a disassembled message, the last packet will also carry an end of message tag. From the network layer, the destination information is in the form of a process identifier, PID. It is then up to the transport layer to transpose the PID to a physical destination address and include it in the packet header. This can be made in the form of a database resource with a fixed address which contains information about every process running, the physical address and its process ID. Information about PIDs can be broadcasted or whenever a new destination address of a PID is to be found, an address resolve request are sent to the database. The reason for working with PIDs and databases is to create a dynamic system which can cope with changing network structure during runtime. The PID may be fixed at the design time but the system implementation may be of the more dynamic type. A resource may
also have more than one service and therefore more PIDs pointing to the same physical address. Incoming packets needs to be buffered, sorted into order and assembled together back to complete messages. The extended version also ensures that the current PDU is delivered without errors, or at least that possible errors are detected and with no packet loss or duplication [10].

3.4 Layer dependent clock relation

Data link layer clock decides the speed at which frames are sent out. A packet can be of a larger size then the size of one frame. For example, a packet that is twice the size has to be sent in two frames in a pair. This means that the quota between the packet and frame is two, although the header is in the beginning of the packet and therefore only in the first frame, see figure 3.4. The utilisation of a frame regarding payload is higher, the payload to header ratio increases. Hence, the network layer clock or the switching clock is, according to the example, half the speed of the data link layer clock. The packet to frame ratio needs to be constant for every switch in the network and is easiest defined in the design stage of the NoC. A message from a resource may not fit into one packet, therefore, methods of splitting and reassembling packets into full messages must be investigated. This is however not of the same fixed relation at the design stage since every packet can be treated independently. On the other hand, the number of packets in relation to the number of messages are always equal or greater than one and can be fairly constant in one communication channel.

![Diagram of packet to frame relation]

Figure 3.4: Packet to frame relation.
Chapter 4

The switching policy in the NoC backbone

The NoC is arranged as a mesh of switches. Each switch has five inputs and five outputs. One input/output pair is for communication with the resource. The remaining four inputs and outputs are connected to the surrounding switches, the connections are named after their logical direction, North, East, South, and West. Incoming packets to a switch are put in input buffers, which can hold one packet. Each packet are first treated separately and their destination direction is investigated. The destination direction is simply the direction to the destination switch of which the packets aims. Take for instance a packet from the lower left corner of the network, south west, and its destination in the upper right corner, north east. As long as the packet is not on the same row or column as the destination switch, it will aim north east.

When a challenge of direction occurs, the simplest method is to let one input choose first and then consequently let the remaining packets in some predefined order choose. This is however not a good idea, some packets may be deflected so many times that the delivery time reaches an unacceptable level. Another method, which still is a fairly simple procedure, is to use some sort of priority to determine which packet that may make the first output choice. There are several ways of rating the priority.

Examples of ratings are:

- FtG (Furthest to Go)
- NtG (Nearest to Go)
- ToC (Time of Creation)
- ToD (Time of Departure)
- User defined priority

Of course, combinations of the above with different scaling are possible.

One field in the packet structure is the hop-counter, HC, which is incremented for each hop between two switches. The hop-counter determines the Time of Departure of each packet. Using the hot-potato approach, where every packet is forwarded to the neighbouring switch or resource every network clock cycle, the hop-counter reveals when the packet was created.
4.1 Fundamental limitations

Many factors affect the limitations on the network, these can be mesh size, packet probability, locality of resources communicating with each others, mean distance, and data traffic behaviour. The relation between mesh size and the packet probability is discussed further in section 7.1.1.

4.2 Switching decisions

To be able to route each packet through its most efficient path, a set of decisions have to be made by the switch. The switch follows policy rules set by the designer. The rules are a result of the desired behaviour of the network. Perhaps it is acceptable that for example 80% of the packets are delivered within a certain time while the remaining packets are received after considerable delay. In this case, the individual packets can be of interest.

Another case could be when a message larger than the payload in a packet has to be divided into several packets. Now, the total delivery time is evident because every single packet has to be received until the whole message can be reassembled. A switch policy that can guarantee that the packet will be delivered within a time given by the constraints should be used. This could result in that every packet takes a longer time to progress through the network mesh but in a fairly straight order with an almost constant delay.

4.2.1 Priority

Every packet that is being received by a switch is going through a number of investigations. This is mainly to extract the following information:

1. The destination address
2. Information used for priority calculation

When a direction challenge occurs between two packets or more, the packet with the highest priority wins. The packet structure is depending on what the priority should be based on. The structure is decided once for all during system layout. Why the packets structure is changing from time to time is because the packet concept increases the information overhead in terms of header information. If the overhead information can be kept to a minimum, the header to message ratio can be optimised. In other words, information not necessary for the switch should not be included in the layer on which the switch is working in, the network layer.

If the priority of two packets are the same, one of the incoming packets must be favoured. To remain a balance in the network, a solution would be to make the choice randomly. This is on the other hand at the cost of a small increase in physical area and gate depth. The simple solution is to fix one incoming direction that always has this built-in advance. The remaining inputs are ordered in the same way. This means that the selection between the incoming packets are not truly random.

In switch implementation, the priority has been solely computed by the hop-counter, HC. The hop-counter is incremented for every network clock cycle which is the same as the number of passed switches since the packet were launched on the network. The hop-counter is the difference between the Present time and Time of Departure, ToD, counted in network clock cycles.
4.2.2 Routing

Assume a case when the resource is neglected, i.e. every packet is only passing through the switch, there are a number of permutations of cross-linking the packets. Each output has its own multiplexer which decides which input that is coming through. Since one input can only be connected to one output, the number of possible permutations are the factorial of the number of inputs, that is $4! = 24$. In table 4.1, every possible permutation of the four multiplexers is stated. The first letter in each column denotes the chosen input for the switch with the output to the north, the second letter corresponds to the east output, the next two means the setting for switch with the output to the south and west. If the routing is restricted with "no bouncing" so that packets are not allowed to return to the direction of its origin only nine possible permutations are left. This is shown in table 4.2

<table>
<thead>
<tr>
<th>Possible permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE SW</td>
</tr>
<tr>
<td>NW SE</td>
</tr>
<tr>
<td>NW ES</td>
</tr>
</tbody>
</table>

Table 4.1: Possible permutations with bouncing allowed.

<table>
<thead>
<tr>
<th>Possible permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN WS</td>
</tr>
<tr>
<td>ES WN</td>
</tr>
<tr>
<td>EW NS</td>
</tr>
</tbody>
</table>

Table 4.2: Possible permutations with bouncing not allowed.

The fixed combinations of permutations opens the possibility to make the switch simpler in logic since the decisions can be a result of a lookup table, LUT. On the other hand, this gives us no ability to reconfigure the switch in a simple manner.

4.3 The buffer-less switching concept

In a conventional network, every network interface has input and output queues. This increases the network utilisation, since packets are sent directly when there are free space and packets are waiting in the output buffer. The communication between the network interface and the resource is more sophisticated since the resource is expected to have a lot more computing power and spare resources such as memory. In the Network on Chip concept, we have four peer-to-peer connections to other switches for each switch. The maximum number of packets that can possibly arrive to a switch during one clock period is four, if the resource connection is neglected. The number of outputs are also four, this means that there will always be at least one output available for each input.

Instead of using queues and let packets wait for a few clock cycles until it can be passed on to the next switch in the desired direction, the buffer-less switching concept and the hot-potato algorithm is used. Every packet coming
in to a switch is passed on to another switch directly the following clock cycle, like a hot-potato. This does not include the effect of the input and output to the resources, nor the boundary effects when a switch is on the edge.

Resources may launch a packet to the network when there is a free output; this occurs when the number of incoming packets are less than four but also when a packet has reached its final destination and sent to the resource. In both cases, there will be at least one output free, which can be used for new packets originating from the resource.

If there are packets on all four inputs and no packet is to be forwarded to the resource, a packet from the resource is unable to be released on the network. In this case, a signal from the switch to the resource must be sent to notify whether the waiting packet was taken or not. This leaves the switch with one buffer space since the updating of the resource will not be made until the next clock cycle, which is when the previous packet are to be dropped by the resource. The number of buffer spaces in the switch is simply an agreement between the switch and the resource, it is either in the switch or in the resource. The smallest number will still be one. The buffer is a FIFO-buffer, First In First Out. By counting the number of packets waiting to be launched gives us a good estimation of the network load.

