Multiprocessor in a FPGA

Nikolaj Dalgaard Tørring

Kongens Lyngby 2007
IMM-B.Sc-2007-10

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

Computer chips are becoming increasingly more complicated with whole systems and multiple processors on a single chip. I have design and implements such a SoC with multiple processors.
To keep down the work load, the processor and some peripheral units are taken from the community at OpenCores.org. This will ensure that no problems with the processor occur since this have been tested thoroughly and is known to work. The same applies for the UART which is used to verify that the system runs correctly at the FPGA. These uses a WISHBONE interface and therefor this have been adapted.
Synchronization unit and network component have been designed from scratch. Within the network aspect such as routing- and forwarding strategy has to be decided along with typology designs.
The design process have been split up in five steps, starting with just a connection between memory and processor, each step adding a new aspect. At the end a multiprocess system which uses NoC are designed. Two different network typologies have been designed thereby making two systems with NoC. Additionally a bus from the OpenCores community have been used, to design a multiprocessor system. The bus have been tested and proven working, which means that it can be used to verify everything else works as intended in a multiprocessor system, before the NoC was developed.
Finally the result from the multiprocessor systems are discussed and compared to find out how well the designed NoCs are working and what could be done better.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark as part of the requirements for acquiring the B.Sc. degree in engineering.

The thesis goals was to implement a multiprocessor in a FPGA. To connect the IP cores a Network-on-Chip was used. Different aspects of Network-on-Chip and parallelism had to be dealt with such as shared resources, synchronization, deadlocks and routing problems.

The report documents the design and implementation of a Network-on-Chip based multiprocessor system. From the beginning with peripheral units to the design of network components such as the routing node and the design of network typologies. Further more the designed components and system are discussed and compared.

Lyngby, June 2007

Nikolaj Dalgaard Tørring

Acknowledgements

I thank my supervisors Jens Sparsø for his support, guidance, ideas, and the beneficial discussions we had, as well as his knowledge within the domain.

I would also like to thank the the Ph.D students Morten Sleth Rasmussen, Matthias Bo Stuart and Mikkel Bystrup Stensgaard, for always be willing to take their time to help me with the problems i had.

Contents

Chapter 1

Introduction

Computers are a part of our daily life to a greater extent than most people will ever realize, they occur on our work and in our mobile phone, but it also turns up in our car, in our refrigerator and our living room. We use use computers to a higher degree for things that they were not intended for when they were first invented, doing highly complex arithmetical calculations. Chips have become more complex than ever containing whole systems on a chip, called System-on-Chip (SoC).
By now chips are used in the most products on the market, creating a demand for cheaper, faster and easier ways to design new chips. This have created a market where companies buys Intellectual Property cores(IP cores) from each other, being a computational part, memory, a I/O controller or something else. This makes it possible to design a whole SoC just connecting IP Cores. As source code is often available for IP Cores, field-programmable gate array(FPGA) is a suitable platform to develop such a design.
The demand for more computational power increases as designers come up with new and more powerful chips, making it a endless pursue for for a faster chip. In this pursue multiprocessing came up, making it possible to multi-task, or do more thing at one time. At first it was shown as multiple chips connected, but by now we see these multiprocessors on a single chip, multi processing is discussed in further details in section 1.1.
As SoC becomes more complex the shared bus connecting the IP Cores often shows up a the bottleneck, limiting the traffic between the IP Cores. To solve

this problem Network-on-Chip(NoC) was developed, allowing multiple IP Cores to communicate at a time. The idea resembles a lot like the familiar computer network and can, to a great extent, be compared with it, but it does have some different requirements given that it is used on chips.
The specified project was very open, as it was basically to design a multiprocessor and implement it on a FPGA. There was no requirements for the processor or the interconnection, also it was not specified what the multiprocessor should do. Chapter 2 will narrow down the project and the requirements for the multiprocessor.

1.1 Multi processing

A multi processor system is all about getting more performance than a single processor is able to deliver. As mentioned earlier in chapter 1, a multi processor system consists off two or more processors connected some how so they are capable of communicating. This could be on a single chip where the processors are connected typical by either a bus or NoC. Alternatively the multiprocessor system can be in more than one chip, typical connected with some sort of bus, each chip can then be a multiprocessor system. A third option is a multiprocessor system over more than one computer, here they are typical connected by a network, again each computer can contain more than one chip that can contain more than one processor. An example of such a system is folding@home[7] with about 200.000 processors, also most modern supercomputers are built up this way.
Making a multiprocessor that works is not an easy task, a lot of questions and problems arise when considering multiprocessing. Making a it work well is even more difficult. It will obviously not be faster to have two processors calculate the result of 2+2, so to be able to take advantage of multiple processors effectively some parallelism is necessary. Making a system parallel is when it is presented with more than one task, known as *threads*. This means that it is important to spread the workload over all the processor, keeping the difference in idle time as low as possible. To be able to do this it is important to coordinate the work and workload between the processors, here it is specially important to take into consideration if some of them are special purpose IP cores. To keep a system with N processors effective it has to work with N or more threads, so that each processor have something to do all the time.
Also it is necessary for the processors to be able to communicate with each other, this is usually done by have some shared memory in which they can store value that other processors then can use. This arise a whole new problem of thread-safety. When this is violated two processors(working threads) access the same value at the same time. Consider the code.

```
1       x = x+1;
```

Having two processors `P1` and `P2` executing this code, there is a number of different outcomes, due to the fact that the code will be split up in three parts.

```
2       l1: get x;
3       l2: add 1 to x;
4       l3: store x;
```

It could be that `P1` will first execute `l1`, `l2` and `l3` and afterward `P2` will execute `l1`,`l2` and `l3`. It could also be that P1 will first execute l1 followed by P2 executing `l1` and `l2`, giving another result. Therefor some methods for restricting access to shared resources are necessary, known as *thread safety* or *synchronization*.
Also it is necessary for each processor to have some private memory, where the processor does not have to think about thread safety to speed up the processor. As an example each processor need to have its stack private.
The benefits of having a multiprocessor is:

- Possible faster calculations.
- More responsive system.
- Ability to have different processors for different tasks.

The drawback of having a multi processor is:

- Many pitfalls.
- Not necessarily faster.
- Multi threaded programs are harder to make than single threaded programs

1.2 Report outline

At first the end goal of the project is specified in chapter 2. It describes different design steps, adding a bit in every step. At the end a fully working multiprocessor build on a NoC should be running. After the goal of the project is outlined,the component needed are described. First the project used from the community at OpenCores.org are descriped in chapter 3. In chapter 4 the

designed components for this project is described, first the peripheral units followed by the NoC components in section 4.3. In this section both some further NoC theory is given along with a description of the designed components. The project is split up in steps and in chapter 5 the design and verification of these steps are looked into. Chapter 6 discuss the results and experiences from the project and looking into how to make a good NoC design. It is then rounded of with a conclusion in chapter 7.

Chapter 2

Specification

As stated in section 1 the project was at the beginning very open, in the section it will be narrowed down. In a multiprocessor system a lot of things needs to be done and a focus could be placed on many interesting subjects, but being only a single person in the group one should be careful not to make the project to large.
The goal is a system capable of running some sort of multi-threaded program. As interconnection a NoC is to be used. To keep the NoC as simple as possible a best-effort routing is chosen and as far possible it should be kept dead-lock free. The design shall be implemented on a FPGA. Besides the processors the system shall contain one or more memory modules, a UART for communication, and a synchronization unit, for thread safety. These peripheral IP cores shall be memory mapped into a shared memory for access.

To complete these goals the development process have been split up in part. The first part is illustrated in figure 2.1. It contains only a single processor and a single memory module. The goal of this part is to connect these and make the connection work.

The next part is illustrated in figure 2.2. It is just as the first part, but an

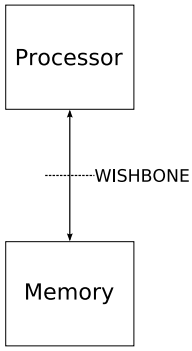

Figure 2.1: Step 1. Containing a memory and processor connected with the wishbone interface.

UART have been added. A switch connects both the memory and the UART with the processor. The goal of this part is add an UART to the system and thereby adding off board communication.

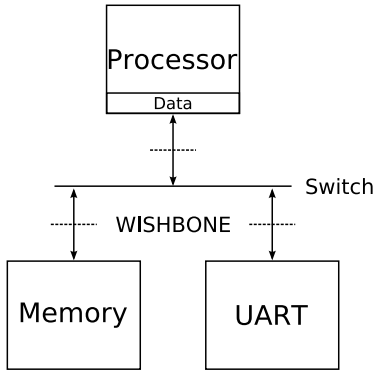

Figure 2.2: Step 2. Adding a UART to the data connection with a bus.

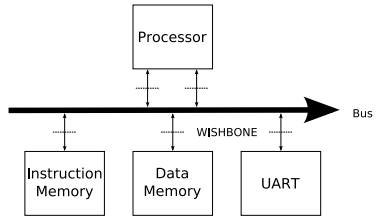

Figure 2.3: Step 3. Communication through a bus.

The third part uses a bus instead of the switch connecting the UART and memory with the processor, this is illustrated in figure 2.3. The goal of this part is to make sure the system works with the bus, and thereby make it ready for multiple processors.

Next the system becomes a multi processor system, adding another processor. This is illustrated in figure 2.4. Additional to the extra core a synchronization unit is added. By doing this it is assured that everything works with a multi processor.

Finally there is "only" left to add the NoC as shown in figure 2.5. Making the

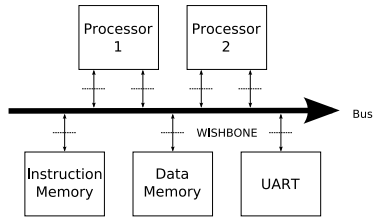

Figure 2.4: Step 4. Multiple processors communication through a bus.

system fulfill the specification. The structure of the NoC is there no requirement for, which is why it is not shown of the figure.

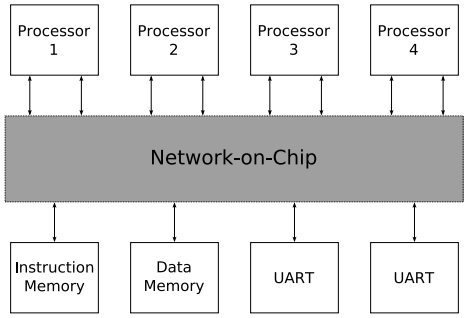

Figure 2.5: Step 5. Multiple processors communication through a network.

Chapter 3

IP cores from OpenCores.org

3.1 Introduction

Just like there for many years have existed a community of people dedicated to developing software available to the general public, there also exists such a community for IP cores at OpenCores.org focusing on freely available, freely usable and re-usable open source hardware[6].
From OpenCores.org a series of IP cores and standard are used, the benefit from this approach is, that it is not necessary to spend time designing these. Instead the time can be used on design and development of the essence of this project, NoC. All IP cores from OpenCores.org is fetched in February 2007.

3.2 WISHBONE

WISHBONE is an interface between IP cores, designed by the community at OpenCores.org to ease the integration of different IP cores. It is designed to be easy to use and adapt in projects and highly scalable so it could be used in large projects[4].
It is built up on a master-slave connection supporting multiple master(multiprocessing),

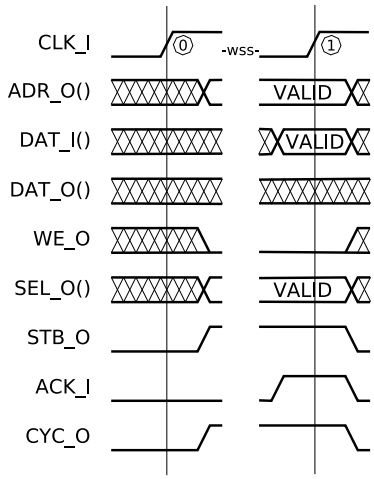

Figure 3.1: The read cycle in a WISHBONE connection.

making it highly usable in this project. End design is primarily up to the end user, with handshaking protocol, single clock transfers, modular address and data width and support for different interconnection structures(point-to-point, shared bus, ect.).
In this project master IP cores are not designed which is described in section 3.3, this means that only the slave interface is in focus here. [4] specifies the slave core must always qualify the `ACK_O`, `ERR_O` or `RTY_O`:`DAT_O()`.
The most commonly used data transfers in this project is the read and write. Figure 3.1 illustrates a read cycle, started by the master core, holding a valid address on `ADR_O()`, setting `CYC_O` and `STB_O` active and `WE_O` inactive representing a read, `SEL_O()` is set to indicate either a load word or load byte. The master then listens on `ACK_I` for the respond and is ready to load the data. When the slave has data ready it responds by setting `ACK_O` according to `STB_I`, presenting the valid data on `DAT_O()`. I the next clock cycle the master sets `STB_O` and `CYC_O` inactive, as does the slave with the `ACK_O` signal.

The write cycle handshaking protocol is also lead by the master, by setting `STB_O` and `CYC_O` active, also `WE_O` is set active to represent a write cycle. As with the read protocol `SEL_O` is set to specify where the data is and of course valid data must be present on `DAT_O` and an address on `ADR_O`. The slave stores the data, and when stored, `ACK_O` is set as respond to `STB_I` on the following clock cycle. At the end both `STB_O` and `CYC_O` is negated by the master and `ACK_O` is negated by slave. Figure 3.2 illustrates the write cycle.

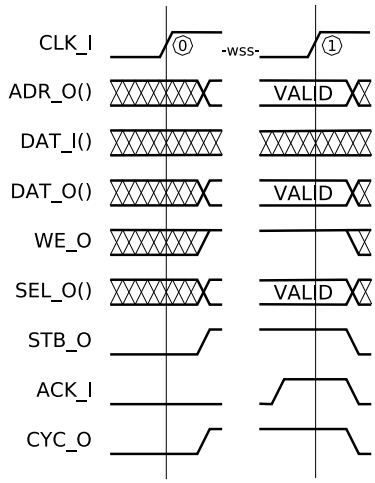

Figure 3.2: The write cycle in a WISHBONE connection.

3.3 OpenRISC 1000

3.3.1 The architecture

OpenCores.org have designed their own open processor architecture called OpenRISC 1000 (OR1k)[5]. The architecture is design with performance, simplicity scalability and low power in mind. It specifies a full 32/64-bit load and store RISC architecture. Some of the main features are:

- A completely open and free architecture.
- WISHBONE interface
- Cache system
- 5 stage pipelined
- OpenRISC Basic Instruction Set (ORBIS32/64)
- A flexible architecture definition
- A linear, 32-bit or 64-bit logical address space with implementation-specific physical address space.
- Branch delay slot for keeping pipeline as full as possible.
- Optimized for use in FGPAs and ASICs.

3.3.2 The implementation

OpenRISC 1200 (or1200)[2] is the first implementation of of the or1k architecture. It is highly configurable with a optional

- Instruction cache
- Data cache
- Instruction MMU
- Data MMU
- arithmetic units such as multiplier, divide

To fulfill any requirements for the processor, being high speed, low space or low power.
In this project the main focus is not to design a high speed or low powered processor, what we do need is a SoC with more than one processor so keeping it as small as possible is crucial, therefor all off above have not included in this project.

3.3.3 The tools

One of the great benefits with the OR1k architecture is that it is very well documented and comes with a large range off tools. The GNU tool chain have been ported to the OR1k architecture including GCC, GNU Binutils, Newlib and GDB, but is only available for the 32-bit OR1k. In this project the C compiler from GCC and the linker and assembler from GNU Binutils are used to compile and link the C test files.

3.4 UART

The OR1200 described in section 3.3 comes with a WISHBONE compatible NS16550A UART, supporting both 32 and 8 bit data bus[3]. It contains both a receive and transmit FIFO. Since the FPGA board used, only make use of the serial input signal(`SRX_PAD_I`) and serial output signal(`STX_PAD_O`), the rest of the external connections are unconnected if it is a output port or hardwired to `'0'` if it is a input port.

3.5 Conbus

Before designing a NoC interconnection, the conbus was used to verify that everything else worked as intended. The conbus is a WISHBONE compliant interconnection core, connecting up to 8 masters with up to 8 slaves. It uses a round robin arbiter. The implementation requires that the address mapping is defined to work properly. This is done by defining the most significant bits and how many bits to compare.

Chapter 4

Specification and design of IP Cores

4.1 Memory

For the memory, the core generator program in ISE webpack from Xilinx is used, this program is used to generate netlist for IP cores. The generated memory core is mapped directly in the FPGA's RAM modules so it does not use up logic, thereby saving space. Further more it is possible to generate the memory with default value so it is possible to hardcode a program into it. Unfortunately the generated cores are not WISHBONE compatible which means a WISHBONE-wrapper for the memory had to be designed, this can be found in appendix A.1.1. From section 3.2 it is described that Wishbone uses a handshake protocol and it is also possible to set the generated memory core to use a handshake protocol which could have made it easy to implement the wrapper. The memory core did however always signal that data was ready on the output port whether or not the data actually was valid. Even if there was no request for data, the memory IP core signaled that the data was ready, which made the handshaking useless. The wrapper was then designed as a state machine. It was discovered that it only took one cycle for the data to be ready. This meant if the request is a read, the `ack`-signal could be set high in the next state. With a write request there is difference, if it is a byte or a word that shall be written. If it is a word, it is as simple as writing the word. If however a byte had to be written the

current word had to bee fetched from the memory and then updated with the byte, before it is rewritten into the memory, making the store byte instruction taking two cycles. This is because the memory module only support to write words.

4.2 Synchronization

To deal with the aspect of concurrency in a multiprocessing system some means of synchronization is required. These synchronization operations has to be atomic. The WISHBONE interface supports read-modify-write(RMW) cycles[4] used for semaphore operations, unfortunately the OR1200 processor does not make use of this. Since the processor has no support for synchronization some other means has to be used. One possibility is to extend the processor to supporting the RMW cycles, this would however require a lot work. An easier way to support this is to design an peripheral unit to handle this instead.
The semaphore is a protected variable used to restrict access to shared resources. To manipulate a semaphore two functions, besides an function to set an initial value, is available, `P` and `V`. The `V` function is a non-blocking function that increases the value of the semaphore, the `P` function blocks while the value of the semaphore is zero. This way N equivalent resources can be controlled in a semaphore with initial value N. A semaphore design to control access to a single equivalent,is called a binary semaphore, which is perfect for this project since there are not any equivalent resources.
Memory mapping a semaphore unit into the system allows for multiple processors sharing the same address space by providing the necessary synchronization. To interact with the semaphore, the processors must use normal read and write requests. As the `V` function is non-blocking it should reached with a write request, which is also non-blocking. A read request however blocks until a respond is received which fits perfectly for the `P` function.
When a `P` is performed and the resource is not free, the processor has to wait until the resource is released. This could be done in a queue, holding identification for the processor. This queue has to be as long as the number of processors minus one, so one processor could hold the semaphore and the rest be in the queue for the semaphore. Such a queue is required for every semaphore. This is a very fast and effective solution but also rather expensive.
A more software oriented approach would be to use a spinlock or busy-wait. In this case the processor will be held busy waiting or spinning, where it repeatedly checks to see if the semaphore is free. This way the test-and-set function `P`is used, with read requests. The value of the semaphore is always returned, meaning that when the semaphore is free the value one is returned and at the same time the semaphore is taken, now holding the value zero. If another processor

tries to take the semaphore a zero is returned and the processor now knows that it does not access to the resource and have to try again, spinning in a loop that way. This is the major difference between the queue and busy-wait solution. This does however require the software to support this.
Besides being much simpler the busy-wait solution also have another advantage over the queue. The queue is blocking resulting in that it can not be used with busses. The reason for this is the whole bus would be blocked and a dead lock would occur since the processor holding the semaphore could not communicate with the semaphore unit to release the semaphore and the bus would be blocked until the resource is released. Because of this, and the fact that it is much cheaper, the busy-wait design have been chosen.