### 4.4 Mesh boundary

There are various ways of treating the connections on the boundary switches. The outputs going to the edge on the boundary switches does not have to be connected but to make the switch as general as possible, the switch does not know of its physical position in the mesh. Therefore, it is better to just return the outgoing connection to the input on the same switch so that the packet sent in that direction returns to the very same switch. This can be used as a one packet buffer, see figure 4.1. The edge connection can also be connected to for example a switch in the clockwise direction and let two connections go from one switch to the adjacent switch in that direction. This results in some kind of ring road in one direction. It seems that this will enhance the network performance but it is shown that for the previous switching policy the most congested area is always in the centre of the network. This will also increase the complexity of the switch if the direction are to be computed only for special switches.

![Figure 4.1: Example of connection in edges.](image)

If the edge outputs are connected to the inputs on the opposite side, i.e. the western outputs on the most west switches are connected to the eastern input on the most east switches and vice versa, this forms a torus which in the addressing point of view is pleasant, but the physical realisation will in the end produce long connection wires which is not eligible.
4.5 Load distribution

In previous sections, the congestion in the network centre is discussed briefly. It is shown by simulation in chapter 7 that the load in the centre of a fairly large network, larger than $5 \times 5$, is much higher compared to the total average. The maximum packet probability for every resource is limited by the centre load. If the load can be spread over a larger number of switches, the network maximum average packet insertion rate can be significantly increased, hence the network throughput is increased.

Examples of routing rules can be:

- None
- Try to avoid congestion
- Force avoidance

The first rule, none, can be useful when there is some built-in locality of the resource, i.e. the distance between the communicating resources is very short. However, in a situation where the source and destination of every packet is completely random, there will be congestions that limit the network performance.

By trying to avoid congested areas, the network throughput can be improved. For larger networks, there are almost always two directions which is decent. The cost of taking the other two directions is two clock cycles, one when going in the wrong direction and another one to return back on track again. Depending on the load in the two directions to which a packet are heading, the direction where there are least congestion is the better one and preferred.

To make the load distribution more uniform, information to help the switches in their routing decision is sent between the switches. The information is sent from one switch to its neighbours in all directions. That is the result of a calculation that relates to the load level in that switch. The surrounding switches get information from the switches in the four directions; this helps the switch to get a picture of the surroundings, see figure 4.2. This informative value is called stress value, Proximity Congestion Awareness [9], mood, or simply load status.

![Stress value distribution](image)

Figure 4.2: Stress value distribution.

The load information from each switch progress over the network in the manner of a cellular automata. Cellular automata is basically a concept for making status information about one cell progress to its surroundings. One cell in a mesh can tell its most adjacent neighbours about its status, such as its load. Other cells will not know about one specific cell until after a few clock cycles. Depending on the network size, the current status will have progressed
over the network like rings on water. This helps each switch to decide in what direction each packet should be sent to limit the hot-spots, where the load is high for a longer time. The status information will not be about one specific cell, only what the load looks like in that particular direction.

Take for instance a cell in the center wants to transmit a packet to east but the status from the east cell tells about a congestion in that direction. It can then be a smarter choice to transmit the packet north, south or even west to let the packet find its way through without increasing the congestion too much. What may become a problem is that the network status will change rapidly when the network locally adjusts to the current load, which in turn may result in load oscillations over the network mesh.

The calculation of the load can be made in many different ways, for example:

- Number of packets switched
- Number of packets switched, averaged over a few clock cycles
- Any of the above and scaled with incoming information from adjacent switches

The complexity of the switch is increased in the order in which the methods are presented. As before, the least cost implementation in terms of physical size and execution time is in descending order from the top.

The simplest implementation is to count the number of packets switched and transmit the result to all the neighbouring switches. This is in chapter 7 shown that the increase in performance is noticeable. Assume that one switch is heavy loaded at a given time, the switch will send this stress value to the adjacent switches. The surrounding switches will then hesitate to transmit to the first switch during the next clock cycle. It is not impossible that the first switch in the next clock cycle does not get any packet to switch, the stress value will then decrease to zero. This can proceed like this over a long period of time. The result is oscillations in the load over the network. To achieve the highest possible network throughput, the number of packets switched in each switch should remain constant over time and equal over the network.

The goal is to achieve a stable and equally distributed load over the whole network. The utilisation of the network capacity is rapidly increased if there are few switches running empty. To overcome the problem with oscillations in the network as a result of the reaction in the switches because of the stress value, the number of packets in a switch should be counted and averaged over a few clock cycles.

To improve the stress value calculation even more, the stress values coming from the surrounding switches can be accounted for in the new stress value together with the number of outputs used over a few clock cycles. These should be weighted and added together. For example, stress values coming from the surrounding switches are multiplied by a constant and added together with the number of busy outputs in the switch itself. In the end, the final value will have to be scaled so that the maximum value is not exceeding the maximum stress value available.

The computing logic for doing this and the time it takes for the operation has to be put in relation to the gain of performance. A congested area is normally spread over a few switches, for instance traffic is going from one side to another in parallel switches, several columns or rows of switches. This means that the stress value calculation with the stress value of the adjacent switches might be a high level of ambition since the decision is only made between four outputs. The small change in stress value this could result in, may not defend the cost it takes.
4.6 Bandwidth reservation using virtual circuits

If many packets are expected to be transmitted from one resource to another, a virtual circuit can be set up. Bandwidth through a fixed path can be reserved. If there for example is a stream of data being sent between two resources, a virtual circuit could be requested and every packet routed through the same path, preferably the shortest possible. Every packet will arrive in order and with the same delay between each packet. The virtual circuit can be a dedicated time slot of for example one of four slots. In this time slot, the switch always route packets from the defined input to the output. Control information to request and cancel the virtual circuit is handled by the requesting resource. The circuit must be a two way communication circuit, full duplex, since the end switches must guarantee that a packet can be sent from the resource. This means that incoming packets must be one less than the number of outputs or one of the incoming packets must be passed on to the resource. For natural reasons, to make sure there is always one packet going to the resource, the virtual circuit must be defined in two directions. The protocol of how to allocate bandwidth is not yet fixed so the request procedure is just an example.

![Bandwidth reservation with virtual circuit](image)

Figure 4.3: Bandwidth reservation with virtual circuit, shortest path or least congested path.

When requesting a virtual circuit, the requesting resource sends out control packets to each switch in a predefined route, starting with the nearest. Each switch in the path replies to the issuing resource whether the request for that certain time slot is accepted.

Assume that two pairs of resources allocates bandwidth where parts of the path overlap. It is obvious that if they transmit so that the packets are being received in a shared switch in the same time slot, there will be a collision. A solution can be to only allow orthogonal crossing of two virtual circuit paths, under the assumption that the path is straight forward in the current switch. Another method to work around the problem is to put the allocation in another time slot. In the end, depending on load and routes, there will be a limited number of possible virtual circuits in the network simultaneously.

27
4.7 Addressing

Two methods of addressing each switch and its adherent resource are by using absolute or relative addressing. There are different aspects of why one is preferred before the other. Each of these two approaches affects the switch size and time of relaying packets.

4.7.1 Absolute addressing

The term absolute means that each switch has been assigned a fixed address. The address is the numbered physical and logical position in the two dimensional mesh. The address consists of two fields, row number and column number. When using absolute addressing, each switch needs to be aware of its logical position, its own address. Each incoming packet needs to have its destination address extracted and compared to the switch address. The comparing logic has to dispatch three signals for row comparison and another three for column comparison. In the row case, the following output signals are of interest:

- Own row address: ownRow
- Packet destination row address: destRow

1. destRow > ownRow
2. destRow = ownRow
3. destRow < ownRow

The column address is of course going through the same inquiry. It can either be done with the same logic but at another time which can detain the routing decision, or the logic can be duplicated, as for all four inputs plus the resource. The information given by the comparing logic can then be used to automatically set a vector of directions without further preparation.

<table>
<thead>
<tr>
<th>absolute addressing</th>
<th>relative addressing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>-2,-1</td>
</tr>
<tr>
<td>1,2</td>
<td>-2,0</td>
</tr>
<tr>
<td>1,3</td>
<td>-2,1</td>
</tr>
<tr>
<td>2,1</td>
<td>-1,-1</td>
</tr>
<tr>
<td>2,2</td>
<td>-1,0</td>
</tr>
<tr>
<td>2,3</td>
<td>-1,1</td>
</tr>
<tr>
<td>3,1</td>
<td>0,-1</td>
</tr>
<tr>
<td>3,2</td>
<td>0,0</td>
</tr>
<tr>
<td>3,3</td>
<td>0,1</td>
</tr>
</tbody>
</table>

Figure 4.4: Absolute addressing compared to relative addressing in the view of the lower centre node.

4.7.2 Relative addressing

By using relative addressing the switch does not need to know its own location in the network. Every switch considers itself to be at row and column position zero, the numbering is therefore positive and negative depending of the relative position of another switch. With the assumption that its own address is zero in both rows and columns the comparison needed is the same as for absolute addressing. There is of course some logic needed to find the vector of directions mentioned previously, although it is simple. To simplify the discussion, absolute addressing is using the comparing logic in relative addressing plus the operation of subtracting the switch’s own address from the destination address. The comparing operation will finally be the same for both methods of addressing:
1. destRow > 0
2. destRow = 0
3. destRow < 0

Designing the switch using relative addressing allows the switch to be very simple regarding direction inquiries in terms of switch size and speed. On the other hand, the destination address must be updated every time the packet is passed on to another switch, something that is not needed for absolute addressing. This is an incrementer on the north and east outputs and a decremer on the south and west outputs. The ambition is to make a general switch that is unaware of its logical position and has the exact same layout throughout the network. The normal update of the address must be specially treated in edge switches or switches with outputs not following the conventional mesh structure. Switches positioned on the borders, that are sending the packets in the edge direction, are not supposed to increment or decrement the destination address. These packets return to the last switch and the destination address should remain unchanged. This under the premise that edge output connections are connected back to the input of the same switch.

To be able to address any switch in an \( M \times N \)-mesh using absolute addressing, the number of possible addresses should be at least \( M \times N \). For example, in a 4\( \times \) 4-mesh, the address space should be four rows and four columns. However, when using relative addressing, it must be possible to address every switch no matter where in the mesh area the origin is. To be able to send from for example a switch in the most southern row, it must be possible to address three rows north to reach every row. Also, if a switch in the most northern row is to be able to transmit to every row south of the switch, it must be possible to address three rows south. In our example, using absolute addressing does only need two bits for row addressing and two more bits for column addressing. Using relative addressing, three bits in the row and three bits in the column address must be used. To conclude, using relative addressing extends the destination field in the packet header with two bits to be able to address as many switches as absolute addressing.

4.8 Error correction

The original layered model of the Network on Chip protocol does not include error correction. Each layer can have a service working in parallel to the base service, which can include error correction. It is not included in the standard service since the NoC can be run without it. It is rather an extended service which can be included depending on the demands the system to be designed has.
Chapter 5

The hardware implications

5.1 The switches

On a NoC, the main area should be occupied by the resources. Interconnections and switches are expected to occupy less than 10% of the whole area. The switch may hereby not include as much intelligence as desired to achieve efficient network utilisation. The switch needs to stay simple to keep a small area but also to have a short gate depth to be able to make fast switch decisions.

To be able to run the switch and the corresponding network at a desired clock frequency, the gate depth must be less than a number at a rough estimate, between ten and twenty. If this goal can not be met, pipelining may have to be used. Introducing pipelining in the design results in switching time of more than one clock cycle. The packets are still coming in to the switch at the same interval but are progressed through the switch during a few clock cycles. Each packet needs a buffer for each clock cycle if it is being held inside the switch. A simple calculation example gives us the number of gates per clock period of buffering.

Assume that we are using a bus width of 128 bits, in a packet is every bit sent in parallel. It takes six gates to buffer one bit [11]. Four packets has to be buffered each clock period, since there are four incoming packets. The number of gates used is then 4 × 6 × 128 = 3072 gates.

To shorten the total gate depth, more calculations will have to be executed in parallel, at the cost of an increased number of gates used and a slower design.

5.2 The network interface

The Network Interface, NI, is located on the threshold between the switch and the resource. It serves the switch with packets that are to be transmitted on the network and is also responsible to assemble messages that are sent in multiple packets. The Network Interface works in the transport layer according to the layered structure in chapter 3. The Network Interface must also be able to buffer packets that are being sent from the resource, while waiting for an empty output buffer in the switch. The buffer space needed is an agreement between the Network Interface and its corresponding resource. How fast a resource can stop emitting packets to the Network Interface when the internal buffer are about to become full is a factor of the buffer size. In addition to that, the probability of a congestion on the network at the position of the switch determines the appropriate buffer size, see section 2.1.
Figure 5.1: The Network Interface on the threshold between the switch and its resource.

5.3 The resources

A resource is the essential object that can be compared to a certain function in a System on Chip. Hence, the NoC is simply the structure of making the interconnections between various resources, which in a SoC is made by explicit connections for each resource. The resources can be of any kind, the only part that needs to share the same functionality within the resource is the connection to the switch, the Network Interface. It can be discussed whether the Network Interface should be considered to reside in the resource or in the switch, but the fact of the matter is that if the switch alone is only to interface to the network layer and down, the network interface is an overview of the transport protocol in the resource or on the threshold.

Examples of various resources:

- Processor elements
- Memory elements
- Input/Output-elements
- Statically dedicated hardware
- Dynamically reconfigurable hardware

What should be regarded as a resource is simply a question of how much that can be fitted into one single resource. It is a question about possible size constraints. For example, a processor element may have special functions that are put in separate resources. These can also in the same time be shared by other resources without interfering with the original processor element, if desired.

5.3.1 Processor elements

A spread purpose system which can be assigned to perform several tasks may be built around one or more processor cores. The processor core can hand out tasks to other resources. For example, specified calculations or handling DSP-characterised information can be distributed to dedicated hardware. Communication between these resources are to be made through the NoC. For some special purposes there can be a direct connection to the most adjacent resource without going through the network. For instance, this can be allowed for cache memory that can not be fitted in the same resource as the processor element. Here, the basic model of the Network on Chip is being put aside to benefit the
direct connection to the dedicated memory in a nearby resource. A more complex model of a resource containing a processor core with perhaps local memory on-board are referred to as a processor element.

5.3.2 Memory elements

In almost every system, there is need for memory of various kinds. For example, memory that complies with short access time or mass storage. Memory in a NoC can be divided into two types, statically allocated memory and shared memory. The statically allocated memory is specially designed to suit a certain instance and are only to communicate with one pre-defined resource. No other resources may use this memory resource. In this case, no memory protecting intelligence is needed, the only supervision needed is the normal memory references in the communicating resource.

The second type earlier mentioned is the shared memory. This is a memory that can be used by any other resource. This type highly motivates the use of network interconnection structure since it can easily be reached from everywhere. Simply put the address to the shared memory as the destination address to request for or write information. To prevent unauthorised use of the same memory space by concurrent resources, memory protection must be used. This protection logic can be a part of the memory resource, but there are always a possibility of dividing the control logic and pure memory block into separate resources. In the second case, every memory access must be made through the controlling resource. The control logic can in its turn request memory access by overruled wiring between the memory protecting resource and the resource of the memory itself.

5.3.3 I/O-elements

Without any external interface, the system is useless. What is the reason of designing and running a system which can not present any output? Every system needs to have some kind of communication to the outside world. It can either be communication to other systems or communication to or from an expected user or observer. An input/output interface, I/O-interface, are to be designed as a resource with its corresponding Network Interface. For example, an I/O-element can be used to connect several NoCs together, interface with external memory or implement any kind of outside connection protocol, for instance TCP/IP interface.