4.3 NoC design

Interconnection delay becomes an increasing problem in a time with increasing processing power and multiple cores. Modern SoC with many IP cores sets high requirements for interconnection and busses have become a bottleneck[1]. NoC is an alternative to buses and other communication structure, offering higher bandwidth and frequencies[8]. Busses does not scale well due to physical limitation such as time-of-flight and power consumption driving long wires[1], leading toward segmented communication structures[8]. NoC offers segmented communication and allows for different parts of the chip to run with different clocks known as Globally Asyncronius Locally Syncronius (GALS). The fact that a NoC is segmented, elimination the problem of long wires, with time-of-flight problems, and allows for high frequencies.

4.3.1 Introduction

4.3.1.1 Design

Communication in NoC does not happen directly as in busses or with point to point links, instead it happens in packets. A packet consist of payload, and the header, containing informations about where it is going and other information needed for the packet.

Generally speaking a NoC is built up upon three components 1) Network Adapters(NA) 2) Routing nodes and 3) Links. NA connects the IP Core with the NoC, thereby splitting up computation and communication. This is described in further details in section 4.3.2. Routing nodes, or just nodes, are

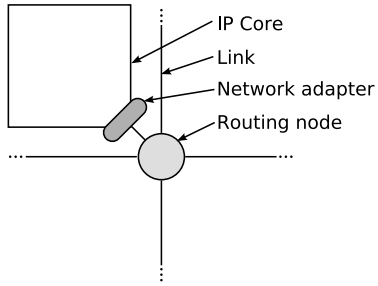

Figure 4.1: Illustration of an IP core, a network adapters, a routing nodes and links.

what directs the packets through the network, see section 4.3.3 for further description. Lastly there are the links, this is what ties the network together, it connects the nodes with other nodes or NAs, these can consist of one or more logical or physical channels[1]. Figure 4.1 shows how these are connected and used in figures.

The way the nodes, links and NAs are put together is called the typology,

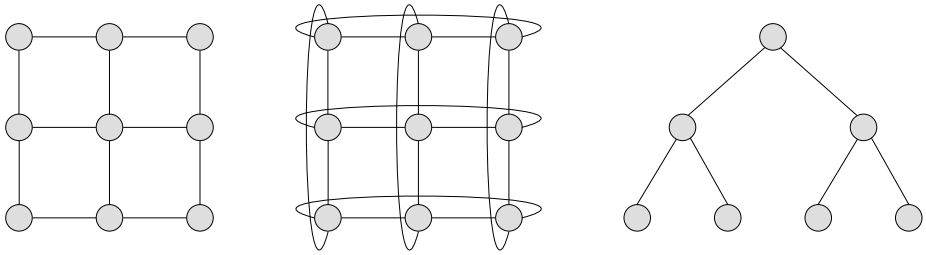

Figure 4.2: Three different of NoC typologies. From the left it is the mesh, torus and last is a binary tree typology.

defining some sort of structure. Many typologies have been designed some of them being the mesh, torus and binary Tree, illustrated in figure 4.2. The mesh is a nice example of a NoC that is possible to lay out on a chip surface, as most typologies are[1]. Each node is connected bidirectionally to its four neighbors, which it can communicate in both directions, additionally it is also connected to a IP core. This of course does not apply for the boundary nodes, which only got two or three neighbors. The shortest path in such a network is $2N$ where N is the size of each dimension.

Alternative to bidirectional connected networks is unidirectional networks, torus is an example of such a typology. That it is unidirectional means that it can only communicate in one direction, this of course does not apply for the connection with the IP core. Torus can also be bidirectional and in that case it resembles

the mesh typology. It has unlike the mesh long distance connections along the boundaries. The result of this is it has longer delays between routing nodes, but in return for this it has shorter paths. Both the mesh and the torus typology is direct networks which means that it has at least one core attached to each node. Tree based networks are typical indirect which means it has nodes that are not connected to any cores.

4.3.1.2 Protocol

The route of a packet can dynamic be decided in the nodes the packet is traveling through, this is called *adaptive routing*, the advantage of this is dynamic load balancing where the packets avoid congestion. The cost of this a more complex node and therefor the alternative *deterministic routing* is more popular. In deterministic routing a packet going from A to B always travels along the same route. The determination of the route can be done in the source, called *source routing*. Alternatively *distributed routing* can be used where the route is determined in the routing nodes. An example of distributed routing is *X-Y routing*, where the packets first follows the rows and then the columns. This requires the node to have the ability to decide which direction to forward the route, this makes the routing nodes more complex. Source routing requires each NA to hold a table, specifying the route to each possible destination in the network, this gives more simple nodes but might give a system that is totally larger than the distrusted routing, depending on the size of the network.
The way packets travels through the network is decided by the networks forwarding strategy, the most common of these is store-and-forward and wormhole. In *store-and-forward* the nodes stores the whole packet before it is forwarded according the the header of the packet. *Wormhole* splits up a packet in flits, the first one containing the header and the following ones containing the payload. As soon as the direction is determined the flits are forwarded, the subsequent flits are then forwarded as they arrive. This allows for a packet to span over several nodes as a worm, hence the name. With this forwarding strategy the buffer size and latency is reduced, the downside of this strategy is a packet spanning over several nodes and is blocked all the nodes in the worm is occupied and thereby blocking other packets, this could also be a problem even if the packet is not blocked.
The routing strategy defines the content of the header, the payload is defined by the data it has to contain. Forwarding strategy defines how wide the links and buffers has to be. Store-and-forward is used as forwarding strategy, which means that the whole packet is sent at a time. To keep the routing node as simple as possible source routing have been used as routing strategy, this means that the only content off the header is the route it has to travel. The route it-

self consist of a predefined number of directions telling the routing nodes which direction to send the packet, the size of the route must be big enough to cover the longest path in the network. How the routing node interprets the direction is described in section 4.3.3.2. For the return route it is required that this is determined along the route, this is described further in section 4.3.3.3. The payload of the packet contains WISHBONE signals. The content of the packet It is

Header		Payload								
83-82	81-74	73-42	41	40-9	8	7	6	5	4-1	0
dir	route	adr	ack	dat	cyc	stb	err	rty	sel	we

Table 4.1: The packet in details. The header contains first a direction for the next node, and then the rest of the route. The payload contains WISHBONE signals.

noticeable that the packet only contains one data signal, while the WISHBONE interface has two data signals. This is due to the fact that only one of the is an output port, by only having the needed signals it is possible to save some wiring.

4.3.1.3 Flow control

Deadlocks is an important problem in NoC as it can make the whole network stall and thereby useless. *Deadlocks* are a condition where resources are waiting for each other to be released in a circular chain, see figure 4.3 for example. Different methods of avoiding deadlock can be used. Such as having a flow con-

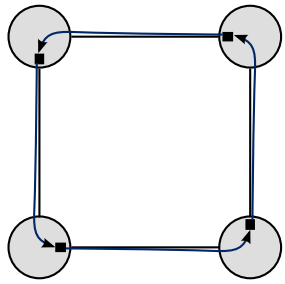

Figure 4.3: A deadlock occur when packets are blocked in a cyclic fashion, thereby the packets are waiting on each other.

trol ensuring no deadlocks occur, by having no circular flow, X-Y routing as an example offering this. Another popular solution is virtual channels(VC), which offers several channels within a physical channels. The idea of VCs is to have

more than one buffer which is independent of each other. Thereby if one buffer is filled by a packet which is blocked, another packet can use the other buffer to pass the node another way than the first one, see figure 4.4. VCs does not come without a cost as both more control and buffer is needed. Besides solving the deadlock problem VCs has some other advantages in term of improved performance, optimized wire utilization and differentiated services[1].

As the design network is rather small, an unconventional solution have been

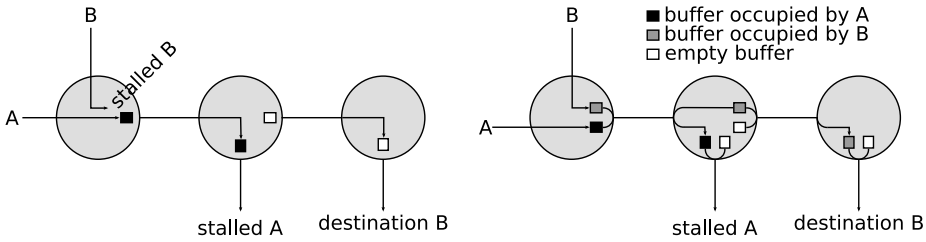

Figure 4.4: In the first figure the packet `B` is blocked in the first node by packet `A` which is blocked in the second node. In the second figure virtual channels are used, this way the packet `B` can go around the packet `A` where is was blocked before.

used instead. The network uses a kind of store-and-forward[1] forwarding strategy, this strategy also has problems with deadlocks, when a buffer is full, but by removing the handshake signals from the network interface, this would remove the possibility of a deadlock, since packet can not be blocked. This however is generally not a good idea, since there is a great risk that a package will be lost lost due to overfilled buffers. It is however not a big risk in this NoC because a minimal number off packets will be sent through the network at a time, with only a max eight masters at a time. This risk can be completely removed by having big enough buffers in the routers. This approach have been chosen because it uses a simplified and thereby smaller router, and also solves the deadlock problem.

4.3.2 Network adapter

The network adapter is the linking element between a IP cores and the network, thereby separating calculations from communication. IP cores communicate with address and address busses along with some control signals, a network communicates with packets. The network adapter wraps the address, data and control signals into packet and unwraps them in the other end. This may be a simple task but it is extremely important. The IP core interface and the services

offered by the network determines how complex this task is. The more the IP cores are aware of the network and build for networks the more simple the task is, and there is a higher potential to make optimal use of the network. If on the other hand the network is design to utilize the exploding resources available, the task is more complex since it has to support different needs.
On the network side the NA connects with a network interface(NI), which defines how communication is done on the network. On the core side the NA connects with a core interface(CI). In section 3.2, the WISHBONE interface, which is the used CI is described. It consist of both a master and a slave interface so also two different network adapters are needed, one for the master interface and one for the slave interface.

4.3.2.1 The master network adapter

The master network interface has to function as a slave for the master IP core. When the master IP core wish to make a request to the slave, the network adapter has to wrap the request into a packet containing the necessary data and the route to the slave, and send off the packet. Details for the packet can be found in section 4.3.1.2.
The NA must only send out the packet for one cycles. This means that can not always send out the packet but only when needed, therefor it has to be able to change the data on the link.
The easiest way to implement the master network adapter is with a FSM, built up as follow:

1. Wait for request from the master.
2. Send the packet to the network.
3. Wait for respond from the slave.
4. Send the respond to the master.

While waiting on the respond from the slave the network adapter check the `ack`-signal, if this is active the NA has received the respond from the slave, this is the job of a *link controller*LC. The code for the master network adapter can be found in appendix A.2.2 on page 74.

4.3.2.2 The slave network adapter

The slave network adapter and master network adapter are fundamentally the same but they differs in some aspects. The slave NA is waiting on the network, where the master network adapter waits on the IP core.
As the packet travels from note to note through the network the the route is updated with a return route, this is described further in section 4.3.3. This means that the slave network adapter does not have to find the route back it self, but can instead use the route given from the packet. The route however is in the reverse order, so the order of the route has to be corrected.
As the master network adapter the slave is also implemented with a FSM, built up as follow:

1. Wait for request packet from network:
2. Send request to slave
3. Wait for respond from slave
4. Send respond packet to the network

The big difference is where the master network adapter only has to handle one request at a time, the slave network adapter could receive another request while it deals with the first. This problem is solved by adding a buffer to the network adapter, just as the router has one. The buffer this is described in details in section 4.3.3.1.
Just as the master NA, the slave NA has a LC to check if there is a packet on the link. If the `stb`-signal is active, and then it has to push the packet into the buffer.
Going into detail with the FSM with the requirements for this is:

1. Wait for a packet to occur in the FIFO. When a packet is in the FIFO go to next state.
2. Set strobe high indicating a request for the slave IP core. Reverse the route. Send out the packet. When respond is ready go to next state.
3. Pop the packet on the FIFO and start all over.

Notice that while waiting on the respond from the slave IP core the packet with the current data is sent out, this does not create a problem with many packet flowing around in the network , because if respond is not ready(`ack` = 0), the packet will be ignored by the routing node. The code for the slave network

adapter can be found in appendix A.2.3 on page 78.

4.3.3 Routing node

The job of the routing node, or node, is to forward the packets through the network. The routing node consist off buffers, a switch, arbitration and routing unit and link controller. The node can have either input or output buffers or both, each with its pros and cons. To connect the input links with the output links a switch is used, see section 4.3.3.3. The arbitration and routing unit, see section 4.3.3.2, is used to decide which way to forward the packet and deciding which packet is to be forwarded, in case off multiple packets wants to go in the same direction. Finally a routing node needs some buffers, section 4.3.3.1, to make sure the packets is not lost in case more signals wants to go in to the same node.
The design of the routing node is illustrated in figure 4.5. As shown in the figure, the switch is designed with five input links and five output links, named `north`, `south`, `east`, `west` and `ip` for easy recognizability.
. The design contains a LC, the job of this is to control the buffer connected

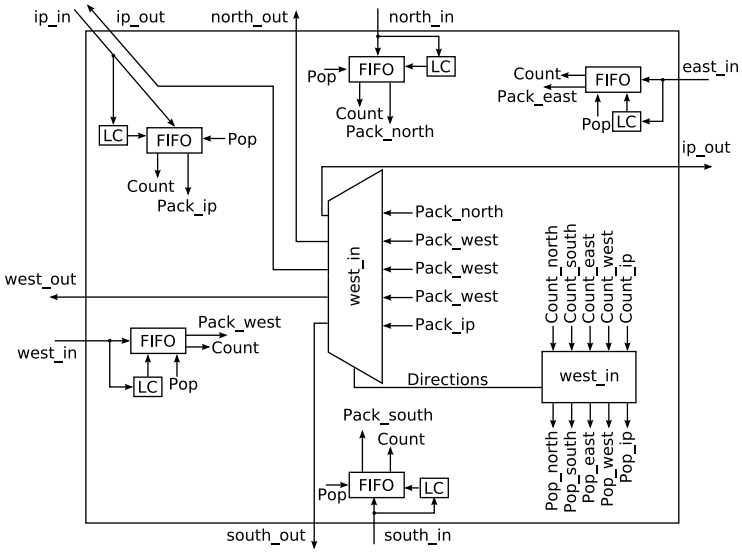

Figure 4.5: Illustration of a routing node with five bidirectional links. The control and status signals are connected to the arbiter. The packet output from the buffers are connected to the switch.

to the link between two nodes. A LC is only assigned to the input links, it check `ack` and `stb` to see if a valid packet is received, in which case the packet is pushed to the buffer. To make this work the sending node must always assure that the `ack` and `stb` in the link is low while it is not sending packets, showing that there is no valid packet on the link. The code for the LC can be found within the source code for the routing node in appendix A.2.4 on page 83.

4.3.3.1 Buffer

The buffer is in the node so it can store a whole packet, or flit depending on forwarding strategy, before it is forwarded. On top off this buffers which is deeper than one, may store more than one packet and thereby lower the possibility of a packet getting blocked. Buffers has some means off telling if it is empty, holding no packets, or full and thereby not being able to receive further packets before some is removed.
The buffer used in this project is a FIFO, which ensure the first packet coming in is the first packet getting out, thereby ensured that packets are served in the order they arrive. To control the status of the FIFO it has a `top` and a `bottom` pointer along with a `count` status signal. `top` and `bottom` are internal signals controlling where to respectively `push` and `pop` to or from. `count` indicate how full the FIFO is. The actual FIFO is constructed with a synchronous push and a asynchronous pop. The source code for the FIFO is found in appendix A.2.1 on page 71.

4.3.3.2 Arbiter

The job of an arbiter and routing unit is to dictate where to forward each incoming packet, implementing the routing algorithm. In a system with distributed routing the arbiter may either calculate the route or look it up in a table, depending on the routing algorithm. In section 4.3.1 it is however stated that source routing is used, in this case the packet itself contains the route and therefor no routing is needed, leaving only arbitration left.
To find out which way to forward the packet, the arbiter just has to look at the two most significant bits of the packet, to decide which way the packet goes. Removing the possibility that the the packet can go back the way it came from, leave four possible output links for the packet to go. This can be decided by a two bit signal. The `north`, `south`, `east` and `west` output links each got a constant route-value assigned to it, the packet is then forwarded according to this. In case the direction is the same as where the packet came from, it should be forwarded to the `ip` link.

In case that more than one packet has to go in the same direction, the arbiter has to decide which one, is sent through first. This is the main job for the arbiter. Which packet to forward is decided with a round robin scheduling, keeping the possibility of deadlocks as low as possible. Further more this ensures a fair distribution of the IP cores packets. When a packet is forwarded, it has to be popped from the FIFO, to leave room for new packets and ensure that the same packet is not sent more than once. Source code for the arbiter is found in appendix A.2.5 on page 93.

4.3.3.3 Switch

The switch connect the input with the output, being a buffer or a link. It makes it possible for an output link, or buffer, to be connected to any input link, or buffer, just like a mux. Depending on the switch it might be able to connect only one output at a time, in a kind of bus way. Or it may connect all outputs with any input at any time, like a crossbar.
The designed routing node has five bidirectional links. Given that the crossbar connection is used, the switch has to be able to connect all these five outputs with the five outputs. The switch has to support for connecting all the output links at the same time, so it would kind of consist of five 5-to-1 multiplexers one for each output. Each mux is controlled by controlling signals from the arbiter. The output from a multiplexer is the chosen packet. However the route needs to be updated with the return route. This is done by shifting the packet two to the left and inserting the return direction in the least significant bits. See figure 4.6 for illustration. This way the direction for the next switch lies as the most significant bits and the direction from which it came is put in as the least significant bits of the route,

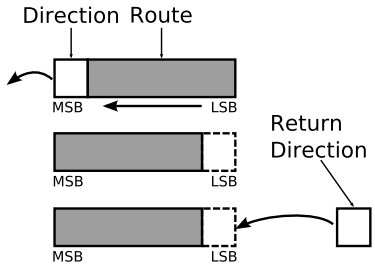

Figure 4.6: Having the current route it is updated by 1. Shift the route n-bits to the left, where n is the width of a direction. 2. Inserting the direction from which the packet came as the least significant bits.

Chapter 5

System design and verification

5.1 Overview of verification strategy

There have not been built a testbench for each component in the system, verifying that it works. The strategy for verification have however been a full system test that verifies that it works in the given environment. Putting the whole system together in one step will however not be a good idea, so stepwise one IP core have been added at a time to verify that it works as intended.
The whole system test is performed, with a program hardcoded into the memory, see section 4.1 for description of this. The idea of the program is to write a string to the UART, if the correct string is printed the system is considered working. This process have also been done in steps. First a simulation of the system have been done, the system is considered working when the transmitter FIFO in the UART gets the correct values. If this have been proven working it is taken to the next step, a synthesis have been done and a simulation of a "'Post-Synthesis Simulation Model"', again the transmitter FIFO has to get the correct values. Because this simulation is equivalent with putting the design down on a board and running it, a working system in this test is considered to be a working system, without any necessary further test. The last step would be to get the system down on the board and running it.

5.2 Single processor

5.2.1 Verification program for single processor design

The OR1k project, see section 3.3, comes with a C test program that print out the string "'Hello world"' on the UART. It consists of a reset assembler file that is executed when reset is activated, it sets up a stack pointer and then jumps to the start of the program. It also contains a linker file placing the instructions at the right addresses in the memory. In addition some header files setting up the program to work with the hardware, are also needed. The most important here is **board.h**, in which it is crucial to set up the correct clock speed, further more the stack size and baud rate also is set in this header file.

5.2.2 Processor and memory

The first implemented design is two memory modules connected to a CPU, one for the instruction and one for the data. Figure 2.1 on page 6 illustrates this. The figure only shows one memory module and connection, this is to keep it simple. By having two memory modules instead of one, it is kept as simple as possible since no switching or other means of connecting both both data and instruction interface to a single memory is needed. Given that the processor is the OR1200 and considered working as it should, the goal of this step is to construct the memory IP core, read about this in section 4.1. This has only been verified by simulation and Post-Synthesis simulation, given that a test on the board will not show anything. It was working as intended, the processor ran through the instructions and the memory delivered the corrected instructions pursuant to the WISHBONE interface.

5.2.3 Processor, memory and UART

This step is split into two. At first design is as shown in figure 2.2 on page 6, the figure though only illustrates the data connection, since this is the important connection. The instruction connection is a direct connection as shown in figure 2.1 on page 6. The difference between this step and the previous one is that a UART is added on the data interface. For the data interface to be able to communicate with both the UART and the data memory a switch, choosing between the two, is included in the top level design, shown in appendix A.3.1. The purpose of this step, is to get confidential with the UART and how it works.

This has been verified first by simulation, next by Post-Synthesis simulation and finally by on board test. It was found to work in all steps.
In the next step, to get familiar with the `conbus`, the `conbus` has been inserted instead of the switch, further more the instruction connection also goes through the the `conbus`. As mentioned in section 3.5 the correct address mapping has to be specified. Also it was discovered that the `conbus` does not fully support the WISHBONE interface. As described in section 3.2 the slave must always qualify the `ACK_O`, `ERR_O` or `RTY_O:DAT_O()`, the `conbus` does not do this. Therefor a WISHBONE qualifier have been included in the top level design shown in appendix A.4.2, the three of the processors and the UART however shall be commented out. This design has been verified with the first two tests, the reason for not doing the last test is that the `conbus` was taken from the OpenCores.org community and therefor already have been tested on a board.

5.3 Multiprocessor

5.3.1 Verification program for multi processor design

The same program for testing the single processor system can obviously not be used for testing a multi processor system for several reasons. First off all it need some sort of concurrency. Further more some sort of synchronization is need, in section 4.2 it is described that a hardware semaphore is used for this. Some means of interacting with this is however needed, for this a header file is used, see appendix B.2.1 for this header file. It defines where in the address space the semaphore is located and means of defining semaphores. It also contains functions for `P` and `V` operation, with respect to section 4.2 the `P` operation is a busy-wait, this is done with a while-loop. The `V` operation is a write with a negative value.
The processors in the multiprocessor system obviously also need to have unique stacks, this is set in the reset code found in appendix B.2.2. The way this is done is by using a semaphore to make sure only one processor is setting its stack at a time. An offset is added to the default stack pointer variable to get a unique stack pointer, the new offset is calculated and stored before the semaphore is passed on to the the next processor.
Then it comes to the actual program, since there is no operating system handling virtual memory, threads or other measures of splitting up programs, the program itself has to handle this. It can be seen in appendix B.2.3, and as one might notice it builds on the hello-UART test program for testing single processor designs. As stated before it is necessary to keep the processors from running the same code, else all processors will perform the same task with the same variables.

First of all it is not possible to predict the outcome of this, as described in section 1.1, and secondly the calculations will not be any faster. To make sure this does not happen a semaphore and a variable is used. The semaphore is used to make sure only one processor is accessing the variable at the time and the variable is used with a case sentence to make the processors run different jobs. Each job is to sends the string "'Hello"' + a unique number on the UART. To make sure that these strings are not mixed together a semaphore is also used here.

5.3.2 Preparing for multiprocessor with Conbus

Almost everything is ready for the multiprocessor design. However a whole new test program is created and a semaphore unit, described in section 4.2, is designed, and these needs to be verified if they work as intended. In its nature this cannot be tested individually and therefore this step is all about verify that these work. The design is illustrated at figure 2.4 on page 7, the only difference from the figure is that only a single processor is used. It was verified to work with simulation and Post-Synthesis simulation, though it is not tested to a full since only a single processor can query it. The source code for this design is found in appendix A.4.2 on page 154.

5.3.3 Conbus and multiple processors

This is the first step with multiple processors, it is an expanded from the design described in section 5.3.2, in that it contains multiple processors. Every thing else was tested to work so the only problem in this part could be that something in the semaphore was not working correct, even though it was tested in previous step, or a problem in the system with adding another processor. The goal of this step of course is to have a full working multi processor system, which is illustrated in figure 2.4 on page 7. As shown of the figure it is possible to have up to four processors and have been tested with both two, three and four processors. All three tests was done with two and three processors, but the board test was not done with four processors because the design was to big to fit the board. The source code for this design can be found in section A.4.2.

5.3.4 Multiprocessor with NoC

This is the last and final goal for the project, to design a fully working multiprocessor with NoC as illustrated in figure 2.5 on page 8. Two NoCs were designed,

both inspirited by the binary tree design. The tree design was chosen because it is simple, easy to implement and keep perspective when testing, the downside off this design is that it is not very efficient and has a significant bottleneck where everything gathers in the middle.

5.3.4.1 The tree NoC

The first NoC design is an ordinary tree design illustrated in figure 5.1. The de-

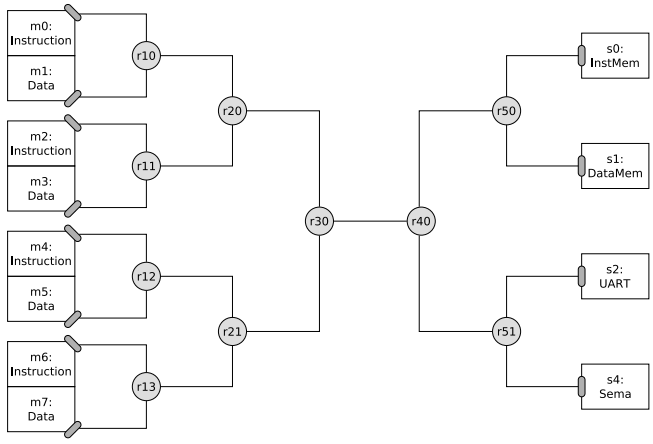

Figure 5.1: The `tree` typology.

sign contains 8 master NA, named `m0` to `m7`, the same way the slaves are named `s0` to `s3`. The routing nodes are name `rXY`, where `X` is the number counted from the left, starting with 1 and `Y` is the number counted from the top starting with 0. So the node connected with `m0` would be named `r10` likewise the node connected to `m3` is named `r12`. The node connected with both `r10` and `r12` are named `r20`.
The design files are found in appendix A.5 on page 193. The source code for the NoC typology, is found in appendix A.5.1 on page 193, and the top level design is in appendix A.5.4 on page 225.
The design itself is not very efficient when it comes to using the designed NoC components described in section 4.3 and it actually much larger than the conbus and could not fit the board. As this might indicate it was only tested to work with a simulation and a Post-Synthesis simulation.

The illustrated design has four processors, giving eight masters, but it would possible to add more processors. There is however a big problem adding more

processors the network gets slower, the same problem occurs with adding more slaves. The problem is that the more master or slaves in the design the longer the path is, and thereby the slower the communication is. More IP cores will also make the problem with the bottleneck, which occur between `r30` and `r40`, even bigger.
Considering the nodes through the network as pipelines, see figure 5.2, makes it possible to has a lot of packets going through the network. But also here the

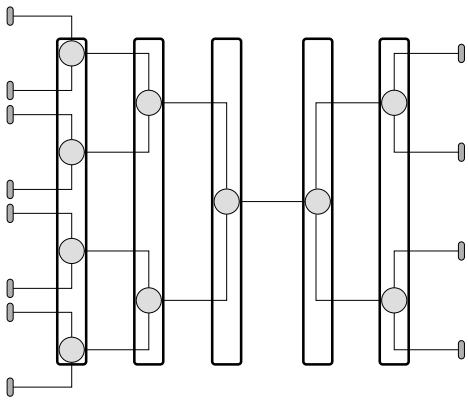

Figure 5.2: The nodes `tree` could be considered as a pipeline. The same is valid for the `stree`.

system lacks efficiency, since only one packet is sent from a master at a time, resulting in a lot of the links being idle the most of the time. Having 21 links a minimum of 13 links is idle all the time. Also only three out of the five ports in the nodes are used , making 2/5 of every node idle all the time.
When it comes to speed the `tree` structure also has problems, table C.1 on page 302 shows the flow of packets when all masters sends out a packet at the same time. This show that in the best case, where the packet is not effected by packet contention, it will take seven cycles for a packet to get from a master to a slave network adapter, or the other way around.
But the tree structure have not been chosen because of its speed nor it efficiency, it has been chosen, as described before because it is easy to design and keep perspective in. No matter from which master a packet is sent the same route is used the get to a given slave, and the return route is auto calculated. Because of its simplicity it is also easy to debug and follow packets in the network, and make sure everything works as intended.

5.3.4.2 `stree` NoC

Since the tree design was not able to fit on the FPGA another design was made with inspiration from the tree design. It makes better use of the switches and therefor has less signals and router resulting in a smaller design.
Taking a look into the data flow it takes minimum seven cycles for a packet to go all the way through the network. Just looking at the single packet, only one step in the pipeline, see figure 5.2, is active, the rest is idle. The processor can not do anything further while the packet travels through the network to the slave and back again and the respond is received. This takes only a single cycle at best, but in the network it takes 14 cycles before the master has the requested data, and thats at best. At worst it takes it takes 28 cycles, based on the data in table C.1 on page 302.
So how is this done better? The fact that it minimum takes seven cycles for a packet to or from a to b is not necessarily a bad thing, if for an example these seven NoC cycles would only take the same physical time as a single outside processor/memory/UART cycle. So a GALS would speed it up, but it requires the NoC to run seven times as fast to do this.
It does however requires some modifications off the entities in the design to work, another solution would be to make the path shorter. Looking into a single flow, from a master makes a request, until it receives the respond. While the packet travels in the network, the master, the slave, 3 routers and 5 links are idle. Of course some of these will be busy with other packet, but as described in section 5.3.4.1, at least 61% of the links are idle all the time. If some of them could be removed the path would be shorter, and the NoC thereby faster.
Looking the nodes, they are designed with possibility to be connected with five other nodes, but each of them in the `tree` is only connected with three other. This gives the possibility to join some of the router and thereby removing some of the links, on top of this it will also make the NoC design smaller. Splitting the design up between `r30` and `r40` gives a master side on the left and a slave side on the right. Looking at the master side, as a binary tree, with `r30` as the root, `r20` has two children each having two children, setting these four grandchildren as children, one link is removed from the path and even though the binary tree structure is removed it still has a tree structure. The same procedure can be done on `r21`. At the slave side the same procedure can be used once again, leaving only a single route.
The final design is illustrated in figure 5.3, as seen on the figure it has four switches and three of them uses all five ports, thereby making a decent use of the switches. The design files are found in appendix A.6 on page 257. The design itself, describing the NoC is found in appendix A.6.1 on page 257 and the table for calculating the route is found in appendix A.6.2 on page 279.

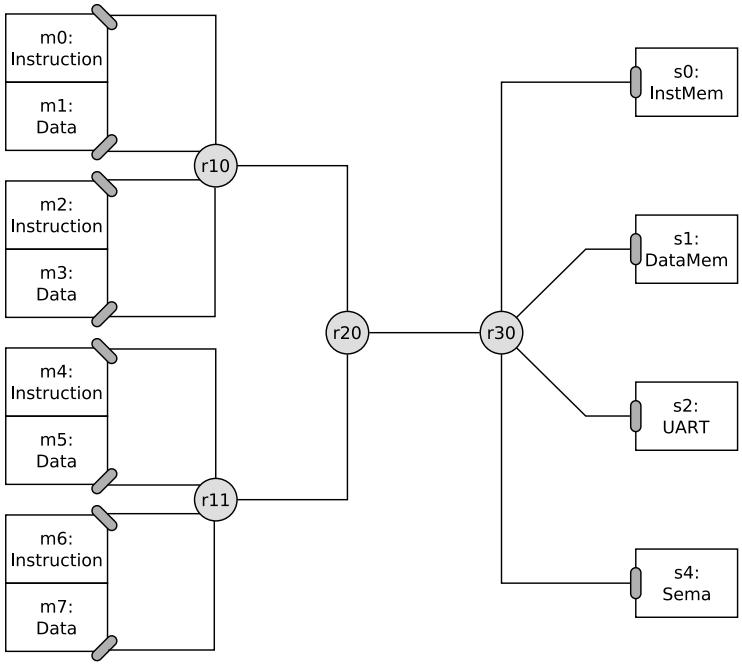

Figure 5.3: The `stree` typology.

5.3.4.3 Buffer size

In section 4.3.1.3 it was described that handshaking was removed to take care of the deadlock problem. Handshaking would have made sure that the receiving node is ready to handle the packet. When removing handshaking the sending node just sends the packet, not caring if the receiving node is ready or not. This is raises the possibility that the buffer is full result in a loss of a packet. There are two possibilities to handle this new problem. One of them is to have a packet loss detection unit, which would make sure the packet would be sent out again if it is lost. This however has a high overhead and is generally not considered a good solution[1]. Alternative it could be assured that the buffers are so large that a packet will never be lost. In large systems this would mean very large buffers, and would not be worth it. But the systems in this project is not large and the buffers would have a moderate size. So how big should the buffers be?
The packets in the network can be split in two types, those coming from the master IP cores (requests) and those coming from the slave IP cores (responds). Request can only go in one direction, from the master interface to the slave interface, making one subnetwork, as shown on figure 5.4. In section 4.3.3.2 it

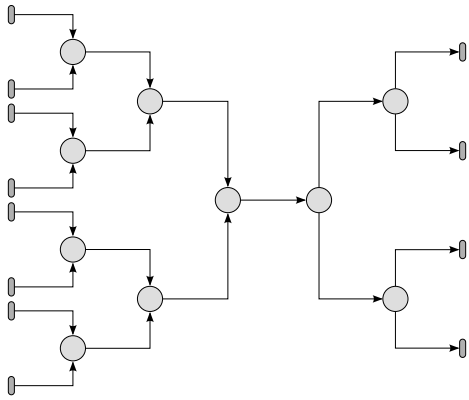

Figure 5.4: Packets from the master NA can only travel in one direction. This creates a unanimous

was described that it is not possible for a package to go back the way it came, ensuring that this data flow is withheld. The opposite subnetwork is available for the responds, thereby ensuring that a respond from a slave is not blocked by requests and vice versa.
As described before removing handshaking requires that the buffers are big enough to ensure no package is lost. Considering the designed tree structure it has a maximum of eight master IP cores, since each master interface only

is capable of sending one packet at a time, there can be no more than eight packages in the request-subnetwork at a time. Additionally a slave can not send a respond packet without removing a request packet, result in a maximum of eight packets can be present on the entire network at a time. The safe thing to do is using buffers with a depth of eight, one for each possible packet.
Table C.1 on page 302 shown the data flow if all eight master sends out a packet at the same time. From the table it is shown that the router holding the most packages at one time is router `r30`, this is the bottleneck, holding 5 packets in cycle 7. Looking a bit more into the table shows that they come from two different directions thereby going into two different buffers, two in one of them and three in another. This indicates that a buffer size of four is sufficient.
The table however only looks at one series of packet, it might be possible that the response from the first packet would return and the master then send out a new packet arriving to `r30` before the last packet has left? All this would at minimum take the three cycles for the packet to arrive at the slave, another seven cycles for the packet to return and if it is then assumed that the master sends out the next request the following cycle, it would take yet another four cycles for it to arrive at `r30`, making a total of 14 cycles. The last packet leaves `r30` after 11 cycles, seven cycles after after the first packet. So there is lots of time from the last packet leaves to the next packet could arrive. This means that a buffer in the router with depth of four will be sufficient, halving the depth of the buffer. This also applies for the `stree` structure, also having 3 packets in the buffer at most and the last packet leaving before the respond from the first packet arrives.
Another buffer is also present in the network, in the network adapter. Looking

address:	0	1	2	3
cycle 1	p1			
cycle 2	p1	p3		
cycle 3		p3	p5	
cycle 4		p3	p5	p7
cycle 5	p0		p5	p7
cycle 6	p0	p2	p5	p7
cycle 7	p0	p2	p4	p7
cycle 8	p0	p2	p4	p7/p6

Table 5.1: Network adapter buffer with depth four and four processors. Address are horizontally and the cycles vertically, p0 to p7 indicating packages from master m0 to m7. The buffer is of type FIFO as described in section section 4.3.3.1. The odd masters are data interfaces, requesting a store byte and the even(including zero) are instruction interfaces request load words.

back at section 4.1 a store byte instruction took two cycles while all other took only one cycle. The buffer would not have any problems with an endless series

off request taking one cycle to handle, because they are handled just as fast as they are received. However store byte instructions could be a problem, since these are not handled just as fast as they are received leaving a possible package loss if the buffer is not big enough. Having four processors is in the `tree` design, involving 8 master interfaces, four of them however are instruction interfaces only requesting reads, taking a single cycle. Table 5.1 shows the content in such a buffer with depth four. The table shows there is a possible collision in cycle eight, where packet *p7* is overwritten by packet *p6*. It should be kept in mind that this is a a fictive example, even so it could happen and therefor the buffer, in a slave network adapters needs to have depth 8, to make sure this does not happen. Since the NoC designs in section 5.3.4 only uses three processors, table 5.2 shows this setup. This table shows that with tree processors a buffer with

address:	0	1	2	3
cycle 1	p1			
cycle 2	p1	p3		
cycle 3		p3	p5	
cycle 4		p3	p5	p0
cycle 5	p2		p5	p0
cycle 6	p2	p4	p5	p0
cycle 7	p2	p4		p0
cycle 8	p2	p4		
cycle 9		p4		

Table 5.2: Same as table table 5.1 only with three processors, instead of four.

depth for is enough if it takes five or more cycles from the respond leaves the network adapter until the next request is received, witch is the case for both the `tree` and `stree` design. So a buffer depth of four will be sufficient.