5.3.4 Statically dedicated hardware

When discussing the processor elements in section 5.3.1, it is mentioned that some operations can be handed out by the processor core to decrease the possibility of bottlenecks. Statically dedicated hardware for decreasing the load on a processor element or some pre-defined algorithm that are to be computed can be the issue for a certain resource. For instance, an off-the-shelf Digital Signal Processor, DSP, can be used as a statically dedicated resource if a network interface is added to communicate with the rest of the network resources. On the other hand, when using pre-fabricated units it is a question about communicating with external resources, I/O-elements for interfacing with the external module should be used, see section 5.3.3.
5.3.5 Dynamically reconfigurable hardware

Dynamically reconfigurable and statically dedicated hardware in the previous section is similar. The dynamically reconfigurable hardware can be a programmable device, FPGA, that can be reconfigured during run time. The use of this hardware is issued by other resources. The issuing resources can launch a new task on the programmable device that suits the current application. The hardware can be used to solve a current problem or temporarily relieve the pressure on another resource by sharing the work between the resources. Of course, the whole programmed hardware does not necessarily need to be reconfigured, it can also mean that a few parameters are changed and is therefore ready for new assignments depending on the current configuration.

5.4 Regions

In the introduction to the Network on Chip, section 2.1, regions are introduced. The purpose of using regions is to create some locality between the resources of the NoC. This is to keep the distance between communicating resources short and to prevent packets of traveling irrational routes. When introducing regions, a number of resource switch pairs are combined together as large blocks. Restrictions that restrain packets with destinations outside the region to enter the region can be assigned. This results in a region that does not allow thorough-faring packets. If the regions are unlucky planned or are placed beside other regions that results in blocks of resources that are concave. Concave area means that a number of resources are enclosed by regions, and therefore produces dead ends, see lower right region in figure 5.2. Keeping the switch size to a minimum, fairly unintelligent, and in the same time constructing a complex network mesh structure is to beg for problems. The simple solution is to have some space between each region and make every region convex which results in no dead ends, compare to the upper left region in figure 5.2. Another method that in the same time also generates a switch of more complex nature is to introduce a passport control. On the border of a concave area, each packet needs to be examined and rejected if the target destination is within the enclosed area or in the region behind.

Figure 5.2: Convex and concave regions.
Chapter 6

Implementation

The implementation of the switch has been performed using VHDL [4]. VHDL has become the industry standard language for describing digital circuits, mainly because it is an IEEE standard. The original standard was adopted in 1987 and was given the name IEEE 1076, but is usually referred to as VHDL 87. The revised version came in 1993 and is called IEEE 1164. VHDL was used as a documentation language for describing the structure of complex digital circuits. The use of a hardware description language provides features for modeling the behaviour of digital circuits [1].

The final VHDL code written to implement the switch is written in VHDL 87 for compatibility reasons.

To create a reliable environment, not only the switch but also its implementation as a whole network has been programmed, as well as resources for providing and saving simulation data.

Matlab was used for generating simulation data for various network sizes and analysing simulation results, mainly because of the feasible possibilities of changing parameters for the characteristics of the input data. But also the simplicity of creating suitable visualisation for representing the analysis.

6.1 Model presentation

The main objective as a starting point, is to keep the final switch very simple. Every little intelligence added in the switch results in a larger design, which is expensive in area, speed and not at least power. In later stages, improvements may be added if the benefits exceeds the cost. The longer distance between the input and the output counted in number of gates passed, i.e. the gate depth, the lower packet frequency is the switch possible to deal with. If the control logic needs a certain time to make the appropriate decisions and set the outputs, the period of the packet frequency must not be higher than the corresponding time period. To achieve as short set up times as possible, many tasks must be executed in parallel. In the situation of the switch, added intelligence ought to be added in parallel to the existing logic to maintain the highest possible speed. This under the assumption that the most basic switch is the fastest, which can be found to be quite natural. On the other hand, it may still be to prefer that added intelligence may decrease the highest possible speed if the gain in network utilisation is acceptable.
6.2 Switching solution

The switch must be able to make a connection from any input to any output, if the restriction with no bouncing is not implemented, see section 4.2.2. This means that there will have to be five multiplexers with five inputs each. One input is allowed to be connected to one output only, an incoming connection can only be used once, no duplication. To accomplish this, a controlling unit makes decisions of what combination the setup of the multiplexers should have. The major development of the switch is made in this unit. The remaining units in the design are standard units such as buffers, incrementers, and decremented.

![Diagram of switch operation]

Figure 6.1: Basic switch operation.

A packet header includes a destination address, hop-counter and a flag telling the packet is an actual packet. The flag is called empty and is set to '1' if there is a packet on the input. The reason for using this flag is to make a simple packet indicator and to save power. Instead of removing a whole packet from an input buffer when no new packets are waiting, only the empty bit needs to be changed for notification. The control unit examines every incoming packet for its destination. From the destination address, the destination direction is derived. The destination address can be divided into two parts, the row address and the column address. When using relative addressing, the first bit in the row address is the sign for north or south and the remaining bits are simply the binary number of how many rows in that direction the destination row is. The same relation applies for the column address. If both the row and column address is zero, the packet has reached its final destination. For further explanation see section 4.7.

The most simple implementation of the control unit is to let one fixed input make its first choice of output and so on with the remaining inputs in a fixed order. For example, packets coming in from the north are given the opportunity to make the first choice, then East, South, West, and finally, if there are free outputs left, packets from the the resource, see the flowchart in appendix B. The pseudo code for routing a fixed input order to north is shown in figure 6.2. The full code of the final controlling logic called ctrl_box is presented in appendix C.

A number of if-statements goes through each input in a fixed order to see whether the packets on that input has a direction that corresponds to that particular output. This is however not the full solution. Assume that a packet
if north.busy = '0' then
if aim(north.dest)(north) = '1' and north.empty = '1' then
    north_select = north_input;
    north_busy = '1';
    north.empty = '0';
elseif aim(east.dest)(north) and east.empty = '1' then
    north_select = east_input;
    north_busy = '1';
east.empty = '0';
end if;
end if;

Figure 6.2: Pseudo code for the most basic operation.

Aims in a direction that is already used, with the above code only, packets will be dropped. Every output must be examined a second time and force the remaining packets in directions which contradict their destination direction. The second time, the if-statement with the aim-function removed and packets are forced into a direction in a specified order. See the pseudo code in figure 6.3.

if north.busy = '0' then
if north.empty = '1' then
    north_select = north_input;
    north_busy = '1';
    north.empty = '0';
elseif east.empty = '1' then
    north_select = east_input;
    north_busy = '1';
east.empty = '0';
end if;
end if;

Figure 6.3: Pseudo code for the second half of the most basic operation.

This implementation is very static. In this example, packets from north always has the advantage. A second stage implementation is to include more dynamic routing decisions. By introducing packet priority, the packet with the highest priority are given the opportunity to make the first choice, and then in descending order. In case of a conflict between two packets or more, the packet with the highest priority wins.

In chapter 4, priority was investigated. In the current implementation, the priority is equal to the hop-counter, which is in fact the number of hops since the packet started its journey on the network. This number is correlated to the time of departure and is used because of the simplicity of achieving a number for the priority without using expensive computing resources.

The implementation is similar to the code given previously, but instead of using fixed variables as north.empty, the four incoming packets are put into an internal matrix of packet variables and handlers such as inMatrix(0).empty is used. By using the row numbers in the matrix, it is possible to refer to the packets like pointers in C. On row zero in the matrix, the packet coming from
north resides. The zero is exchanged by a variable which is a number between zero and three. A sorting unit creates four integers which is row numbers of the packet with the highest priority, second highest, and so on in descending order. Instead of calling north.empty first, the packet having the highest priority is called by using inMatrix(prio_L.row).empty. This results in that the packet with the highest priority makes the first choice. See figure 6.4 for example.

```
if north.busy = '0' then
  if aim(inMatrix(prio1row).dest)(north) = '1' and inMatrix(prio1row).empty = '1' then
    north_select = prio1row;
    north_busy = '1';
  end if;
  inMatrix(prio1row).empty = '0';
else if aim(inMatrix(prio2row).dest)(north) and inMatrix(prio2row).empty = '1' then
  north_select = prio2row;
  north_busy = '1';
  inMatrix(prio2row).empty = '0';
end if;
```

Figure 6.4: Pseudo code where the input and output query is depending on priority.