Chapter 6

Results and discussion

Chapter 2 describes the systems that was to be designed. These designs have been implemented and components for these designs have been made. The last step was to design a NoC for the system and them implement it. Two NoC typologies have been designed and these are described in section 5.3.4, the `tree` shown in figure 5.1 and `stree` shown in figure 5.3. All these have been designed and data and results collected, these are overviewed in section 6.2. During the process some optimizations of the designed components have been made these are described in section 6.1. In section 6.3 the result from the different systems are discussed and some evaluations are made.

6.1 Optimizations of router node

The final size of the routing node is 1482 slices, some optimizations have however been made to get to this size. The size of the first version of the routing node was 1889 slices, which was founded to be to large. At first optimizations have been made of the switch in the node, the first version of this was built up with a mux for each output port having case-statements.

```
1   ...
2   case(select_north_i) is
3     when source_south =>  --South is source
```

```
4           --Set data
5           north_o(NOC_DATA_WIDTH-route_wdth-1 downto 0)   <=
                south_i(NOC_DATA_WIDTH-route_wdth-1 downto 0);
6           --Update route
7           north_o(NOC_DATA_WIDTH-route_wdth+1 downto NOC_DATA_WIDTH
                -route_wdth)   <=
8             south_i(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2);
9           north_o(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-route_wdth
                +1)      <=
10            south_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-
                  route_wdth-1);
11  ...
```

To reduce the size, the code inside the case-statements was compressed, gathering the calculation of the route into one statement, this gave a small improvement. The next big improvement came when the case statements was taken out of the process and changed to with-statements

```
1   ...
2   WITH select_north_i SELECT
3     north_o(NOC_DATA_WIDTH-route_wdth-1 downto 0) <=
4       south_i(NOC_DATA_WIDTH-route_wdth-1 downto 0) when
            source_south,
5   ...
```

The last change was to produce a complete output package in a single statement, thereby not only reducing the code and the readability of it but also the percentage of the total chip area. The switch was totally reduced with 14% of the original size.
It was however another story with the arbiter because it was quite small in the first place, therefor there have not been made any big improvement.
Since the nodes have four buffers, reducing the entity with a single slice would reduce the design of the routing node with four slices. The first version had a case statement switching on push and pop signal.

```
1   ...
2   case(control) is
3     when "10" => -- Only push
4       mem(top)  <= data_i;
5       if(top = fifo_size-1) then
6         top <= 0;
7       else
8         top <= top+1;
9       end if;
10      counter   <= counter+1;
11    when "01" => -- Only pop
12      ...
```

This was improved with a better way of calculation the bottom and top pointer. At first, this was a really good improvement, but it had a flaw, it did not push

if both push and pop signal were active. After correcting this error, it turned out that there was no improvement at all. It was however not for nothing, since it was now known that having fewer cases writing to the memory, the smaller the buffer was. The case-statement was now rewritten to two if-statement, one for pushing and one for poping, which meant there was only one case where the memory was updated.

```
1   ...
2   if(push = '1') then
3     mem(conv_integer(unsigned(top)))  <= data_i;
4     top   <= unsigned(top) + unsigned(one);
5     if(pop /= '1') then
6       counter   <= unsigned(counter) + unsigned(one);
7     end if;
8   end if;
9   ...
```

This gave a great improvement in both area and speed, filling only 78% of the original design and improved the maximum frequency with 17%. In total the router area was improved by 16%, in the cost of a slightly slower design. But there was more to get, by using the latest version of ISE WebPack from Xilinx[9] the design was improved from 1592 slices to 1482 slices. It was now tried to set the syntesis tool to optimize for other than area, with normal optimization effort for speed, the same number of slices was used, but the maximum frequency was increased from 102 MHz to 173.5 MHz an improvement of 45% of the original design. So the final design of the routing node use 1482 slices and has a maximum frequency of 173.5 MHz.

6.2 Area and performance

A lot of data has been gathered with the systems, such as size and maximum frequency. These data have been gathered by synthesis. When synthesising it is possible to optimization with different goal and effort, to get an all uniform and comparable results, synthesises have been done with **speed as goal** and a **normal optimization effort**, because these gave the best results with the tree structures both concerning speed and area. The synthesis have been made to a Xilinx Virtex4 FGPA model number xc4vlx25-10ff668. It should also be mentioned that the stated maximum frequencies are given by the synthesis tool.

Table 6.1 list the used components and the size of these components in slices. Three different systems with multiprocessors have been designed. They are all very similar, they have three processors, two memory, a UART and a semaphore unit. The differ Also data from the communication designs have been gathered. Table 6.3 shows the size of the communication design along with the maximum

Component	# of slices
OR1200	2605 slices
UART	509 slices
Semaphore	18 slices
Memory wrapper	101 slices
Master NA	69 slices
Slave NA	190 slices
Routing node	1482 slices
- Buffer	171 slices
- Arbiter	93 slices
- Switch	596 slices

Table 6.1: Size of components.

	`conbus`	`tree`	`stree`
Area	8856 slices	12847 slices	10587 slices
Max frequency	81.332 MHz	93.470 MHz	83.076 MHz

Table 6.2: Size and speed of systems.

frequency for the design. It should be noted that the data is acquired by having

	Area	Max frequency
`conbus`	392 slices	303.430MHz
`tree`	7679 slices	102.176MHz
`stree`	5705 slices	107.746MHz

Table 6.3: Size and speed of interconnections.

the component or system as a top level design. The reason is that this was the only way to get the data from the tools. This means that the `tree` NoC does not necessarily fill 7679 slices in the `tree` system, or that the UART fills 509 slices when used in a system. This only tells the size of the component when used without any other components.
All these figure does not tell how fast the system really is, example how fast it can execute a program. This table 6.4 tells, by showing how long time it takes for the system to run the 4coretest program listed in appendix B.2.3 on page 289. These data are gained by simulating with the program modelsim. The clock period lasted 80 ns which is the same as a frequency of 12.5 MHz. In table 6.4 the first row tells how long it took in total, until the last symbol was sent from the UART, including the reset from the test bench. The second how long it took from the first cycle after the last reset. The last row tells how many cycles it took from the last reset.

	conbus	tree	stree
Total time	26,473,955 ns	27,047,715 ns	26,727,715 ns
CPU time	26,472,720 ns	27,046,480 ns	26,726,480 ns
Cycles	330,909 cycles	338,081 cycles	334,081 cycles

Table 6.4: Time to execute the `4coretest` test-program found in appendix B.2.3.

6.3 Comparing interconnections

Looking into the interconnections, the NoC designs and the bus both uses a round robin arbitration, and both supports up to eight master interfaces, but while the bus also supports 8 slave interfaces, the NoC was design for this project and therefor only supports four slaves. In theory the NoC is potentially faster since it is able to handle requests from all eight masters at a time while the bus handles a single request at a time. As a bus would contain an arbiter, an address decoder, some muxes and state information it would seem to be simpler, and thereby smaller than a NoC, containing several of network adapters and routers.

6.3.1 Area

Looking at the size of the components in table 6.1 the most of the components are quite small. It is clear that components without buffers are smaller than components with buffers, even though the task of the component is almost the same. Comparing the master and slave NAs there is not a great difference in what these do, however the slave is 2.75 times bigger, this is primarily due to the buffer.
The processor, OR1200, and the routing node is by far the two largest components. As described in section 6.1 the first version of the routing node filled 1889. Even though the task of the processor is much more complicated than the task of the routing node, the processor is only 1.38 times as big as the first design. This clearly indicates that the design of the node was not optimal and it could be done better. The processor is 1.748 times as big as the final version of the routing node, this is much better than the first version, but indicates that it might be done better.
Looking more into the node design the area is primarily taken up by buffers and the switch. First looking at the buffer it has 86 output bits in data and status signals. The buffer contains

- One 4x84 distributed ram.

- Two 2-bit up accumulator, these are the pointers.
- One 2-bit updown accumulator, this is the counter

Summing up this there are 336 bits ram and a total of 354 bits of data. In details 45 flipflops and 179 LUTs have been used. With this in mind 171 slices is not much, it could however in theory be better given that each slice contains two LUTs.

The UART contains a FIFO which is much the same as the one in the routing

	Width	Depth	Total data	Area
Node:	84 bits	4	336 bits	171 slices
UART:	8 bits	8	64 bits	35 slices

Table 6.5: Detail view of FIFO in the UART and routing node.

node, the details for this is listed in table 6.5. From this table it is shown that the node FIFO contains 5.25 as much data as the one in the UART, but it is only 4.89 times as big. This indicates that improving the buffer in the routing nodes would not be an easy task.

Concerning the switch there is no other component to compare it with. Looking at it instead, it does not contain any data, it is only connecting inputs with outputs. As it has five 84-bit outputs it would at best bee made with 420 LUTs, or 210 slices. This indicates that there is more area to get from the switch. This improvement could possibly be found in the fact that the switch also handles updates of the route. Instead of updating the route of the packet just before it is send out, an improvement would be to update the route before it is given to the switch as an input. This way instead of having five route updaters for each input, 25 in total, there would only be one for each input, 5 in total, that way saving some area.

Looking at figure 5.1 and 5.3, the `stree` structure is clearly smaller containing only four routers and 15 links, compared to the `tree` having 10 routers and 21 links. This shows it also makes better use of five ports in the router, the `tree` having 2.1 links per router, the `stree` network has 3.75 links per router. But what really matters is how much it fills on the board, this is listed in table 6.2. From the figures it is shown that the `stree` is 2260 slices smaller, which is quite an improvement. Since the only difference between the two designs is in the communication this is where the improvement have been done. Looking at the size of the NoC itself listed in table 6.3 the difference is 1974 slices which is almost the same as the difference between the entire systems.

Comparing with the `conbus` the two NoC systems are much larger, which is quite intuitive since they are more complex. Just comparing a single node in the NoC with the bus, and the bus is still smaller. Looking at what they do the reason for this is clear. The function of the two is almost the same, the

bus connects 8 masters with 8 slaves the routing node connects 5 link to each another. But while the bus only connects a single master and slave at a time the node connect all the five links in a crossbar fashion. Looking at table 6.1 and 6.3, the switch which is responsible for this is also larger than the whole `conbus` interconnection. The interconnection in the NoC systems are also the largest part these.
Finally it is noticeable how small the `tree` system, table 6.2, is compared the what it contains in components including the interconnection in table 6.3. Actually summing up the total area of the components and interconnection in the `tree` system they use 16223 slices. Even more noticeable is it when summing up the area of the ten routers, eight master NA and four slave NA, which is what the `tree` interconnection uses. They use a total of 16132 slices, while the `tree` interconnection is only listed to use 7679 slices.
This is a big improvement for a synthesis. The reason for these improvement should be found in the fact, that when synthesising a single component, this component have a series of in- and outputs that are unconnected. This means that the synthesis tool con not find out what values they might contain and therefor can not optimize them.
Connection this component to another component some of these, or maybe all, of these I/O ports are connected with the new component. The synthesis tool can now follow the whole path of a signal and thereby see the use of this. It might be that it is never used, and therefor can be removed, or that it will always have the same value as another signal and therefor these are joined. It could also just be that some of the bits in the signal is never used and therefor these are removed. Looking at many components one at a time there will be many unconnected I/O ports and therefor connecting all these will give a big improvement.

6.3.2 Performance

With multiple processors a lot off communication will be generated and there is a big possibility that the communication will slow down the system. The `stree` was designed to be better than the `tree`, in section 6.3.1 it was shown that it is smaller now we will focus on the performance, besides comparing the two NoC systems also the bus system will be looked upon. Taking a look at the imaginary case that each master interface send out a request to the same slave IP core at the same time. Table C.1 shows how the packets will flow in the `tree` network, while table C.2 shows the flow for the `stree` network. Table 6.6 sums up these two tables along with the flow from the `conbus`. Looking at the `tree` and `stree` it is shown that the packet arrives two cycles before with the `stree` design. This is not the whole story since it is unknown if the average packet would spend longer time in the network, this could be due to a higher degree

packet #	`tree`	`stree`	`conbus`
1	7 cycles	5 cycles	1 cycles
2	11 cycles	7 cycles	2 cycles
3	9 cycles	9 cycles	3 cycles
4	13 cycles	11 cycles	4 cycles
5	8 cycles	6 cycles	5 cycles
6	12 cycles	8 cycles	6 cycles
7	10 cycles	10 cycles	7 cycles
8	14 cycles	12 cycles	8 cycles

Table 6.6: Shows how many cycles it takes for each packet to arrive to its destination, when all master interfaces send out a request to the same slave at the same time. Data are gathered from table C.1 and C.2.

of traffic contention. It would require a lot more testing and data gathering to find out this. Looking at the `conbus` it is even faster, four cycles faster in every case to be exact. Again this does not tell the whole story.
So according to table6.6 the `bus` design would be faster than any of the two NoC designs and `stree` would be the fastest of these two. But the clock frequency also has a impact on this, and in theory the NoCs would be faster since they has simpler arbitration and shorter wires. The two NoC should be able to run at same speed. Table 6.2 list the maximum frequency of each system, the `conbus` system is not as fast as the two NoC design, but it is surprisingly not that much slower. The reason for this would bee that it is a fairly simple design with not that many connections and the bus therefor do not have big problems with arbitration and driving the bus. Having a longer bus this problems would occur, unless a NoC is used. Also good NoC has a much higher bandwidth than buses, being able to let many masters communicate with different slaves, while only one master can communicate with a slave in a bus. To overcome the problems with the bus, it is often pipelined, resulting in using much more area. This extra area are mainly going to buffers, but extra muxes, arbiters and state information is needed.
Looking at the two NoC systems the `tree` system is faster than the `stree` system according to table 6.2. But looking at table 6.3 and it is shown that the `stree` typology is faster than the `tree` typology. First of all they would be expected run at the same speed, because they are built almost the same way and with the same component. Further more the one is faster in one of the tables and the other is faster in the other table. Since they use the same component and the same IP cores the explanation for this lies within the synthesis and the way the optimization in the synthesis tool are done. This is greatly illustrated in that with as an example the routing node an synthesis with a normal optimization effort for speed gave a faster result than a synthesis with high optimization effort with speed for goal. It also result of a normal optimization for speed also gave

a smaller result than both normal and high optimization effort with low area as goal.

So the `conbus` would be expected to be the fastest of the three, `stree` the

conbus	tree	stree
4,068,526 ns	3,616,790 ns	4.021.333 ns

Table 6.7: The amount of time it takes for the testprogram `4coretest`, found in section B.2.3, to finish with a given system.

second fastest and finally `tree` would be the slowest based on table 6.6. To find out the correctness of this they have been simulated with the test program and the result of this have been listed in table 6.4 on page 43. It clearly shows everything is as expected, but this is at same frequency. When considering figure 6.2, which shows the `tree` is capable off running at higher speeds, it might be another result. The actual program has a low level of concurrency since most of the time it uses shared resources which only one processor may access at a time, so it is limited how much it represent an real life concurrent program. It does however has some level of concurrency and therefor it is not useless. The number off cycles it takes to execute the program for each system is also listed in table 6.4, from this it is possible to calculate how long time it would take if the system ran at its maximum frequency, by calculating the clock period with the formula $t_{clk}(ns) = \frac{1000}{frequency(MHz)}$ this is then multiplied with the number of clock cycles. Table 6.7 shows the result of this. Here, very surprisingly, the `tree` system is the fastest and it is actually 11% faster than the `conbus` system. The `conbus` and the `stree` is almost the same execution time. This clearly tells that the `tree` system benefits from the higher frequency. This is however only one program and other systems might be faster with other programs, depending on varius factors such as level of concurrency, amount of comminucation with memory, I/O devices and so on.

6.4 Getting the performance gain from parallelism

NoC is potentially faster than busses, so why is the designed NoC not always faster?

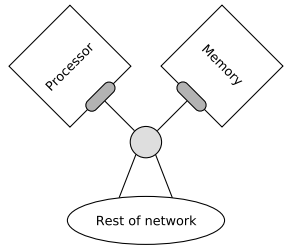

Figure 6.1: The processor makes use of locality when requesting data from the nearby memory.

6.4.1 Hardware

Looking into the design of the `tree` in figure 5.1 and table C.1 it is clear that the link between r30 and r40 is a bottle neck letting only one packet go to the slaves at a time. With this in mind, it is quite intuitive that to be able to fully benefit from parallelism, not only calculation has to be parallel but also communication. This is done by letting the masters have multiple paths to the slaves, making it as unlikely as possible that two packages collide. This is not the case in the `tree` and `stree` networks.
This will not help if there is only one slave in the network, a memory module as an example. It is in parallelism better to have a lot smaller modules than one big, say 8 4kb memory modules in stead of a single 32kb. But this either does not help, if all communication goes to a single of these modules, therefor it also crucial to make sure the communication is spread out on all slaves as much as possible. Having eight master interfaces and only two memory modules this is not perfect in the `tree` and `stree`. Having eight master interfaces and only two memory modules this is not perfect in the `tree` and `stree`.