The sorting unit uses the hop-counter to decide the incoming packet priority only. In case of two packets having the same priority, there will be one fixed direction of input that always wins. However, this may not seem to be a big issue, since it is a possible answer as to why the average load is not fully centered in the mesh, see figure 7.8 where this is further elaborated. Another way of dealing with packets aiming in the same direction is to make a random choice which packet that will be favoured at the moment. But as always, this has a cost in terms of speed and size.

The final implementation can be seen as a schematic in appendix A, observe the bit width on the inputs and outputs. The bit width is only from zero to 126 which is 127 bits, that contradicts to the previous size of 128 bits. The reason is that the payload used in the implementation are a number of BCD coded integers which have a size of four bits. It is of course possible to add an extra bit to achieve a bit width of 128 bits but the higher layers are not yet determined and the implementation is easy to adjust to adapt new preferences of packet layout.

### 6.2.1 With no stress value

In section 4.5, the load distribution was discussed. It has been revealed that the highest load will be in the centre of the network mesh. The amount of packets waiting in the network interfaces of the resources in the centre switches are limiting the availability for packets to be launched there. If many packets are traveling through the centre, packets leaving the centre resources will have to wait for a long time for an empty output where the new packet can be put. The average number of packets waiting for transmission must not increase over a longer period of time. An empty switch output is needed to release a packet from the resource on the network. This occurs when one incoming packet is forwarded to the resource, or when the number of incoming packets is less than four. If the possibility of a packet creation is greater than the possibility of an
empty output in that particular switch, then the number of packets waiting will increase.

The most basic operation is to ignore the hot-spot in the centre. Instead, an easy and natural solution is to avoid packets to travel through the centre by placing resources with frequent communication close to each other.

### 6.2.2 With stress value

The number of packets going through the very centre of the mesh can be spread over a larger area by sending load status information to the surrounding switches. That information aids surrounding switches to select paths through switches with lower load, see section 4.5.

During the whole thesis, the switch complexity is always kept to a minimum to save physical space but more important to be able to run the switch at the highest possible clock frequency. By adding small extra units may increase the network throughput even if the clock frequency is decreased. This is of course a trade-off between speed, size, and capacity. There are no general cases, what can be efficient in one network design may be bad in another, it depends on the nature of the system and the way it is implemented.

In the current implementation, the packet with the highest priority is given the opportunity to make its first choice of output. For a large network, there will most of the time be two possible outputs which are in the direction of the destination, if not the current row or column is the same as the destination. In the implementation where stress value was not used, the order of outputs the packet was suggested to be routed were fixed. When using stress value from the surrounding switches, the order of suggested outputs changes. Four stress values are imported on wires separate from the packets from the most adjacent switches and inserted into a sorting unit which outputs a vector where the outputs are ordered with the best choice of outputs first.

One could imagine that by adding extra logic to handle the output priority order with the sorting unit and the logic for generating the stress value will increase the gate depth of the whole switch. However, the stress value sorting unit is applied in parallel with the priority sorting unit while the stress value evaluation in the same way works in parallel to the output buffers. This results in a switch which has a larger area and more power consuming compared to the simpler one, but with the maximum clock frequency remained constant.

### 6.2.3 Averaged stress value

The stress value are sent from each switch to its four most adjacent switches. Each switch will also receive four stress values from the surrounding switches, in the previously described functionality, this is used once directly in the following switch cycle. The reaction to a high stress value is that the surrounding switches will avoid sending to the switch with the high stress value. The high load number may the next switch cycle rapidly change to a considerable lower value, which in its turn makes the surrounding switches a preferable output destination. The value increases again and will keep on oscillating in this way and so will the packets during its journey. A too low stress value in an area with high load, which could become the case after the switch reaction of a high stress value, will fool all the surrounding switches that the particular switch is a good choice of route.

The solution is to create an average of the stress value over a few clock cycles to prevent the surrounding switches of making bad routing decisions. It can be done on the stress value input on every switch but it is better to make the averaged stress value on the switch it concerns before it is transmitted. The

39
gain is that only one stress value averager per switch is needed. Instead of adding the stress value during for example four switch cycles, divide by four to get the average and round up to an integer. It could be found better to transmit the result after the adder without dividing. To better distinguish between two nearby values during the stress value sorting phase, a larger range of possible numbers may be used. This will need more connections between the switches but averaging logic is saved at the cost of larger stress values. The implementation using averaged stress values are averaging over four cycles used and a value between zero and sixteen is sent. With negligible effect, the maximum number of the stress value could be allowed to be maximum 15 instead of 16 in order to save an extra bit and handle four bits instead of five. However, the final implementation uses five bits for representing the stress value. The stress value averager in the implemented model is fairly small, about 200\(^1\) gates.

### 6.2.4 Gate depth

The maximum clock period has an upper limit which is related to the distance between the input and the output. When a clock pulse occurs, every signal in the design must stabilise before the next clock pulse. This restricts the number of following gates between input and output. If the gate depth exceeds the maximum depth, pipelining must be used, see section 5.1. Even though the simplicity of the switch has been a main objective, the implemented switch has a maximum gate depth between input and output, the critical path, of several times the maximum depth. In the design, this is not implemented, although the back-end tool supports pipelined signals which simplifies the positioning of the pipeline buffers. The negative effects of using pipelining, are that a packet can not be forwarded the next clock cycle. Instead, the number of internal buffers used for pipelining determines how many clock periods the switching occupies.

### 6.2.5 Synthesis

The synthesis of the switch was made using Synopsys together with the lsi10k-library. Today's technology are improved many times compared to the technology of the libraries used in the synthesised model. Still, the number of gates are likely to be in the same order but the delays extracted from the synthesis are much less.

The final switch has been synthesised using two different constraints, first the default which is optimised on area and second optimised on speed. When speed is considered, the area may be increased rapidly since extra logic is used in parallel for making the gate depth shorter.

In table 6.1, it is shown in what order the total combined logic and the critical path gate depth needed to accomplish the switch depending on the optimisation constraints.

<table>
<thead>
<tr>
<th>constraint</th>
<th>total combined logic</th>
<th>critical path gate depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimised for area</td>
<td>13 964</td>
<td>79</td>
</tr>
<tr>
<td>optimised for speed</td>
<td>21 029</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 6.1: Number of gates and gate depth with averaged stress value using the lsi10k-library.

The critical path is the path of the signal that take the longest time to progress. Hence, the shorter gate depth, the faster the result of the input

\(^{1}\)218 gates using the lsi10k-library.
signals is computed on the outputs.
Chapter 7

Simulation results

To achieve stable results from the simulations, it has been shown that the number of packets each switch must forward should be of more than about 5000, for a $3 \times 3$ network with the packet probability of 0.8, i.e. that the probability of how often a packet is transmitted, and the mean distance to and from each switch is $2/3 \times N = 3$ [9]. This results in the number of clock cycles the network must be simulated in is approximately:

$$\frac{5000}{3 \times \frac{2}{3} \times 0.8} \approx 3100 \quad (7.1)$$

The same calculation for a $32 \times 32$ network with the packet probability of 0.1, results in:

$$\frac{5000}{32 \times \frac{2}{3} \times 0.1} \approx 2300 \quad (7.2)$$

The time for simulating the small $3 \times 3$ network is fairly short, approximately five minutes. The time for simulating the large $32 \times 32$ network is longer, between five to ten hours, because of the increased number of switches.

Creation of a complete set of simulation results occupies great amount of time, which is far more than available. Therefore, most of the simulations are focused on the no stress value case. Rough graphs showing the maximum probability for various network mesh sizes can be seen in figure 7.1. To refine that graph, a lot more time must be spent on this type of simulations.

The simulation results are investigated for three different implementations, with no stress value, with stress value, and with averaged stress value over four cycles.