Another way of improving parallelism is to decrease the amount of communication, this could as an example be done with caches, where the most used data is still present in the processor thereby making it unnecessary to request the data from the memory. This does however also requires that the SoC support some mechanism to control if data is valid. Taking advantage of locality also keeps the time spent on communication at a minimum. This is as an example when it is known that a processor will communicate a lot with another specific IP core. It would then be wise to have a short distance between these two, if on the other hand it was known that two IP cores would not communicate much, or not at all, it is not so important to keep these close, see figure 6.1.
Figure 6.2 is made with locality and parallel communication in mind. Each processor has a small memory module combined with it, designed to hold the stack. This ensure that communication with the stack is as fast as possible, since this is what is going to happen the most. It should also be noted that

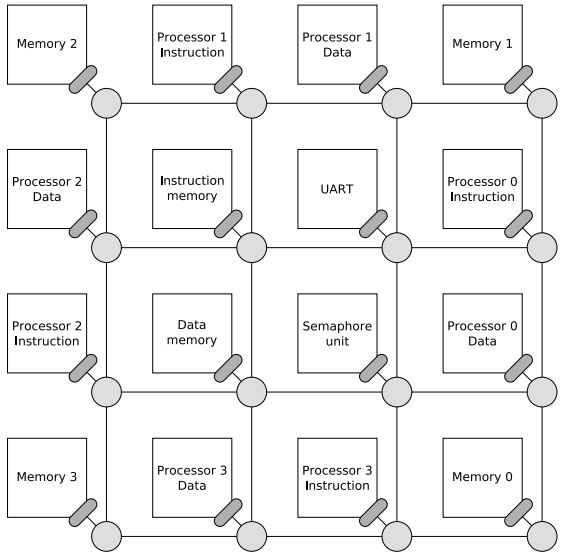

Figure 6.2: A mesh network making use of parallel communication and locality.

these "'private"' memory modules are kept in the corner, as long way away from the other masters as possible because the other masters will not communicate with this memory module. The master interfaces are gathered in the outer ring to make it possible to gather slave devices in the center, because every master module will have to communicate with these, and thereby making the path to slave modules as short as possible. This way parallel communication have been improved and locality have exploit.

6.4.2 Software

But having a system that make use a higher degree of parallelism does not help anything if the program is note made to be parallel. First of all it is a requirement that the given problem can be solved with concurrency, the greatest common divisor algorithms for an example, needs the sub-result before it can start calculating the next. Searching a string, called the `text` for a given substring, called the `word`, is an example off such a problem. It is then needed to split the problem in sub-problems that can be solved independently, in the example with the string this could be to split the `text` into `sub-text`s and then each sub-problem would be to search the `sub-text` for the `word`.
When splitting the problem into subproblems it is important to make sure that

the result is the same. Ensuring this in the word-search example, requires that the word does not occur where the`text` is split, thereby resulting in the `word` would be split and not counted. On the other hand if the `sub-text`s overlap the same occurrence of the `word` might bee counted twice.
Further more it is also required that each subproblem is solved, and the result is gathered. In appendix B.2.4 such a word-search program is made. It should be noted the program is not tested and it might not work fully as intended, but should instead be considered as an example of how to do it.

Chapter 7

Conclusion

7.1 Achivements

The main result for this project is designing and implementing a SoC using IP cores and NoC. This was archived by starting with a simple design and then adding more and more into it, ending up with multiprocessor using a bus just before going to the NoC.
A WISHBONE memory wrapper have been designed to use with the generated memory cores from the xilinx tools. Further more a synchronization unit have been designed based on binary semaphores to keep thread-safety in a multiprocessor environment, this unit supports the WISHBONE interface and requires the used programs to support the unit.
While the res of the peripheral units have been taken from OpenCores, everything within the NoC have been designed from scratch. The NoC uses source routing where the route of the packet is calculated at the source. Further more the store-and-forward strategy have been used, which means that the whole packet is sent at the time instead of splitting up the packet in flits. These choices have been made to keep the NoC simple. A routing node which supports this and uses round robin arbitration have been designed, along with the necessary network adapters adapters.
In the end the network was found to be very large, even after some optimizations. The size of the node, which is the largest component in the network, is

analyzed. Here it is found that the buffer is small even if it might be possible to make it smaller. The switch within the nodes is found to be large and it suggested that taking out route updating from it would reduce the size significant. When it comes to speed the NoC is found to be surprisingly slow, the main reason for this is a bottleneck removing the possibility of concurrent communication. Even so the NoC was also expected to run at much higher speeds than the bus, this is not the case. This is expected to be because the design is rather small and simple and the bus thereby did not suffer from any of its flaws.

7.2 Future work

The project ended with only binary tree inspirited NoCs, but there is many more typologies that could be used, the mesh or torus for naming some of them. Also parallelism could be better exploit, and this should be taken into consideration when designing new typologies. As an example the typology illustrated in figure 6.2 could be a starting point for new typologies.
Also reintroducing handshaking, and thereby make the system more scalable, would be an aspect of future work. An deeper analysis of the designed network to find its weaknesses and design a new network from scratch with this knowledge would be an idea.
If I was to design a new network from scratch i would still use source routing as this seem like an good idea. First of all it keeps the nodes more simple, not having to know where in the network they are, and they do not need an algorithm to find out in which direction to forward a packet. It does however depend on the network typology, as i do believe simple distributed routing, such as X-Y routing would be a better solution than source routing in a mesh network, the drawback of this is it can not be used in every typology.
The chosen forwarding strategy would be wormhole, as this seems like an better solution keeping everything in the nodes smaller. To take care of worms blocking another, VC would be introduced to the system. How many VC need would be up to further analysis, but it will depend on the size of the system. For a system as the one designed two virtual channels would seem fair considering the low amount of packets. Also the design of the routing nodes has to be reconsidered, what to keep and what to do better. Here i specifically think of the switch.
Future work also lies in software development, looking at how to solve problems using concurrency, and making the software run faster with the multiprocessor. This is an interesting aspect to make software for the system, but it could also be the other way around making multiprocessor systems that will solve the problem faster.

Appendix A

HDL files

A.1 Common Files

A.1.1 core_mem.vhd

```
1 ----------------------------------------------------------------
2 --
```

```
3  --       SoC design for multiprocessor
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.1
6  --       Data last modified: 30/3-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.1: Fix problem with ack going high before data was ready
14 --
15 ----------------------------------------------------------------------
16
17 library IEEE;
18 use IEEE.std_logic_1164.ALL;
19 use IEEE.std_logic_arith.ALL;
20 use work.types.all;
21
22 entity core_mem is
23   port (
24     --I/O Ports
25     wb_clk_i    : in bit_t;    -- The Clock
26     wb_rst_i    : in bit_t;    -- The reset signal
27
28     --WB slave i/f
29     wb_data_i   : in word_t;     -- WB data in
30     wb_data_o   : out word_t;    -- WB data out
31     wb_adr_i    : in word_t;     -- WB adress
32     wb_sel_i    : in std_logic_vector(3 downto 0); --WB select input array
33     wb_we_i     : in bit_t;    -- WB write enable
34     wb_cyc_i    : in bit_t;    -- WB cycle input
35     wb_stb_i    : in bit_t;    -- WB strobe input
36     wb_ack_o    : out bit_t;     -- WB acknowledge output
37     --wb_rty_o    : out bit_t;     -- WB retry output
38     wb_err_o    : out bit_t    -- WB error output
```

```
39        );
40
41  end core_mem;
42
43  architecture struc of core_mem is
44
45    component mem_block
46      port(
47        addr     : in std_logic_vector(13 downto 0);    -- Address
48        clk      : in bit_t;    -- Clock
49        din    : in word_t;     -- Data in
50        dout     : out word_t;    -- Data out
51        en     : in bit_t;    -- Enable
52        we     : in bit_t     -- Write enable
53      );
54    end component;
55
56    --
57    --SIGNALS
58    --
59
60    SIGNAL mem_we : bit_t;
61    SIGNAL mem_en : bit_t;
62    SIGNAL mem_din  : word_t;
63    SIGNAL mem_dout : word_t;
64
65    --State Signals and types
66    TYPE state is (STATE_REQUEST, STATE_WAIT, STATE_RESPOND);  -- States for FSM
67    SIGNAL current_state  : state;
68    SIGNAL next_state : state;
69
70
71  begin
72
73    mem_en     <= wb_cyc_i and wb_stb_i;
74    wb_err_o   <= '0';
```

```
75
76
77
78
79    mem : mem_block
80      port map(
81        addr  => wb_adr_i(15 downto 2),
82        clk => wb_clk_i,
83        din => mem_din,
84        dout  => mem_dout,
85        en  => mem_en,
86        we  => mem_we
87      );
88
89
90    -- purpose: Next state and output logic
91    -- type   : combinational
92    -- inputs :
93    next_state_logic : process (current_state, wb_adr_i, wb_stb_i, wb_cyc_i, wb_data_i, wb_we_i, wb_sel_i
          , mem_dout)
94    begin
95        mem_we    <= '0';
96        wb_ack_o  <= '0';
97        mem_din   <= wb_data_i;
98      if wb_stb_i = '1' and wb_cyc_i = '1'then
99        case current_state is
100         when STATE_REQUEST =>
101           wb_ack_o  <= '0';
102           if wb_we_i = '1' then --write
103             -- write word
104             if wb_sel_i = "1111" then
105               next_state  <= STATE_RESPOND;
106               mem_we    <= '1';
107               mem_din   <= wb_data_i;
108             else --write byte
109               next_state  <= STATE_WAIT;
```

```
110               mem_we    <= '0';
111               mem_din   <= wb_data_i;
112             end if;
113
114           else --read
115             next_state  <= STATE_RESPOND;
116             mem_we    <= '0';
117             mem_din   <= wb_data_i;
118           end if;
119         when STATE_WAIT =>
120           next_state  <= STATE_RESPOND;
121           mem_we    <= '1';
122           if wb_sel_i = "0001" then
123             mem_din <= mem_dout(31 downto 8) & wb_data_i(7 downto 0);
124           elsif wb_sel_i = "0010" then
125             mem_din <= mem_dout(31 downto 16) & wb_data_i(15 downto 8) & mem_dout(7 downto 0);
126           elsif wb_sel_i = "0100" then
127             mem_din <= mem_dout(31 downto 24) & wb_data_i(23 downto 16) & mem_dout(15 downto 0);
128           else
129             mem_din <= wb_data_i(31 downto 24) & mem_dout(23 downto 0);
130           end if;
131         when STATE_RESPOND =>
132           next_state  <= STATE_REQUEST;
133           wb_ack_o <= '1';
134         when others =>
135           null;
136       end case;
137     else
138       next_state  <= STATE_REQUEST;
139       wb_ack_o  <= '0';
140     end if;
141   end process next_state_logic;
142
143   -- purpose: State register
144   -- type   : sequential
145   -- inputs : wb_clk_i, wb_rst_i
```

```
146     -- outputs:
147     State_register: process (wb_clk_i, wb_rst_i)
148     begin  -- process State register
149       if wb_rst_i = '0' then                    -- asynchronous reset (active low)
150         current_state <= STATE_REQUEST;
151       elsif wb_clk_i'event and wb_clk_i = '1' then  -- rising clock edge
152         current_state <= next_state;
153       end if;
154     end process State_register;
155     wb_data_o <= mem_dout;
156   end struc;
```

A.1.2 memory.vhd

```
1  library IEEE;
2  use IEEE.std_logic_arith.all;
3  use IEEE.std_logic_1164.all;
4  use ieee.numeric_std.to_unsigned;
5  use STD.textio.all;
6  use WORK.types.all;
7
8  entity memory is
9
10   generic(
11     filename  : string := "";      -- Name of file to load
12     memsize   : natural := 8192;
13     membase   : natural := 0);
14   port (
15     --I/O Ports
16     wb_clk_i  : in bit_t;    -- The Clock
17     wb_rst_i  : in bit_t;    -- The reset signal
18
19     --WB slave i/f
20     wb_data_i : in word_t;     -- WB data in
21     wb_data_o : out word_t;    -- WB data out
22     wb_adr_i  : in word_t;     -- WB adress
23     wb_sel_i  : in std_logic_vector(3 downto 0); --WB select input array
24     wb_we_i   : in bit_t;    -- WB write enable
25     wb_cyc_i  : in bit_t;    -- WB cycle input
26     wb_stb_i  : in bit_t;    -- WB strobe input
27     wb_ack_o  : out bit_t;     -- WB acknowledge output
28     wb_rty_o  : out bit_t;     -- WB retry output
29     wb_err_o  : out bit_t);    -- WB error output
30
31 end memory;
32
33 architecture behavioural of memory is
34
```

```
35    subtype MemAddr is integer range 0 + membase to memsize + membase - 1;
36    type MemType is array(MemAddr) of integer;
37
38    shared variable memory : MemType;
39    shared variable test : Std_logic_vector(31 downto 0);
40
41    procedure loadmemory (constant filename : in string) is -- Load a file into memory
42
43      file memfile    : text open read_mode is filename;
44      variable memline  : line;
45      variable line_no  : natural := 0;
46      variable read_line  : integer;
47      variable datamem  : boolean := false;   -- are we writing data to memory
48      variable str    : string(1 to 5);
49      variable mem_count  : natural := membase; -- number of data/instr blocks put into memory
50
51      procedure read_hex (
52        myline    : inout line;
53        word    : out   integer) is
54
55        variable ch : character;
56        variable digit  : integer;
57        variable part : word_t;
58
59        begin --read_hex
60
61          for i in 0 to 7 loop
62            read(myline, ch);
63            if ch >= '0' and ch    <= '9' then
64              digit := character'pos(ch) - character'pos('0');
65            elsif ch >= 'A' and ch <= 'F' then
66              digit := character'pos(ch) - character'pos('A') + 10;
67            elsif ch >= 'a' and ch <= 'f' then
68              digit := character'pos(ch) - character'pos('a') + 10;
69            else
```

```
70              report "ERROR in input file : " & filename & " on line " & integer'image(line_no) severity
                    error;
71            end if;
72
73            part(31-i*4 downto 28-i*4) := conv_std_logic_vector(digit, 4);
74          end loop;  -- i
75          word := conv_integer(unsigned(part));
76        end read_hex;
77      begin --loadmemory
78
79        while not endfile(memfile) loop
80          readline(memfile, memline);
81          read_hex(memline, read_line);
82          memory(mem_count) := read_line;
83          mem_count := mem_count + 1;
84        end loop;
85        end loadmemory;
86
87  begin  -- behavioral
88
89    -- purpose: write to memory and asynchronous reset
90    -- type   : sequential
91    -- inputs : wb_clk_i, reset, wb_adr_i , wb_data_i
92    -- outputs:
93    MEM: process (wb_clk_i,wb_rst_i)
94      variable mem_vector : std_logic_vector(31 downto 0);
95    begin  -- process memory
96      wb_rty_o <= '0';
97      wb_err_o <= '0';
98      if wb_rst_i = '0' then                -- asynchronous reset (active low)
99        if filename /= "" then
100         loadmemory(filename);
101       else
102         report "No file specified for memory" severity ERROR;
103       end if;
104       wb_ack_o <= '0';
```

```
105       --  conv_std_logic_vector(membase,32);
106       elsif falling_edge(wb_clk_i) then  -- rising clock edge and adr is in memspace
107         if wb_we_i = '1' and wb_stb_i = '1' then --write
108           test := wb_adr_i(addr_wdth-1 downto 2) & "00" ;
109           mem_vector := conv_std_logic_vector(memory(conv_integer(unsigned(wb_adr_i(addr_wdth-1 downto 2)
                  ))), data_wdth);
110           if wb_sel_i(0) = '1' then
111             mem_vector(7 downto 0) := wb_data_i(7 downto 0);
112           end if;
113           if wb_sel_i(1) = '1' then
114             mem_vector(15 downto 8) := wb_data_i(15 downto 8);
115           end if;
116           if wb_sel_i(2) = '1' then
117             mem_vector(23 downto 16) := wb_data_i(23 downto 16);
118           end if;
119           if wb_sel_i(3) = '1' then
120             mem_vector(31 downto 24) := wb_data_i(31 downto 24);
121           --end if;
122           --else
123           --  mem_vector := wb_data_i;
124           end if;
125           memory(conv_integer(unsigned(wb_adr_i(addr_wdth-1 downto 2))))
126             := conv_integer(unsigned(mem_vector));
127           wb_ack_o <= '1';
128         --end if;
129         elsif wb_we_i = '0' and wb_stb_i = '1' then -- read
130           mem_vector := conv_std_logic_vector(memory(conv_integer(unsigned(wb_adr_i(addr_wdth-1 downto 2)
                  ))), data_wdth);
131           if wb_sel_i(0) = '1' then
132           --  wb_data_o <= X"000000" &  mem_vector(7 downto 0);
133             wb_data_o(7 downto 0) <= mem_vector(7 downto 0);
134           else
135             wb_data_o(7 downto 0) <= X"00";
136           end if;
137           if wb_sel_i(1) = '1' then
138             wb_data_o(15 downto 8 ) <= mem_vector(15 downto 8);
```

```
139       else
140         wb_data_o(15 downto 8) <= X"00";
141       end if;
142       if wb_sel_i(2) = '1' then
143         wb_data_o(23 downto 16) <= mem_vector(23 downto 16);
144       else
145         wb_data_o(23 downto 16) <= X"00";
146       end if;
147       if wb_sel_i(3) = '1' then
148         wb_data_o(31 downto 24) <= mem_vector(31 downto 24);
149       else
150         wb_data_o(31 downto 24) <= X"00";
151       end if;
152       wb_ack_o <= '1';
153     else
154       wb_ack_o <= '0';
155     end if;
156
157     end if;
158   end process MEM;
159 end behavioural;
```

A.1.3 types.vhd

```
1  -- Constant and types
2
3  library IEEE;
4  use IEEE.std_logic_1164.all;
5  use IEEE.std_logic_arith.ALL;
6
7  package types is
8
9    -- General Constants
10   constant RESET          : std_logic := '0'; -- Reset active low
11   constant addr_wdth      : integer := 32;  -- Width of address
12   constant data_wdth      : integer := 32;  -- Width of data
13   constant WORD_ZERO      : std_logic_vector(data_wdth-1 downto 0) := (others => '0');  -- Zero word
14
15   -- NoC constants
16
17   constant dest_wdth    : integer := 2;
18   constant route_wdth   : integer := dest_wdth*5; -- width of route
19
20   subtype dest is std_logic_vector(dest_wdth-1 downto 0);
21   constant dest_north : dest  := CONV_STD_LOGIC_VECTOR(0, dest_wdth); --North
22   constant dest_south : dest  := CONV_STD_LOGIC_VECTOR(1, dest_wdth); --South
23   constant dest_east  : dest  := CONV_STD_LOGIC_VECTOR(2, dest_wdth); --East
24   constant dest_west  : dest  := CONV_STD_LOGIC_VECTOR(3, dest_wdth); --West
25
26   subtype source is std_logic_vector(2 downto 0);
27   constant source_north : source := "000";  --North
28   constant source_south : source := "001";  --South
29   constant source_east  : source := "010";  --East
30   constant source_west  : source := "011";  --West
31   constant source_ip  : source := "100";  --IP core
32   constant source_none  : source := "101";  --None
33
34   constant NOC_DATA_WIDTH : integer :=
```

```
35      route_wdth + addr_wdth + data_wdth + 6 + 4;
36    --route, adress, data_in, data_out and 6 single bit signals and the select signal
37
38    --Noc bits
39    constant we_start : integer := 0;
40    constant sel_start  : integer := 1;
41    constant rty_start  : integer := 5;
42    constant err_start  : integer := 6;
43    constant stb_start  : integer := 7;
44    constant cyc_start  : integer := 8;
45    constant adr_start  : integer := 9;
46    constant ack_start  : integer := 41;
47    constant dat_start  : integer := 42;
48
49    -- Types
50    subtype bit_t  is std_logic;
51    subtype byte_t is std_logic_vector(7 downto 0);
52    subtype halfword_t is std_logic_vector(data_wdth/2 downto 0);
53    subtype word_t is std_logic_vector(data_wdth-1 downto 0);
54    subtype noc_data is std_logic_vector(NOC_DATA_WIDTH-1 downto 0);
55
56    type vector_file is file of std_logic_vector(31 downto 0);
57
58    --Fifo types
59    constant fifo_size  : integer := 4;
60    subtype fifo_vector is std_logic_vector(1 downto 0);
61    subtype FifoAddr is integer range 0 to fifo_size-1; -- FIFO size 4
62    --subtype FifoAddr is std_logic_vector(3 downto 0); -- FIFO size 8
63    type FifoData is array(FifoAddr) of noc_data; -- FIFO mem
64
65  end types;
66
67  package body types is
68
69  end types;
```

A.1.4 semaphore.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Semaphore design for multiprocessor
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 25/4-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity semaphore is
22   generic (
23     size      : integer := 8
24   );
25   port (
26   --I/O Ports
27     wb_clk_i    : in bit_t;   -- The Clock
28     wb_rst_i    : in bit_t;   -- The reset signal
29
30     --WB slave i/f
31     wb_data_i   : in word_t;    -- WB data in
32     wb_data_o   : out word_t;   -- WB data out
33     wb_adr_i    : in word_t;    -- WB adress
34     wb_sel_i    : in std_logic_vector(3 downto 0); --WB select input array
```

```
35      wb_we_i     : in bit_t;    -- WB write enable
36      wb_cyc_i    : in bit_t;    -- WB cycle input
37      wb_stb_i    : in bit_t;    -- WB strobe input
38      wb_ack_o    : out bit_t;     -- WB acknowledge output
39      wb_err_o    : out bit_t    -- WB error output
40        );
41
42  end semaphore;
43
44  architecture struc of semaphore is
45
46    --Semaphores
47    SIGNAL semaphores : std_logic_vector(size-1 downto 0);
48    SIGNAL sema_update  : std_logic_vector(size-1 downto 0);
49
50    --active semaphore
51    SIGNAL data   : bit_t;
52    --zero value
53    SIGNAL sema_zero  : bit_t;
54
55  begin
56
57    wb_err_o  <= '0';
58    sema_zero <= '0';
59
60    calculate : process(wb_we_i, wb_data_i, wb_stb_i, wb_cyc_i, semaphores, data)
61    begin
62        wb_data_o <= (OTHERS => '0');
63        wb_data_o(0)  <= data;
64      if wb_stb_i = '1' and wb_cyc_i = '1'then
65        data    <= semaphores(conv_integer(unsigned(wb_adr_i(18 downto 2))));
66
67
68        if wb_we_i = '1' then -- Release semaphore
69          sema_update <= semaphores;
70            sema_update(conv_integer(unsigned(wb_adr_i(18 downto 2))))  <=
```

```
71              '1';
72            wb_ack_o  <= '1';
73
74        else  -- Take semaphore
75          if data = sema_zero then      -- Semaphore not free
76            sema_update <= semaphores;
77            wb_ack_o  <= '1';
78          else            -- Semaphore free
79              sema_update <= semaphores;
80              sema_update( conv_integer(unsigned(wb_adr_i(18 downto 2)))) <=
81                '0';
82              wb_ack_o  <= '1';
83
84          end if;
85        end if;
86      else
87        wb_ack_o  <= '0';
88        data    <= sema_zero;
89        sema_update <= semaphores;
90      end if;
91    end process calculate;
92
93
94    -- purpose: State register
95    -- type   : sequential
96    -- inputs : wb_clk_i, wb_rst_i
97    -- outputs:
98    State_register: process (wb_clk_i, wb_rst_i)
99    begin  -- process State register
100     if wb_rst_i = '0' then                    -- asynchronous reset (active low)
101       semaphores  <= (OTHERS => '1');
102     elsif wb_clk_i'event and wb_clk_i = '1' then  -- rising clock edge
103       semaphores <= sema_update;
104     end if;
105   end process State_register;
106
```

```
107
108  end struc;
```

A.2 Common NoC files

A.2.1 noc_fifo.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Fifo for NOC
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.3
6  --       Data last modified: 24/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.1: Changed to use case instead if if/elsif
14 --       v1.2: Primary uses std_logic now instead of integers
15 --       v1.3: Using two if statements instead of a case statement
16 --
17 ----------------------------------------------------------------------
18
19 library IEEE;
20 use IEEE.std_logic_1164.ALL;
21 use IEEE.std_logic_arith.ALL;
22 use work.types.all;
23
24 entity noc_fifo is
25
26   port(
27     --Common I/O signals
28     clk    : in bit_t;
29     rst    : in bit_t;
30     data_i     : in noc_data;
31     data_o     : out noc_data;
32
33     --Control signals
34     push     : in bit_t;
```

```
35      pop   : in bit_t;
36
37      --Status signals
38      count   : out FifoAddr
39          );
40
41  end noc_fifo;
42
43  architecture struc of noc_fifo is
44
45    --SIGNAL
46    SIGNAL mem  : FifoData;
47
48    --FIFO pointer
49    SIGNAL top  : fifo_vector;
50    SIGNAL bottom : fifo_vector;
51    SIGNAL counter  : fifo_vector;
52
53    --Add signal
54    SIGNAL one  : std_logic_vector(1 downto 0)  := conv_std_logic_vector(1, 2);
55
56  begin
57    count <= conv_integer(unsigned(counter));
58    data_o  <= mem(conv_integer(unsigned(bottom)));
59
60    -- Synchronous FIFO
61    fifo : process(clk, rst)
62    begin
63      if rst = '0' then
64        top <= (OTHERS => '0');
65        bottom  <= (OTHERS => '0');
66        counter <= (OTHERS => '0');
67      elsif clk'event and clk = '1' then
68        if(push = '1') then
69          mem(conv_integer(unsigned(top)))  <= data_i;
70          top   <= unsigned(top) + unsigned(one);
```

```
71        if(pop /= '1') then
72          counter   <= unsigned(counter) + unsigned(one);
73        end if;
74      end if;
75      if(pop = '1') then
76        bottom    <= unsigned(bottom) + unsigned(one);
77        if(push /= '1') then
78          counter   <=  unsigned(counter) - unsigned(one);
79        end if;
80      end if;
81    end process fifo;
82
83  end struc;
84
```

A.2.2 noc_na_master.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       NOC network adapter for mater component
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.1
6  --       Data last modified: 23/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.1: Some small optimisations made
14 --
15 ----------------------------------------------------------------------
16
17 library IEEE;
18 use IEEE.std_logic_1164.ALL;
19 use IEEE.std_logic_arith.ALL;
20 use work.types.all;
21
22 entity noc_na_master is
23
24   generic(
25     X    : integer := 0;
26     Y    : integer := 0
27   );
28
29   port(
30     --Common WishBone signals
31     wb_clk_i  : in bit_t;
32     wb_rst_i  : in bit_t;
33     wb_data_i : in word_t;
34     wb_data_o : out word_t;
```

```
35
36      --WishBone Master
37      wb_ack_o   : out bit_t;
38      wb_adr_i   : in word_t;
39      wb_cyc_i   : in bit_t;
40      wb_err_o   : out bit_t;
41      wb_rty_o   : out bit_t;
42      wb_sel_i   : in std_logic_vector(3 downto 0);
43      wb_we_i    : in bit_t;
44      wb_stb_i   : in bit_t;
45
46      --NoC signals
47      noc_in     : in noc_data;
48      noc_out    : out noc_data
49          );
50
51  end noc_na_master;
52
53  architecture struc of noc_na_master is
54
55    component noc_table
56      generic(
57        --Source
58        x_s : integer := X;
59        y_s : integer := y
60      );
61      port(
62        address : in std_logic_vector(addr_wdth-1 downto 0);
63        route : out std_logic_vector(route_wdth-1 downto 0)
64      );
65    end component;
66
67
68    SIGNAL    route     : std_logic_vector(route_wdth-1 downto 0);
69    --State register
70    type state is (STATE_INIT, STATE_WAIT, STATE_OUTPUT, STATE_RESPOND);  -- States for FSM
```

```
71    SIGNAL    current_state : state;
72    SIGNAL    next_state  : state;
73
74
75    --NOC registers
76    SIGNAL    reg_in    : noc_data;
77
78
79  begin --struc
80
81    wb_data_o <= reg_in(dat_start+data_wdth-1 downto dat_start);
82    wb_err_o  <= reg_in(err_start);
83    wb_rty_o  <= reg_in(rty_start);
84
85
86    table : noc_table
87    port map (
88        address   => wb_adr_i,
89        route   => route
90      );
91
92
93    fsm : process(noc_in, wb_stb_i, current_state, wb_data_i, wb_adr_i, wb_cyc_i, wb_sel_i, wb_we_i,
          route)
94    begin
95      wb_ack_o  <= '0';
96      noc_out   <= (OTHERS => '0');
97      case(current_state) is
98        when STATE_INIT => --Initial state
99          if(wb_stb_i = '1') then
100           next_state  <= STATE_OUTPUT;
101         else
102           next_state  <= STATE_INIT;
103         end if;
104       when STATE_OUTPUT =>
105         next_state  <= STATE_WAIT;
```

```
106         noc_out <= route & wb_data_i & "0" & wb_adr_i & "1100" & wb_sel_i & wb_we_i;
107       when STATE_WAIT =>
108         if(noc_in(ack_start) = '1') then
109           next_state  <= STATE_RESPOND;
110         else
111           next_state  <= STATE_WAIT;
112         end if;
113       when STATE_RESPOND =>
114         wb_ack_o  <= '1';
115         next_state  <= STATE_INIT;
116       when OTHERS =>
117         next_state  <= STATE_INIT;
118     end case;
119   end process;
120
121
122   next_state_register : process(wb_clk_i, wb_rst_i)
123   begin
124     if wb_rst_i = '0' then  -- asynchronous reset (active low)
125       current_state <= STATE_INIT;
126     elsif wb_clk_i'event and wb_clk_i = '1' then
127       current_state <= next_state;
128       if(current_state =  STATE_WAIT)then
129         reg_in  <= noc_in;
130       elsif(current_state =  STATE_INIT) then
131         reg_in  <= (OTHERS => '0');
132       end if;
133     end if;
134   end process next_state_register;
135
136 end struc;
```

A.2.3 noc_na_slave.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Network adapter for slave component
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.1
6  --       Data last modified: 23/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.0: Some small optimizations made
14 --
15 ----------------------------------------------------------------------
16
17 library IEEE;
18 use IEEE.std_logic_1164.ALL;
19 use IEEE.std_logic_arith.ALL;
20 use work.types.all;
21
22 entity noc_na_slave is
23
24   port(
25     --Common WishBone signals
26     wb_clk_i  : in bit_t;
27     wb_rst_i  : in bit_t;
28     wb_data_i : in word_t;
29     wb_data_o : out word_t;
30
31     --WishBone Master
32     wb_ack_i  : in bit_t;
33     wb_adr_o  : out word_t;
34     wb_cyc_o  : out bit_t;
```

```
35      wb_stb_o  : out bit_t;
36      wb_err_i  : in bit_t;
37      wb_rty_i  : in bit_t;
38      wb_sel_o  : out std_logic_vector(3 downto 0);
39      wb_we_o   : out bit_t;
40
41      --NoC signals
42      noc_in    : in noc_data;
43      noc_out   : out noc_data
44          );
45
46  end noc_na_slave;
47
48  architecture struc of noc_na_slave is
49
50    component noc_fifo
51      port(
52      --Common I/O signals
53      clk   : in bit_t;
54      rst   : in bit_t;
55      data_i    : in noc_data;
56      data_o    : out noc_data;
57
58      --Control signals
59      push    : in bit_t;
60      pop   : in bit_t;
61
62      --Status signals
63      count   : out FifoAddr
64          );
65    end component;
66
67    --State register
68    type state is (STATE_INIT, STATE_WAIT, STATE_OUTPUT);  -- States for FSM
69    SIGNAL    current_state : state;
70    SIGNAL    next_state  : state;
```

```
71
72
73    -- Fifo data
74    SIGNAL  fifo_out  : noc_data;
75
76    --Fifo status signals
77    SIGNAL  count   : FifoAddr;
78
79    --Fifo control signals
80    SIGNAL push   : bit_t;
81    SIGNAL pop    : bit_t;
82
83    --return route
84    SIGNAL  route     : std_logic_vector(route_wdth-1 downto 0);
85
86
87  begin --struc
88
89    wb_data_o <= fifo_out(dat_start+data_wdth-1 downto dat_start);
90    wb_adr_o  <= fifo_out(adr_start+addr_wdth-1 downto adr_start);
91    wb_cyc_o  <= fifo_out(cyc_start);
92    wb_sel_o  <= fifo_out(sel_start+3 downto sel_start);
93    wb_we_o   <= fifo_out(we_start);
94    push    <=  '1' WHEN noc_in(stb_start) = '1' else
95          '0';
96
97    fifo : noc_fifo
98      port map(
99        --Common I/O signals
100       clk   => wb_clk_i,
101       rst   => wb_rst_i,
102       data_i    => noc_in,
103       data_o    => fifo_out,
104
105       --Control signals
106       push    => push,
```

```
107         pop   => pop,
108
109         --Status signals
110         count   => count
111       );
112
113     fsm : process(wb_ack_i, wb_err_i, wb_rty_i, wb_data_i, current_state, count, fifo_out, route)
114     begin
115       wb_stb_o  <= '0';
116       route   <= (OTHERS => '0');
117       noc_out   <= (OTHERS => '0');
118       pop   <= '0';
119       case(current_state) is
120         when STATE_INIT => --Initial state
121           if(count /= 0) then
122             next_state  <= STATE_WAIT;
123           else
124             next_state  <= STATE_INIT;
125           end if;
126         when STATE_WAIT =>
127           --reverse route
128           for i in 0 to (route_wdth-dest_wdth)/2 loop
129             route(i*dest_wdth+dest_wdth-1 downto i*dest_wdth) <= fifo_out(NOC_DATA_WIDTH-dest_wdth-i*
                    dest_wdth+1 downto NOC_DATA_WIDTH-dest_wdth-i*dest_wdth);
130           end loop;
131           wb_stb_o  <= '1';
132           if(wb_ack_i = '1') then
133             next_state  <= STATE_OUTPUT;
134           else
135             next_state  <= STATE_WAIT;
136           end if;
137           noc_out <= route & wb_data_i & wb_ack_i & CONV_STD_LOGIC_VECTOR(0, addr_wdth) & "00" & wb_err_i
                & wb_rty_i & "00000";
138         when STATE_OUTPUT =>
139           next_state  <= STATE_INIT;
140           pop   <= '1';
```

```
141       when OTHERS =>
142         next_state  <= STATE_INIT;
143       end case;
144     end process;
145
146     next_state_register : process(wb_clk_i, wb_rst_i)
147     begin
148       if wb_rst_i = '0' then  -- asynchronous reset (active low)
149         current_state <= STATE_INIT;
150       elsif wb_clk_i'event and wb_clk_i = '1' then
151         current_state <= next_state;
152       end if;
153     end process next_state_register;
154
155   end struc;
```

A.2.4 noc_router.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Router for network
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 14/3-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity noc_router is
22
23   port(
24     --I/O ports
25     clk_i    : in bit_t;
26     rst_i    : in bit_t;
27
28     --NoC signals
29     --North
30     noc_n_i    : in noc_data;
31     noc_n_o    : out noc_data;
32
33     --South
34     noc_s_i    : in noc_data;
```

```
35      noc_s_o   : out noc_data;
36
37      --East
38      noc_e_i   : in noc_data;
39      noc_e_o   : out noc_data;
40
41      --West
42      noc_w_i   : in noc_data;
43      noc_w_o   : out noc_data;
44
45      --IP
46      ip_i      : in noc_data;
47      ip_o      : out noc_data
48          );
49
50  end noc_router;
51
52  architecture struc of noc_router is
53
54  component noc_router_mux
55    port(
56      -- Source selector
57      select_north_i  : in source;
58      select_south_i  : in source;
59      select_east_i : in source;
60      select_west_i : in source;
61      select_ip_i : in source;
62
63      --data in
64      north_i     : in noc_data;
65      south_i     : in noc_data;
66      east_i      : in noc_data;
67      west_i      : in noc_data;
68      ip_i      : in noc_data;
69
70      --data out
```

```
71    north_o   : out noc_data;
72    south_o   : out noc_data;
73    east_o    : out noc_data;
74    west_o    : out noc_data;
75    ip_o      : out noc_data
76        );
77    end component;
78
79    component noc_arbiter
80    port(
81      --I/O ports
82      clk_i   : in bit_t;
83      rst_i   : in bit_t;
84
85      --NoC signals
86      --North
87      rout_north  : in dest;
88      count_north : in FifoAddr;
89      select_north_o  : out source;
90      pop_north : out bit_t;
91
92      --South
93      rout_south  : in dest;
94      count_south : in FifoAddr;
95      select_south_o  : out source;
96      pop_south : out bit_t;
97
98      --East
99      rout_east : in dest;
100     count_east  : in FifoAddr;
101     select_east_o : out source;
102     pop_east  : out bit_t;
103
104     --West
105     rout_west : in dest;
106     count_west  : in FifoAddr;
```

```
107       select_west_o : out source;
108       pop_west  : out bit_t;
109
110       --IP
111       rout_ip   : in dest;
112       count_ip  : in FifoAddr;
113       select_ip_o : out source;
114       pop_ip    : out bit_t
115           );
116     end component;
117
118     component noc_fifo
119       port(
120       --Common I/O signals
121       clk   : in bit_t;
122       rst   : in bit_t;
123       data_i    : in noc_data;
124       data_o    : out noc_data;
125
126       --Control signals
127       push    : in bit_t;
128       pop   : in bit_t;
129
130       --Status signals
131       count   : out FifoAddr
132           );
133     end component;
134
135     -- Select signals
136     SIGNAL  select_north  : source;
137     SIGNAL  select_south  : source;
138     SIGNAL  select_east : source;
139     SIGNAL  select_west : source;
140     SIGNAL  select_ip : source;
141
142     -- Fifo data
```