7.1 Maximum average distributed probability

Depending on the physical layout, packets can be put into frames. The frames do not necessarily have to be of the same size as the packets, for example, one packet can be exactly the size of two frames. For instance, a packet with the size of 128 bits over a connection of 64 bits results in a Network clock of half the Data Link clock. The packet probability is counted in packets which hereby is half the amount compared to the case when the relation between the Network clock and the Data Link clock are one. This is however not of importance as long as the statistics are related to the packet rate instead of the Data Link clock. The clock relation has to be a general setting over the whole network since each frame does not include the header information and mixing frames with and without header information must not occur. Hence, every packet must be reassembled in every switch.
7.1.1 Theoretical values

If the network is a quadratic mesh with one side $N$, the total number of switches and resources are $N^2$. Assume that in a cross-section in the middle of one side, the number of packet switches on either side is $N^2/2$. If the chance of transmitting a packet to any other destination is equal, the number of packets over the cross-section in one direction should be $N^2/4$. The average probability for a packet to be transmitted is $p$ for every resource and the number of connections in one direction over the cross-section is $N$. Hence, the number of packets must be less than the number of connections over the cross-section to be able to fulfill its service. This is a reason for using a symmetric mesh structure. The following relation is derived:

$$\frac{N^2}{4} \times p < N \Rightarrow p < \frac{4}{N} \quad (7.3)$$

Take for example a $6 \times 6$ network. To remain stable and to ensure that the number of connections between the two halves is enough, the average probability must be less than $4/6 = 2/3 \approx 0.66$. This is a big difference from the value of the probability achieved from simulations. An explanation to this difference can be that this is under the assumption that the load over the whole cross-section is equal. This is never the reality. In the network centre, there is a hot-spot through which packets tend to be routed, since it is the shortest and most natural path. For instance, a packet with its source and destination in two corners of the diagonal has its projected direction through the network centre. Therefore, the network gets choked in the center and the maximum possible probability is a lot less than the theoretical.

For a larger network, the boundary effects affect the network centre a lot less than for a small network of for instance a mesh with the size of $6 \times 6$ where 20 of 36 switches are edge switches. Although, it shows us that the maximum average probability is proportional to the inverse of $N$, see figure 7.2.
Figure 7.2: Maximum probability with no stress value compared to theoretical maximum of $4/N$.

The second limitation in the maximum number of packets that can be transported in the network simultaneously, is a result of the limitation in the output buffers of each switch. Since no queues are used for stacking packets in its optimal output direction, packets may be routed in less fortunate directions. Assume that every output buffer in the network is occupied by a packet. No more packets can possibly be inserted in the network until any packet has reached its final destination.

It has been determined earlier that the average distance $\bar{d}$ between any two switches is $2/3 \times N$. The number of packets in the network is a result of the probability $p$ of how often packets are inserted in the network. The number of buffers in the network is four times the number of switches, since there is one output buffer in each direction, even on edge switches. The number of packets in the network must as a result of the previous argument be less than the number of buffers in the network

\[ p \times N^2 \times \bar{d} < 4 \times N^2 \]  
\[ (7.4) \]

\[ p \times N^2 \times \frac{2}{3} \times N < 4 \times N^2 \]  
\[ (7.5) \]

\[ p \times \frac{2}{3} \times N < 4 \]  
\[ (7.6) \]

\[ p < \frac{6}{N} \]  
\[ (7.7) \]

The result from equation 7.7 is a significant limitation in size, packet probability, and the average distance. The average distance is the average number of clock cycles each packet is in the network. When the load increases, the distance increases. Most likely, the resource layout are performed with a sense of locality so that the actual mean distance of the normal data flow is less than the predicted.
7.2 Delivery time

Another way of examining the network load is to investigate how many extra steps every packet takes during its travel from the source to the destination switch. For a 12 × 12 network with a packet probability of 0.15, which is close to the highest possible using averaged stress value, the real distance expressed in percent of the shortest distance possible for each packet can be seen in figure 7.3.

![Bar graph showing number of packets versus percentage of shortest distance](image)

**Figure 7.3:** Real distance in relation to shortest distance for high load.

Compare figure 7.3 with figure 7.4, where the only difference in the incoming data is the lower packet probability in the second figure. Observe in figure 7.4 that most packets are delivered within 50% extra distance compared to the optimal path. This in relation to the figure with higher load where most packets are delivered after traveling a distance of 50% and more. As a result of the lower packet probability in the second figure, the total number of packets are of course lower while the axis in both figures are fixed.

During the simulations, the FIFO buffers from the resources are set to be very large, this is however not very interesting since there are normally one or zero packets in the FIFO buffer. The number of packets in the FIFOs are remained very few even when the packet probability is close to the maximum possible. When the maximum probability is reached, a very sharp edge in the increasing number of packets in the FIFO can be seen. The number of packets in the FIFOs increases rapidly and reacts fast to small changes in the packet probability. This is how the value for the maximum probability is extracted. This can clearly be seen in figure 7.5 where the packet probability is increased every 1000 cycle with the amount of 0.01 from 0.47 to 0.50. The last 2000 cycles, the average number of packets in the FIFO buffer of the centre switch in the 5 × 5 network is increasing without control.

The rapid increase of packets in the FIFO buffer waiting for a free output on the switch is a function of a high network load. When the number of packets being transported in the network is large, the packets may not always be routed the shortest path between the source and the destination. If the mean
Figure 7.4: Real distance in relation to shortest distance for intermediate load.

Figure 7.5: Number of packets in the centre FIFO buffer. The packet creation probability is increased by 0.01 every 1000 simulation cycle.

...distance increases as a result of the high load, the load itself increases since each packet stays longer in the network before its destination is reached. This is an explanation of why there is a very sharp edge where the path length increases and the packets never or after a very long time manages to reach their targets without being deflected.
7.3 Hop-count study

The hop-count study is similar to the delivery time study. In the hop-count study, the actual number of steps between source and destination are counted and presented in a graph. In the beginning of this chapter it was stated that the average distance between two switches in a $N \times N$-mesh is $2/3 \times N$, under the assumption that the distance to the switch itself is zero. Using this argument, the hop-count graph should have a mean value of $2/3 \times N$, however, the switches implemented are not able to transmit packets to itself without leaving the switch and return first in the following cycle. If the destination of a packet is the same as the source, the hop-count will end on two instead of the more natural distance, zero. In addition to this, two extra steps are added since the hop-counter is increased by one when the packet leaves the resource to the switch and then another one when it is forwarded from the switch to the resource. In figure 7.6 the packet probability is low enough to keep the number of deflected packets low, this is to achieve the shortest path possible for every packet.

![Figure 7.6: Number of hops for every packet during simulation. The y-axis show how many packets that arrives in the number of steps read on the x-axis.](image)

The mean value from figure 7.6 can be estimated to be between 14 and 16 (the exact expected value value is 15.3338 according to Matlab), subtract with two for the extra steps from resource to switch and back and the value is between 13 and 15. The network size in the figure is $20 \times 20$ which has the expected average distance $2/3 \times 20 \approx 13.3$. The estimated value corresponds to the expected. The graph in figure 7.6 has the form of a $\chi^2$-distribution with the expected value as previously mentioned [12]. This is the expected distribution of the product of two random numbers with uniform distribution, according to [12].
7.4 FIFO-buffer study

To see how the load is distributed around a hot-spot, the study of all FIFOs in the network mesh is a good approach. The study is made in two steps. First, make a visualisation of the average number of packets in every FIFO. An example is shown in figure 7.7. Unfortunately, the scale is not shown clearly. The maximum average load is in the switch on row three and column five counted from the upper left corner and has a value of slightly more than 0.01. This tells us that the load is low in the network since it can be interpreted as a case where the FIFO is occupied by one packet every 100 switch cycle, which is not very much.

![Example of packets in the resource FIFOs](image)

Figure 7.7: Example of packets in the resource FIFOs.

Second, switches which from the first step are shown to have a high load can be further investigated. Instead of using the mean value, a time dependent packet counting can be used to discover FIFO behaviour. The type of resulting graph has already been introduced in figure 7.5.

The intention is now to show the real difference between the three cases of using stress value.