```
143   SIGNAL  fifo_north  : noc_data;
144   SIGNAL  fifo_south  : noc_data;
145   SIGNAL  fifo_east : noc_data;
146   SIGNAL  fifo_west : noc_data;
147   SIGNAL  fifo_ip   : noc_data;
148
149   --Fifo status signals
150   SIGNAL  count_north : FifoAddr;
151   SIGNAL  count_south : FifoAddr;
152   SIGNAL  count_east  : FifoAddr;
153   SIGNAL  count_west  : FifoAddr;
154   SIGNAL  count_ip  : FifoAddr;
155
156   --Fifo control signals
157   SIGNAL push_north : bit_t;
158   SIGNAL push_south : bit_t;
159   SIGNAL push_east  : bit_t;
160   SIGNAL push_west  : bit_t;
161   SIGNAL push_ip    : bit_t;
162   SIGNAL pop_north  : bit_t;
163   SIGNAL pop_south  : bit_t;
164   SIGNAL pop_east   : bit_t;
165   SIGNAL pop_west   : bit_t;
166   SIGNAL pop_ip   : bit_t;
167
168 begin
169   north_fifo : noc_fifo
170     port map(
171       --Common I/O signals
172       clk    => clk_i,
173       rst    => rst_i,
174       data_i    => noc_n_i,
175       data_o    => fifo_north,
176
177       --Control signals
178       push    => push_north,
```

```
179     pop   => pop_north,
180
181     --Status signals
182     count   => count_north
183   );
184
185   south_fifo : noc_fifo
186     port map(
187       --Common I/O signals
188       clk   => clk_i,
189       rst   => rst_i,
190       data_i    => noc_s_i,
191       data_o    => fifo_south,
192
193       --Control signals
194       push    => push_south,
195       pop   => pop_south,
196
197       --Status signals
198       count   => count_south
199     );
200
201   east_fifo : noc_fifo
202     port map(
203       --Common I/O signals
204       clk   => clk_i,
205       rst   => rst_i,
206       data_i    => noc_e_i,
207       data_o    => fifo_east,
208
209       --Control signals
210       push    => push_east,
211       pop   => pop_east,
212
213       --Status signals
214       count   => count_east
```

```
215       );
216
217     west_fifo : noc_fifo
218       port map(
219         --Common I/O signals
220         clk    => clk_i,
221         rst    => rst_i,
222         data_i     => noc_w_i,
223         data_o     => fifo_west,
224
225         --Control signals
226         push     => push_west,
227         pop    => pop_west,
228
229         --Status signals
230         count    => count_west
231       );
232
233     ip_fifo : noc_fifo
234       port map(
235         --Common I/O signals
236         clk    => clk_i,
237         rst    => rst_i,
238         data_i     => ip_i,
239         data_o     => fifo_ip,
240
241         --Control signals
242         push     => push_ip,
243         pop    => pop_ip,
244
245         --Status signals
246         count    => count_ip
247       );
248
249     arbiter : noc_arbiter
250       port map(
```

```
251         --I/O ports
252         clk_i    => clk_i,
253         rst_i    => rst_i,
254 
255         --NoC signals
256         --North
257         rout_north  => fifo_north(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2),
258         count_north => count_north,
259         select_north_o  => select_north,
260         pop_north => pop_north,
261 
262         --South
263         rout_south  => fifo_south(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2),
264         count_south => count_south,
265         select_south_o  => select_south,
266         pop_south => pop_south,
267 
268           --East
269         rout_east => fifo_east(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2),
270         count_east  => count_east,
271         select_east_o => select_east,
272         pop_east  => pop_east,
273 
274         --West
275         rout_west => fifo_west(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2),
276         count_west  => count_west,
277         select_west_o => select_west,
278         pop_west  => pop_west,
279 
280         --IP
281         rout_ip   => fifo_ip(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-2),
282         count_ip  => count_ip,
283         select_ip_o => select_ip,
284         pop_ip    => pop_ip
285 
286     );
```

```
287
288   mux : noc_router_mux
289     port map(
290       -- Source selector
291       select_north_i  => select_north,
292       select_south_i  => select_south,
293       select_east_i => select_east,
294       select_west_i => select_west,
295       select_ip_i => select_ip,
296
297       --data in
298       north_i   => fifo_north,
299       south_i   => fifo_south,
300       east_i    => fifo_east,
301       west_i    => fifo_west,
302       ip_i    => fifo_ip,
303
304       --data out
305       north_o   => noc_n_o,
306       south_o   => noc_s_o,
307       east_o    => noc_e_o,
308       west_o    => noc_w_o,
309       ip_o    => ip_o
310
311   );
312
313   link_controller : process(noc_n_i, noc_s_i, noc_e_i, noc_w_i, ip_i)
314   begin
315     push_north  <= '0';
316     push_south  <= '0';
317     push_east <= '0';
318     push_west <= '0';
319     push_ip   <= '0';
320     if(noc_n_i(stb_start) = '1' or noc_n_i(ack_start) = '1') then
321       push_north  <= '1';
322     end if;
```

```
323     if(noc_s_i(stb_start) = '1' or noc_s_i(ack_start) = '1') then
324       push_south  <= '1';
325     end if;
326     if(noc_e_i(stb_start) = '1' or noc_e_i(ack_start) = '1') then
327       push_east <= '1';
328     end if;
329     if(noc_w_i(stb_start) = '1' or noc_w_i(ack_start) = '1') then
330       push_west <= '1';
331     end if;
332     if(ip_i(stb_start) = '1' or ip_i(ack_start) = '1') then
333       push_ip   <= '1';
334     end if;
335   end process link_controller;
336
337 end struc;
```

A.2.5 noc_router_arbiter.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Arbiter for router
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.1
6  --       Data last modified: 23/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.1: Optimized for area
14 --
15 ----------------------------------------------------------------------
16
17 library IEEE;
18 use IEEE.std_logic_1164.ALL;
19 use IEEE.std_logic_arith.ALL;
20 use work.types.all;
21
22 entity noc_arbiter is
23
24   port(
25     --I/O ports
26     clk_i    : in bit_t;
27     rst_i    : in bit_t;
28
29     --NoC signals
30     --North
31     rout_north  : in dest;
32     count_north : in FifoAddr;
33     select_north_o  : out source;
34     pop_north : out bit_t;
```

```
35
36      --South
37      rout_south  : in dest;
38      count_south : in FifoAddr;
39      select_south_o  : out source;
40      pop_south : out bit_t;
41
42      --East
43      rout_east : in dest;
44      count_east  : in FifoAddr;
45      select_east_o : out source;
46      pop_east  : out bit_t;
47
48      --West
49      rout_west : in dest;
50      count_west  : in FifoAddr;
51      select_west_o : out source;
52      pop_west  : out bit_t;
53
54      --IP
55      rout_ip   : in dest;
56      count_ip  : in FifoAddr;
57      select_ip_o : out source;
58      pop_ip    : out bit_t
59          );
60
61  end noc_arbiter;
62
63  architecture struc of noc_arbiter is
64
65    --State signals
66    SIGNAL north_state  : source  := source_none;
67    SIGNAL north_next_state : source  := source_none;
68    SIGNAL south_state  : source  := source_none;
69    SIGNAL south_next_state : source  := source_none;
70    SIGNAL east_state : source  := source_none;
```

```
71    SIGNAL east_next_state  : source  := source_none;
72    SIGNAL west_state : source  := source_none;
73    SIGNAL west_next_state  : source  := source_none;
74    SIGNAL ip_state   : source  := source_none;
75    SIGNAL ip_next_state  : source  := source_none;
76
77    --Pop signals
78    SIGNAL north_pop_south  : bit_t;
79    SIGNAL north_pop_east : bit_t;
80    SIGNAL north_pop_west : bit_t;
81    SIGNAL north_pop_ip : bit_t;
82    SIGNAL south_pop_north  : bit_t;
83    SIGNAL south_pop_east : bit_t;
84    SIGNAL south_pop_west : bit_t;
85    SIGNAL south_pop_ip : bit_t;
86    SIGNAL east_pop_north : bit_t;
87    SIGNAL east_pop_south : bit_t;
88    SIGNAL east_pop_west  : bit_t;
89    SIGNAL east_pop_ip  : bit_t;
90    SIGNAL west_pop_north : bit_t;
91    SIGNAL west_pop_south : bit_t;
92    SIGNAL west_pop_east  : bit_t;
93    SIGNAL west_pop_ip  : bit_t;
94    SIGNAL ip_pop_north : bit_t;
95    SIGNAL ip_pop_south : bit_t;
96    SIGNAL ip_pop_east  : bit_t;
97    SIGNAL ip_pop_west  : bit_t;
98
99
100
101
102 begin
103
104   -- Pop signals
105   pop_north <= south_pop_north or east_pop_north or west_pop_north or ip_pop_north;
106   pop_south <= north_pop_south or east_pop_south or west_pop_south or ip_pop_south;
```

```
107   pop_east  <= north_pop_east or south_pop_east or west_pop_east or ip_pop_east;
108   pop_west  <= north_pop_west or south_pop_west or east_pop_west or ip_pop_west;
109   pop_ip    <= north_pop_ip or south_pop_ip or east_pop_ip or west_pop_ip;
110   -- Round robin implementation
111
112   --Arbiter for north gate
113
114   north : process(north_state, rout_south, rout_east, rout_west,rout_ip, count_south, count_east,
          count_west, count_ip)
115   begin
116     north_pop_south   <= '0';
117     north_pop_east    <= '0';
118     north_pop_west    <= '0';
119     north_pop_ip    <= '0';
120     north_next_state  <= north_state;
121     case(north_state) is
122     when source_ip => -- Last was IP core
123       if rout_south = dest_north and count_south /= 0 then
124         north_next_state  <= source_south;  -- Choose south
125         north_pop_south   <= '1';
126       elsif rout_east = dest_north and count_east /= 0 then
127         north_next_state  <= source_east;   -- Choose east
128         north_pop_east    <= '1';
129       elsif rout_west = dest_north and count_west /= 0 then
130         north_next_state  <= source_west;   -- Choose west
131         north_pop_west    <= '1';
132       elsif rout_ip = dest_north and count_ip /= 0 then
133         north_next_state  <= source_ip;   -- Choose IP core
134         north_pop_ip    <= '1';
135       else
136         north_next_state  <= source_none;   -- Choose nothing
137       end if;
138     when source_south =>  -- Last was south
139       if rout_east = dest_north and count_east /= 0 then
140         north_next_state  <= source_east;   -- Choose east
141         north_pop_east    <= '1';
```

```
142       elsif rout_west = dest_north and count_west /= 0 then
143         north_next_state  <= source_west;   -- Choose west
144         north_pop_west    <= '1';
145       elsif rout_ip = dest_north and count_ip /= 0 then
146         north_next_state  <= source_ip;   -- Choose IP core
147         north_pop_ip    <= '1';
148       elsif rout_south = dest_north and count_south /= 0 then
149         north_next_state  <= source_south;  -- Choose south
150         north_pop_south   <= '1';
151       else
152         north_next_state  <= source_none;   -- Choose nothing
153       end if;
154     when source_east => -- Last was east
155       if rout_west = dest_north and count_west /= 0 then
156         north_next_state  <= source_west;   -- Choose west
157         north_pop_west    <= '1';
158       elsif rout_ip = dest_north and count_ip /= 0 then
159         north_next_state  <= source_ip;   -- Choose IP core
160         north_pop_ip    <= '1';
161       elsif rout_south = dest_north and count_south /= 0 then
162         north_next_state  <= source_south;  -- Choose south
163         north_pop_south   <= '1';
164       elsif rout_east = dest_north and count_east /= 0 then
165         north_next_state  <= source_east;   -- Choose east
166         north_pop_east    <= '1';
167       else
168         north_next_state  <= source_none;   -- Choose nothing
169       end if;
170     when others => -- Last was west or none
171       if rout_ip = dest_north and count_ip /= 0 then
172         north_next_state  <= source_ip;   -- Choose IP core
173         north_pop_ip    <= '1';
174       elsif rout_south = dest_north and count_south /= 0 then
175         north_next_state  <= source_south;  -- Choose south
176         north_pop_south   <= '1';
177       elsif rout_east = dest_north and count_east /= 0 then
```

```
178         north_next_state  <= source_east;   -- Choose east
179         north_pop_east    <= '1';
180       elsif rout_west = dest_north and count_west /= 0 then
181         north_next_state  <= source_west;   -- Choose west
182         north_pop_west    <= '1';
183       else
184         north_next_state  <= source_none;   -- Choose nothing
185       end if;
186     end case;
187   end process north;
188
189   --Arbiter for south gate
190
191   south : process(south_state, rout_north, rout_east, rout_west, rout_ip, count_north, count_east,
        count_west, count_ip)
192   begin
193     south_pop_north   <= '0';
194     south_pop_east    <= '0';
195     south_pop_west    <= '0';
196     south_pop_ip    <= '0';
197     south_next_state  <= south_state;
198     case(south_state) is
199     when source_ip => -- Last was IP core
200       if rout_north = dest_south and count_north /= 0 then
201         south_next_state  <= source_north;  -- Choose north
202         south_pop_north   <= '1';
203       elsif rout_east = dest_south and count_east /= 0 then
204         south_next_state  <= source_east;   -- Choose east
205         south_pop_east    <= '1';
206       elsif rout_west = dest_south and count_west /= 0 then
207         south_next_state  <= source_west;   -- Choose west
208         south_pop_west    <= '1';
209       elsif rout_ip = dest_south and count_ip /= 0 then
210         south_next_state  <= source_ip;   -- Choose IP core
211         south_pop_ip    <= '1';
212       else
```

```
213         south_next_state  <= source_none;   -- Choose nothing
214       end if;
215     when source_north =>  -- Last was south
216       if rout_east = dest_south and count_east /= 0 then
217         south_next_state  <= source_east;   -- Choose east
218         south_pop_east    <= '1';
219       elsif rout_west = dest_south and count_west /= 0 then
220         south_next_state  <= source_west;   -- Choose west
221         south_pop_west    <= '1';
222       elsif rout_ip = dest_south and count_ip /= 0 then
223         south_next_state  <= source_ip;   -- Choose IP core
224         south_pop_ip    <= '1';
225       elsif rout_north = dest_south and count_north /= 0 then
226         south_next_state  <= source_north;  -- Choose north
227         south_pop_north   <= '1';
228       else
229         south_next_state  <= source_none;   -- Choose nothing
230       end if;
231     when source_east => -- Last was east
232       if rout_west = dest_south and count_west /= 0 then
233         south_next_state  <= source_west;   -- Choose west
234         south_pop_west    <= '1';
235       elsif rout_ip = dest_south and count_ip /= 0 then
236         south_next_state  <= source_ip;   -- Choose IP core
237         south_pop_ip    <= '1';
238       elsif rout_north = dest_south and count_north /= 0 then
239         south_next_state  <= source_north;  -- Choose north
240         south_pop_north   <= '1';
241       elsif rout_east = dest_south and count_east /= 0 then
242         south_next_state  <= source_east;   -- Choose east
243         south_pop_east    <= '1';
244       else
245         south_next_state  <= source_none;   -- Choose nothing
246       end if;
247     when others =>-- Last was west or none
248       if rout_ip = dest_south and count_ip /= 0 then
```

```
249         south_next_state  <= source_ip;    -- Choose IP core
250         south_pop_ip     <= '1';
251       elsif rout_north = dest_south and count_north /= 0 then
252         south_next_state  <= source_north;  -- Choose north
253         south_pop_north   <= '1';
254       elsif rout_east = dest_south and count_east /= 0 then
255         south_next_state  <= source_east;   -- Choose east
256         south_pop_east    <= '1';
257       elsif rout_west = dest_south and count_west /= 0 then
258         south_next_state  <= source_west;   -- Choose west
259         south_pop_west    <= '1';
260       else
261         south_next_state  <= source_none;   -- Choose nothing
262       end if;
263     end case;
264   end process south;
265
266   --Arbiter for east gate
267
268   east : process(east_state, rout_north, rout_south, rout_west, rout_ip, count_north, count_south,
          count_west, count_ip)
269   begin
270     east_pop_north    <= '0';
271     east_pop_south    <= '0';
272     east_pop_west   <= '0';
273     east_pop_ip   <= '0';
274     east_next_state <= east_state;
275     case(east_state) is
276     when source_ip => -- Last was IP core
277       if rout_north = dest_east and count_north /= 0 then
278         east_next_state <= source_north;    -- Choose north
279         east_pop_north    <= '1';
280       elsif rout_south = dest_east and count_south /= 0 then
281         east_next_state <= source_south;    -- Choose south
282         east_pop_south    <= '1';
283       elsif rout_west = dest_east and count_west /= 0 then
```

```
284     east_next_state <= source_west;      -- Choose west
285     east_pop_west   <= '1';
286   elsif rout_ip = dest_east and count_ip /= 0 then
287     east_next_state <= source_ip;      -- Choose IP core
288     east_pop_ip   <= '1';
289   else
290     east_next_state <= source_none;      -- Choose nothing
291   end if;
292 when source_north =>   -- Last was north
293   if rout_south = dest_east and count_south /= 0 then
294     east_next_state <= source_south;      -- Choose south
295     east_pop_south    <= '1';
296   elsif rout_west = dest_east and count_west /= 0 then
297     east_next_state <= source_west;      -- Choose west
298     east_pop_west   <= '1';
299   elsif rout_ip = dest_east and count_ip /= 0 then
300     east_next_state <= source_ip;      -- Choose IP core
301     east_pop_ip   <= '1';
302   elsif rout_north = dest_east and count_north /= 0 then
303     east_next_state <= source_north;      -- Choose north
304     east_pop_north    <= '1';
305   else
306     east_next_state <= source_none;      -- Choose nothing
307   end if;
308 when source_south =>   -- Last was south
309   if rout_west = dest_east and count_west /= 0 then
310     east_next_state <= source_west;      -- Choose west
311     east_pop_west   <= '1';
312   elsif rout_ip = dest_east and count_ip /= 0 then
313     east_next_state <= source_ip;      -- Choose IP core
314     east_pop_ip   <= '1';
315   elsif rout_north = dest_east and count_north /= 0 then
316     east_next_state <= source_north;      -- Choose north
317     east_pop_north    <= '1';
318   elsif rout_south = dest_east and count_south /= 0 then
319     east_next_state <= source_south;      -- Choose south
```

```
320         east_pop_south    <= '1';
321       else
322         east_next_state <= source_none;      -- Choose nothing
323       end if;
324     when others => -- Last was west or none
325       if rout_ip = dest_east and count_ip /= 0 then
326         east_next_state <= source_ip;      -- Choose IP core
327         east_pop_ip   <= '1';
328       elsif rout_north = dest_east and count_north /= 0 then
329         east_next_state <= source_north;     -- Choose north
330         east_pop_north    <= '1';
331       elsif rout_south = dest_east and count_south /= 0 then
332         east_next_state <= source_south;     -- Choose south
333         east_pop_south    <= '1';
334       elsif rout_west = dest_east and count_west /= 0 then
335         east_next_state <= source_west;      -- Choose west
336         east_pop_west   <= '1';
337       else
338         east_next_state <= source_none;      -- Choose nothing
339       end if;
340     end case;
341   end process east;
342
343   --Arbiter for west gate
344
345   west : process(west_state, rout_north, rout_south, rout_east, rout_ip, count_north, count_south,
        count_east, count_ip)
346   begin
347     west_pop_north    <= '0';
348     west_pop_south    <= '0';
349     west_pop_east   <= '0';
350     west_pop_ip   <= '0';
351     west_next_state <= west_state;
352     case(west_state) is
353     when source_ip => -- Last was IP core
354       if rout_north = dest_west and count_north /= 0 then
```

```
355     west_next_state <= source_north;     -- Choose north
356       west_pop_north    <= '1';
357   elsif rout_south = dest_west and count_south /= 0 then
358     west_next_state <= source_south;     -- Choose south
359     west_pop_south    <= '1';
360   elsif rout_east = dest_west and count_east /= 0 then
361     west_next_state <= source_east;      -- Choose east
362     west_pop_east     <= '1';
363   elsif rout_ip = dest_west and  count_ip /= 0  then
364     west_next_state <= source_ip;        -- Choose IP core
365     west_pop_ip     <= '1';
366   else
367     west_next_state <= source_none;      -- Choose nothing
368   end if;
369 when source_north =>  -- Last was north
370   if rout_south = dest_west and count_south /= 0 then
371     west_next_state <= source_south;     -- Choose south
372     west_pop_south    <= '1';
373   elsif rout_east = dest_west and count_east /= 0 then
374     west_next_state <= source_east;      -- Choose east
375     west_pop_east     <= '1';
376   elsif rout_ip = dest_west and count_ip /= 0 then
377     west_next_state <= source_ip;        -- Choose IP core
378     west_pop_ip     <= '1';
379   elsif rout_north = dest_west and count_north /= 0 then
380     west_next_state <= source_north;     -- Choose north
381     west_pop_north    <= '1';
382   else
383     west_next_state <= source_none;      -- Choose nothing
384   end if;
385 when source_south =>  -- Last was south
386   if rout_east = dest_west and count_east /= 0 then
387     west_next_state <= source_east;      -- Choose east
388     west_pop_east     <= '1';
389   elsif rout_ip = dest_west and count_ip /= 0 then
390     west_next_state <= source_ip;        -- Choose IP core
```

```
391         west_pop_ip   <= '1';
392       elsif rout_north = dest_west and count_north /= 0 then
393         west_next_state <= source_north;     -- Choose north
394         west_pop_north    <= '1';
395       elsif rout_south = dest_west and count_south /= 0 then
396         west_next_state <= source_south;     -- Choose south
397         west_pop_south    <= '1';
398       else
399         west_next_state <= source_none;      -- Choose nothing
400       end if;
401     when others =>  -- Last was east or none
402       if rout_ip = dest_west and count_ip /= 0 then
403         west_next_state <= source_ip;      -- Choose IP core
404         west_pop_ip   <= '1';
405       elsif rout_north = dest_west and count_north /= 0 then
406         west_next_state <= source_north;     -- Choose north
407         west_pop_north    <= '1';
408       elsif rout_south = dest_west and count_south /= 0 then
409         west_next_state <= source_south;     -- Choose south
410         west_pop_south    <= '1';
411       elsif rout_east = dest_west and count_east /= 0 then
412         west_next_state <= source_east;      -- Choose east
413         west_pop_east   <= '1';
414       else
415         west_next_state <= source_none;      -- Choose nothing
416       end if;
417     end case;
418   end process west;
419
420   --Arbiter for ip gate
421
422   ip : process(ip_state, rout_north, rout_south, rout_east, rout_west, count_north, count_south,
        count_east, count_west)
423   begin
424     ip_pop_north    <= '0';
425     ip_pop_south    <= '0';
```

```
426       ip_pop_east   <= '0';
427       ip_pop_west   <= '0';
428       ip_next_state <= ip_state;
429       case(ip_state) is
430       when source_west => -- Last was IP core
431         if rout_north = dest_north and count_north /= 0 then
432           ip_next_state <= source_north;     -- Choose north
433           ip_pop_north    <= '1';
434         elsif rout_south = dest_south and count_south /= 0 then
435           ip_next_state <= source_south;     -- Choose south
436           ip_pop_south    <= '1';
437         elsif rout_east = dest_east and count_east /= 0 then
438           ip_next_state <= source_east;      -- Choose east
439           ip_pop_east   <= '1';
440         elsif rout_west = dest_west and count_west /= 0 then
441           ip_next_state <= source_west;      -- Choose west
442           ip_pop_west   <= '1';
443         else
444           ip_next_state <= source_none;      -- Choose nothing
445         end if;
446       when source_north =>  -- Last was south
447         if rout_south = dest_south and count_south /= 0 then
448           ip_next_state <= source_south;     -- Choose south
449           ip_pop_south    <= '1';
450         elsif rout_east = dest_east and count_east /= 0 then
451           ip_next_state <= source_east;      -- Choose east
452           ip_pop_east   <= '1';
453         elsif rout_west = dest_west and count_west /= 0 then
454           ip_next_state <= source_west;      -- Choose west
455           ip_pop_west   <= '1';
456         elsif rout_north = dest_north and count_north /= 0 then
457           ip_next_state <= source_north;     -- Choose north
458           ip_pop_north    <= '1';
459         else
460           ip_next_state <= source_none;      -- Choose nothing
461         end if;
```

```
462       when source_south =>  -- Last was south
463         if rout_east = dest_east and count_east /= 0 then
464           ip_next_state <= source_east;      -- Choose east
465           ip_pop_east   <= '1';
466         elsif rout_west = dest_west and count_west /= 0 then
467           ip_next_state <= source_west;      -- Choose west
468           ip_pop_west   <= '1';
469         elsif rout_north = dest_north and count_north /= 0 then
470           ip_next_state <= source_north;     -- Choose north
471           ip_pop_north    <= '1';
472         elsif rout_south = dest_south and count_south /= 0 then
473           ip_next_state <= source_south;     -- Choose south
474           ip_pop_south    <= '1';
475         else
476           ip_next_state <= source_none;      -- Choose nothing
477         end if;
478       when others =>  -- Last was east or none
479         if rout_west = dest_west and count_west /= 0 then
480           ip_next_state <= source_west;      -- Choose west
481           ip_pop_west   <= '1';
482         elsif rout_north = dest_north and count_north /= 0 then
483           ip_next_state <= source_north;     -- Choose north
484           ip_pop_north    <= '1';
485         elsif rout_south = dest_south and count_south /= 0 then
486           ip_next_state <= source_south;     -- Choose south
487           ip_pop_south    <= '1';
488         elsif rout_east = dest_east and count_east /= 0 then
489           ip_next_state <= source_east;      -- Choose east
490           ip_pop_east   <= '1';
491         else
492           ip_next_state <= source_none;      -- Choose nothing
493         end if;
494       end case;
495     end process ip;
496
497     next_state_register : process(clk_i, rst_i)
```

```
498  begin
499    if rst_i = '0' then -- asynchronous reset (active low)
500      north_state <= source_none;
501      south_state <= source_none;
502      east_state  <= source_none;
503      west_state  <= source_none;
504      ip_state  <= source_none;
505    elsif clk_i'event and clk_i = '1' then
506      north_state <= north_next_state;
507      south_state <= south_next_state;
508      east_state  <= east_next_state;
509      west_state  <= west_next_state;
510      ip_state  <= ip_next_state;
511    end if;
512  end process next_state_register;
513
514  select_north_o  <= north_next_state;
515  select_south_o  <= south_next_state;
516  select_east_o <= east_next_state;
517  select_west_o <= west_next_state;
518  select_ip_o <= ip_next_state;
519
520  end struc;
```