- With no stress value
- With stress value
- Averaged stress value

In the three cases, the same input data has been used. The mesh size is quite large, 16 × 16, to be able to spread the load over a larger area without too much influence of the boundary effects. The packet probability is for the 'no stress value' case close to the maximum possible, see figure 7.2, to create as much congestions as possible, to be able to make a clear separation where the use of stress value really brings matters to its head. Observe the scaling of the x-axis since every bar in the whole figure is scaled from the switch with the maximum average load.

49
### 7.4.1 With no stress value

In the previous example, no stress values are used for equalising the load over the mesh. The hot-spot is pushed to the north east corner, if north is up in the figure, which is the logical reference direction. The reason for the non-centered load is a result of the routing decisions in the switch. If two packets arrive to an intended destination at the same time, one packet is captured by the resource while the other is deflected. In this case, it is a fixed choice of deflection output order. First east, then north, west, and last south.

Figure 7.8 shows the similar behaviour as figure 7.7, but the fixed choice of deflection output order is in a different order.

![Image of average load in FIFOs without using stress value]

Figure 7.8: Average load in FIFOs without using stress value. The largest average number is 3.2.

### 7.4.2 With stress value

The stress value notifies the surrounding switches about how many packets the switch handles during that cycle. The stress value is updated every clock cycle and is not dependent on the previous values. Using stress value in this manner increases the performance of the switch since the outputs are ordered in the most preferable order. Compare the maximum value on the x-axis in figure 7.9, which has a maximum value of 0.9 to the value in figure 7.8, which is 3.2. The number of packets waiting in the buffers has decreased with approximately a factor of 3.5 by using stress value.

### 7.4.3 Averaged stress value

The averaged stress value is the sum of the four last stress values, see section 6.2.3. The number of switches that presents a high load is greater than before. The load is therefore more distributed over the mesh when the averaged stress value is used. In comparison to the previous figures, the load is not only close to maximum in a few switches, instead, a larger area of the network shares the work of transporting packets through the centre zone. From the figure, the maximum average load can be found by estimating the value of the axis at the
Figure 7.9: Average load in FIFOs using stress value. The largest average number is 0.9.

Figure 7.10: Average load in FIFOs using averaged stress value. The largest average number is 0.15.

right end of the x-axis. With experience of the visualised data, the maximum average load is estimated to be 0.15. Compared to the implementation using stress value with no averaging, the average load now achieved is six times less compared to the non averaged stress value. In relation to the most basic implementation where no stress value was used, it is enhanced with a factor of more than 20.
7.5 Stress value analysis

It can clearly be seen from the previous discussion that the implementation of a more balanced load using stress values increase the network throughput and decrease the packet delivery time. The network load is decreased with a factor of 20 for a heavily loaded network. When the network becomes more loaded, all outputs are occupied and the output priority makes no effect since there are always packets in all buffers. In a less loaded network, the stress value together with the output priority order are not necessary since most packets gets routed the shortest path anyway and almost always there are free outputs in every switch every clock cycle. If there are no packets waiting in the FIFO buffers, the gain of using stress value are not as evident as before.
Chapter 8

Conclusion

While entering the billion-transistor era, it is a fact that the development time for new designs increases as the number of transistor functions rises. It is natural to realise the need for a design methodology that ensures a final design with high reliability in short time. This is possible using the concept of Network on Chip, where the strength lies in the resource and data packet information flow structure. The simulation results shows how the network on chip can be improved in a few steps. The result of using stress values for improving the network throughput is clearly shown. The gain of using the stress value for one period only compared to a simulation with no attempt for load distribution is shown to be of about three times. By using an averaged stress value over four cycles, the network load is decreased further by a factor of six. Hence, the product of using averaged stress values in relation to a case where no stress value at all was used is, for the network with the size of 16 x 16 switch-resource pairs and the packet probability of 0.15, a gain in the order of 20 times. The area added to accomplish the stress value calculation and the processing of the order of output priority related to stress values, is low compared to the gain in the network performance. No increase in gate depth has been noticed when adding the extra facilities for regarding stress values, since the processing is made in parallel to the existing basic model.

Figure 8.1: Slow and fast model of serial adders.

The use of the Synopsys software has been limited, as the gate delays in the latest technologies is not reflected. However, the number of gates in the final design ought to be in the same or similar order as the result from Synopsys. This renders the possibility to get a landmark of the resulting design size. A deduction of the number of gates in relation to the number of gates in the critical path for two different synthesising efforts is shown in table 6.1. After a brief look on the critical path, it is shown that the worst path is through the load averager. This is because of the nature of the implemented averager, the stress value is clocked through three registers and consequently delayed. The input value is added together with the output from each register and the result is used as the stress value. This can however be enhanced simply. In figure 8.1, the left figure is the one implemented. However, the figure to the right is showing how...
a faster adder could be implemented. Although it is not made in the master thesis project, it is one of the first things that should be reflected in future work.

8.1 Future work

The work on developing a set of protocols that comprehends the concept of the NoC is a continuous process. Regarding the master thesis itself, it would be an idea to investigate further how the load distribution can be even better enhanced. The improvement in the load averager to shorten the worst critical path is an obvious approach for the future work.

Also, implementing routines for packets traveling far distances that could be forced to be routed along a path around a highly congested area. To include regions and examine how the locality of resources affects the network behaviour. It could also be interesting to see what effect different types of data flow makes. For instance, bursty data traffic between two fixed points, compared to continuous data with perhaps lower bandwidth requirements. Simulation and investigation of the result of using virtual circuits is another issue that is needed to scrutinise.

The displacement of the hop-count has been compared to the $\chi^2$-distribution. This gives the opportunity of making simulations at a higher level, which would increase the simulation speed. This should also be looked into more deeply.
Bibliography


Appendix A

The final implementation

Figure A.1: The final implementation in modules.
Appendix B

Flowchart of basic model

Figure B.1: Chart over model when no stress value is used.
Appendix C

The control box, VHDL-code

library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.noc_package.all;

entity ctrl_box is
  port ( iNorth, iEast, iSouth, iWest, iResource :
in packet_type;
prio_1_row, prio_2_row, --rownumber to the relative priority
prio_3_row, prio_4_row :
in integer range 0 to 3;
OutputPriorityOrder :
in int_vector_0to4 (3 downto 0);
reset :
in std_logic;
Resource_Ready :
out std_logic;
N.select, E.Select, S.Select, --controlling multiplexers
W.select, R.select :
out integer range 0 to 4;
N.empty, E.empty, S.empty, --marking inputs
W.empty, R.empty :
out std_logic;
switchload :
out integer range 0 to LoadLength);
end ctrl_box;

architecture behaviour of ctrl_box is
begin
  --generate stress value
  process ( iNorth, iEast, iSouth, iWest )
  variable sum : integer range 0 to 4;
  begin
    sum := 0;
    if iNorth.empty = '1' then
      sum := sum + 1;
    end if;
    if iEast.empty = '1' then
      sum := sum + 1;
    end if;
    if iSouth.empty = '1' then
      sum := sum + 1;
    end if;
    if iWest.empty = '1' then
      sum := sum + 1;
    end if;
end process;
end;
sum := sum + 1;
end if;

switchload <= sum;
end process;

process ( reset, iNorth, iEast, iSouth, iWest, iResource,
prio_1_row, prio_2_row, prio_3_row, prio_4_row,
OutputPriorityOrder)
variable aimMatrix : std_logic_vector(3 downto 0);
variable inMatrix : packet_vector(3 downto 0);
variable R_aim : std_logic_vector(3 downto 0);
variable switched : std_logic_vector(3 downto 0);
variable outputA, outputB,
outputC, outputD : integer range 0 to 3;
variable selectvector : int_vector_0to4 (3 downto 0);
begin
  if reset = '1' then
    N_empty <= '0'; N_select <= 0;
    E_empty <= '0'; E_select <= 0;
    S_empty <= '0'; S_select <= 0;
    W_empty <= '0'; W_select <= 0;
    R_empty <= '0'; R_select <= 0;
    Resource_Ready <= '1';
  else
    switched := "0000";
    selectvector(3) := 0;
    selectvector(2) := 0;
    selectvector(1) := 0;
    selectvector(0) := 0;

--insert incoming packets in matrix
  inMatrix(North) := iNorth;
  inMatrix(East) := iEast;
  inMatrix(South) := iSouth;
  inMatrix(West) := iWest;

--get aiming direction
  aimMatrix(North) := get_aim ( iNorth.d_addr );
  aimMatrix(East) := get_aim ( iEast.d_addr );
  aimMatrix(South) := get_aim ( iSouth.d_addr );
  aimMatrix(West) := get_aim ( iWest.d_addr );

--route packet to resource if exists
  if aimMatrix(prio_1_row) = "0000" and
     --destination resource
    inMatrix(prio_1_row).empty = '1' then
    --real packet
    R_select <= prio_1_row;
    inMatrix(prio_1_row).empty := '0';
  elseif aimMatrix(prio_2_row) = "0000" and
           --empty packet
    inMatrix(prio_2_row).empty = '1' then
    --mark resource output busy
    R_select <= prio_2_row;
    inMatrix(prio_2_row).empty := '0';
  end if;
R_empty <= '1';
elseif aimMatrix(prio_3_row) = "0000" and
  inMatrix(prio_3_row).empty = '1' then
  R_select <= prio_3_row;
  inMatrix(prio_3_row).empty := '0';
  R_empty <= '1';
elseif aimMatrix(prio_4_row) = "0000" and
  inMatrix(prio_4_row).empty = '1' then
  R_select <= prio_4_row;
  inMatrix(prio_4_row).empty := '0';
  R_empty <= '1';
else
  R_empty <= '0';
  R_select <= 0;
end if;

--setup the Output priority
outputA := OutputPriorityOrder(3); --update OutputPriorityOrder
outputB := OutputPriorityOrder(2);
outputC := OutputPriorityOrder(1);
outputD := OutputPriorityOrder(0);

--switch priority 1 to output
if inMatrix(prio_1_row).empty = '1' then  --packet exists
  if aimMatrix(prio_1_row)(outputA) = '1' then --packet aiming outputA
    selectvector(outputA) := prio_1_row;
    switched(outputA) := '1';
  elsif aimMatrix(prio_1_row)(outputB) = '1' then --packet aiming east
    selectvector(outputB) := prio_1_row;
    switched(outputB) := '1';
  elsif aimMatrix(prio_1_row)(outputC) = '1' then
    selectvector(outputC) := prio_1_row;
    switched(outputC) := '1';
  else
    selectvector(outputD) := prio_1_row;
    switched(outputD) := '1';
  end if;
end if;

--switch priority 2 to output
if inMatrix(prio_2_row).empty = '1' then  --packet exists
  if aimMatrix(prio_2_row)(outputA) = '1'
    and switched(outputA) = '0' then
    selectvector(outputA) := prio_2_row;
    switched(outputA) := '1';
  elsif aimMatrix(prio_2_row)(outputB) = '1'
    and switched(outputB) = '0' then
    selectvector(outputB) := prio_2_row;
    switched(outputB) := '1';
  elsif aimMatrix(prio_2_row)(outputC) = '1'
    and switched(outputC) = '0' then
    selectvector(outputC) := prio_2_row;
    switched(outputC) := '1';
  elsif aimMatrix(prio_2_row)(outputD) = '1'
7
and switched(outputD) = '0' then
selectvector(outputD) := prio_2_row;
switched(outputD) := '1';
elseif switched(outputA) = '0' then --desired aim not free, switch oA if free
selectvector(outputA) := prio_2_row;
switched(outputA) := '1';
elseif switched(outputB) = '0' then
selectvector(outputB) := prio_2_row;
switched(outputB) := '1';
elseif switched(outputC) = '0' then
selectvector(outputC) := prio_2_row;
switched(outputC) := '1';
else
selectvector(outputD) := prio_2_row;
switched(outputD) := '1';
end if;
end if;

--switch priority 3 to output
if inMatrix(prio_3_row).empty = '1' then --packet exists
if aimMatrix(prio_3_row)(outputA) = '1' and switched(outputA) = '0' then
selectvector(outputA) := prio_3_row;
switched(outputA) := '1';
elseif aimMatrix(prio_3_row)(outputB) = '1' and switched(outputB) = '0' then
selectvector(outputB) := prio_3_row;
switched(outputB) := '1';
elseif aimMatrix(prio_3_row)(outputC) = '1' and switched(outputC) = '0' then
selectvector(outputC) := prio_3_row;
switched(outputC) := '1';
elseif aimMatrix(prio_3_row)(outputD) = '1' and switched(outputD) = '0' then
selectvector(outputD) := prio_3_row;
switched(outputD) := '1';
elseif switched(outputA) = '0' then
selectvector(outputA) := prio_3_row;
switched(outputA) := '1';
elseif switched(outputB) = '0' then
selectvector(outputB) := prio_3_row;
switched(outputB) := '1';
elseif switched(outputC) = '0' then
selectvector(outputC) := prio_3_row;
switched(outputC) := '1';
else
selectvector(outputD) := prio_3_row;
switched(outputD) := '1';
end if;
end if;

--switch priority 4 to output
if inMatrix(prio_4_row).empty = '1' then --packet exists
if aimMatrix(prio_4_row)(outputA) = '1'
and switched(outputA) = '0' then
  selectvector(outputA) := prios_row;
  switched(outputA) := '1';
elsif aimMatrix(prios_row)(outputB) = '1'
  and switched(outputB) = '0' then
  selectvector(outputB) := prios_row;
  switched(outputB) := '1';
elsif aimMatrix(prios_row)(outputC) = '1'
  and switched(outputC) = '0' then
  selectvector(outputC) := prios_row;
  switched(outputC) := '1';
elsif aimMatrix(prios_row)(outputD) = '1'
  and switched(outputD) = '0' then
  selectvector(outputD) := prios_row;
  switched(outputD) := '1';
elsif switched(outputA) = '0' then
  selectvector(outputA) := prios_row;
  switched(outputA) := '1';
elsif switched(outputB) = '0' then
  selectvector(outputB) := prios_row;
  switched(outputB) := '1';
elsif switched(outputC) = '0' then
  selectvector(outputC) := prios_row;
  switched(outputC) := '1';
else
  selectvector(outputD) := prios_row;
  switched(outputD) := '1';
end if;
end if;

--switch packet from resource if free output exists
if switched = "1111" then
  --all outputs busy
  Resource_Ready <= '0';
else
  Resource_Ready <= '1'; --ready for a new packet
  if iResource.empty = '1' then
    R_aim := get_aim ( iResource.d_addr );
    if R_aim(outputA) = '1' and switched(outputA) = '0' then
      selectvector(outputA) := Resource;
      switched(outputA) := '1';
    elsif R_aim(outputB) = '1' and switched(outputB) = '0' then
      selectvector(outputB) := Resource;
      switched(outputB) := '1';
    elsif R_aim(outputC) = '1' and switched(outputC) = '0' then
      selectvector(outputC) := Resource;
      switched(outputC) := '1';
    elsif R_aim(outputD) = '1' and switched(outputD) = '0' then
      selectvector(outputD) := Resource;
      switched(outputD) := '1';
    elsif switched(outputA) = '0' then
      selectvector(outputA) := Resource;
      switched(outputA) := '1';
    elsif switched(outputB) = '0' then
      selectvector(outputB) := Resource;
      switched(outputB) := '1';
    elsif switched(outputC) = '0' then
      selectvector(outputC) := Resource;
      switched(outputC) := '1';
    end if;
  end if;
end if;
switched(outputB) := '1';
elsif switched(outputC) = '0' then
    selectvector(outputC) := Resource;
    switched(outputC) := '1';
else
    selectvector(outputD) := Resource;
    switched(outputD) := '1';
end if;
end if;
end if;

--update empty identifier
    N_empty <= switched(North);
    E_empty <= switched(East);
    S_empty <= switched(South);
    W_empty <= switched(West);
    N_select <= selectvector(North);
    E_select <= selectvector(East);
    S_select <= selectvector(South);
    W_select <= selectvector(West);
end if;
end process;
end;