A.2.6 noc_router_mux.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Arbiter for router
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.3
6  --       Data last modified: 23/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --       v1.1: Some optimisations to improve space
14 --       v1.2: Further optimizations, did not affect used area
15 --       v1.3: joined the with statements
16 --
17 ----------------------------------------------------------------------
18
19
20 library IEEE;
21 use IEEE.std_logic_1164.ALL;
22 use IEEE.std_logic_arith.ALL;
23 use work.types.all;
24
25 entity noc_router_mux is
26
27   port(
28     -- Source selector
29     select_north_i  : in source;
30     select_south_i  : in source;
31     select_east_i : in source;
32     select_west_i : in source;
33     select_ip_i : in source;
34
```

```
35      --data in
36      north_i    : in noc_data;
37      south_i    : in noc_data;
38      east_i     : in noc_data;
39      west_i     : in noc_data;
40      ip_i     : in noc_data;
41
42      --data out
43      north_o    : out noc_data;
44      south_o    : out noc_data;
45      east_o     : out noc_data;
46      west_o     : out noc_data;
47      ip_o     : out noc_data
48    );
49  end noc_router_mux;
50
51  architecture struc of noc_router_mux is
52
53  begin
54
55    --North data
56    WITH select_north_i SELECT
57    north_o <=
58      south_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_south & south_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
59        when source_south,
60      east_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_east & east_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
61        when source_east,
62      west_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_west & west_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
63        when source_west,
64      ip_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_north & ip_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
65        when source_ip,
66      (OTHERS => '0')               when source_none,
```

```
67      (OTHERS => 'X')                when others;
68
69
70    --South data
71    WITH select_south_i SELECT
72    south_o <=
73      north_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_north & north_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
74        when source_north,
75      east_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_east & east_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
76        when source_east,
77      west_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_west & west_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
78        when source_west,
79      ip_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_south & ip_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
80        when source_ip,
81      (OTHERS => '0')                when source_none,
82      (OTHERS => 'X')                when others;
83
84    --East data
85    WITH select_east_i SELECT
86    east_o  <=
87      north_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_north & north_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
88        when source_north,
89      south_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_south & south_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
90        when source_south,
91      west_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_west & west_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
92        when source_west,
93      ip_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth)  & dest_east & ip_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
94        when source_ip,
```

```
95      (OTHERS => '0')               when source_none,
96      (OTHERS => 'X')               when others;
97
98    --West data
99    WITH select_west_i SELECT
100   west_o  <=
101     north_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_north & north_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
102       when source_north,
103     south_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_south & south_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
104       when source_south,
105     east_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_east & east_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
106       when source_east,
107     ip_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth)  & dest_west & ip_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
108       when source_ip,
109     (OTHERS => '0')               when source_none,
110     (OTHERS => 'X')               when others;
111   --IP data
112   WITH select_ip_i SELECT
113   ip_o  <=
114     north_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_north & north_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
115       when source_north,
116     south_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_south & south_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
117       when source_south,
118     east_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_east & east_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
119       when source_east,
120     west_i(NOC_DATA_WIDTH-3 downto NOC_DATA_WIDTH-route_wdth) & dest_west & west_i(NOC_DATA_WIDTH-
          route_wdth-1 downto 0)
121       when source_west,
122     (OTHERS => '0')               when source_none,
```

```
123     (OTHERS => 'X')      when others;
124
125  end struc;
```

A.2.7 arbiter_tb.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Arbiter test bench
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 10/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity arbiter_tb is
22
23 end arbiter_tb;
24
25 architecture behav of arbiter_tb is
26
27   component noc_arbiter
28   port(
29     --I/O ports
30     clk_i    : in bit_t;
31     rst_i    : in bit_t;
32
33     --NoC signals
34     --North
```

```
35    n_direction : in dest;
36    n_select  : out source;
37
38    --South
39    s_direction : in dest;
40    s_select  : out source;
41
42    --East
43    e_direction : in dest;
44    e_select  : out source;
45
46    --West
47    w_direction : in dest;
48    w_select  : out source;
49
50    --IP
51    i_direction : in dest;
52    i_select  : out source
53        );
54  end component;
55
56
57  SIGNAL clk    : bit_t;
58  SIGNAL rst    : bit_t;
59  --north
60  SIGNAL n_direction  : dest;
61  SIGNAL n_select   : source;
62
63  --South
64  SIGNAL s_direction  : dest;
65  SIGNAL s_select   : source;
66
67  --East
68  SIGNAL e_direction  : dest;
69  SIGNAL e_select   : source;
70
```

```
71  --West
72  SIGNAL w_direction  : dest;
73  SIGNAL w_select     : source;
74
75  --IP
76  SIGNAL i_direction  : dest;
77  SIGNAL i_select     : source;
78
79  SIGNAL state        : std_logic_vector(3 downto 0);
80  SIGNAL next_state   : std_logic_vector(3 downto 0);
81
82  --Expect
83  SIGNAL n_exp        : source;
84  SIGNAL s_exp        : source;
85  SIGNAL e_exp        : source;
86  SIGNAL w_exp        : source;
87  SIGNAL i_exp        : source;
88
89  begin
90
91  clock: process
92  begin  -- process clock
93    clk <= '0';
94    wait for 5 ns;
95    clk <= '1';
96    wait for 5 ns;
97  end process clock;
98
99  process
100 begin  -- process
101   rst <= '0';
102   wait for 12 ns;
103   rst <= '1';
104   wait;
105 end process;
106
```

```
107   next_state_register : process(clk, rst)
108   begin
109     if rst = '0' then -- asynchronous reset (active low=
110       state <= "0000";
111     elsif clk'event and clk = '1' then
112       state <= next_state;
113     end if;
114   end process next_state_register;
115
116   test :  process(state)
117   begin
118     case(state) is
119       when "0000" =>  --All north
120         next_state  <= "0001";
121         --NoC signals
122         --North
123         n_direction <= dest_north;
124         n_exp   <= "111";
125         --South
126         s_direction <= dest_north;
127         s_exp   <= source_none;
128         --East
129         e_direction <= dest_north;
130         e_exp   <= source_none;
131         --West
132         w_direction <= dest_north;
133         w_exp   <= source_none;
134         --IP
135         i_direction <= dest_north;
136         i_exp   <= source_north;
137       when "0001" =>  -- Two and two
138         next_state  <= "0010";
139         --NoC signals
140         --North
141         n_direction <= dest_south;
142         n_exp   <= "111";
```

```
143       --South
144       s_direction <= dest_west;
145       s_exp   <= "111";
146       --East
147       e_direction <= dest_west;
148       e_exp   <= "111";
149       --West
150       w_direction <= dest_north;
151       w_exp   <= "110";
152       --IP
153       i_direction <= dest_north;
154       i_exp   <= "111";
155     when others =>
156       next_state  <= "0000";
157   end case;
158 end process test;
159
160
161 arbiter : noc_arbiter
162   port map(
163     --I/O ports
164     clk_i   => clk,
165     rst_i   => rst,
166
167     --NoC signals
168     --North
169     n_direction => n_direction,
170     n_select  => n_select,
171
172     --South
173     s_direction => s_direction,
174     s_select  => s_select,
175
176     --East
177     e_direction => e_direction,
178     e_select  => e_select,
```

```
179
180     --West
181     w_direction => w_direction,
182     w_select    => w_select,
183
184     --IP
185     i_direction => i_direction,
186     i_select    => i_select
187
188   );
189
190
191   end behav;
```

A.2.8 mux_tb.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Arbiter test bench
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 10/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity mux_tb is
22
23 end mux_tb;
24
25 architecture behav of mux_tb is
26
27   component noc_router_mux
28   port(
29     --I/O ports
30     clk_i    : in bit_t;
31     rst_i    : in bit_t;
32
33     -- Source selector
34     select_north_i  : in source;
```

```
35    select_south_i  : in source;
36    select_east_i : in source;
37    select_west_i : in source;
38    select_ip_i : in source;
39
40    --data in
41    north_i    : in noc_data;
42    south_i    : in noc_data;
43    east_i     : in noc_data;
44    west_i     : in noc_data;
45    ip_i       : in noc_data;
46
47    --data out
48    north_o    : out noc_data;
49    south_o    : out noc_data;
50    east_o     : out noc_data;
51    west_o     : out noc_data;
52    ip_o       : out noc_data
53       );
54  end component;
55
56    --I/O ports
57  SIGNAL  clk    : bit_t;
58  SIGNAL  rst    : bit_t;
59
60  -- Source selector
61  SIGNAL  select_north_i  : source;
62  SIGNAL  select_south_i  : source;
63  SIGNAL  select_east_i : source;
64  SIGNAL  select_west_i : source;
65  SIGNAL  select_ip_i : source;
66
67    --data in
68  SIGNAL  north_i    : noc_data;
69  SIGNAL  south_i    : noc_data;
70  SIGNAL  east_i     : noc_data;
```

```
71  SIGNAL  west_i    : noc_data;
72  SIGNAL  ip_i      : noc_data;
73
74    --data out
75  SIGNAL  north_o   : noc_data;
76  SIGNAL  south_o   : noc_data;
77  SIGNAL  east_o    : noc_data;
78  SIGNAL  west_o    : noc_data;
79  SIGNAL  ip_o      : noc_data;
80
81  SIGNAL state      : std_logic_vector(3 downto 0);
82  SIGNAL next_state : std_logic_vector(3 downto 0);
83
84  --Expect
85  SIGNAL n_exp      : noc_data;
86  SIGNAL s_exp      : noc_data;
87  SIGNAL e_exp      : noc_data;
88  SIGNAL w_exp      : noc_data;
89  SIGNAL i_exp      : noc_data;
90
91  begin
92
93  clock: process
94  begin  -- process clock
95    clk <= '0';
96    wait for 5 ns;
97    clk <= '1';
98    wait for 5 ns;
99  end process clock;
100
101 process
102 begin  -- process
103   rst <= '0';
104   wait for 12 ns;
105   rst <= '1';
106   wait;
```

```
107   end process;
108
109   next_state_register : process(clk, rst)
110   begin
111     if rst = '0' then -- asynchronous reset (active low=
112       state <= "0000";
113     elsif clk'event and clk = '1' then
114       state <= next_state;
115     end if;
116   end process next_state_register;
117
118   test :  process(state)
119   begin
120     case(state) is
121       when "0000" =>  --From ip
122         next_state  <= "0001";
123         --NoC signals
124         --North
125         select_north_i  <= source_ip;
126         n_exp   <= ip_i;
127
128         --South
129         select_south_i  <= source_north;
130         s_exp   <= north_i;
131
132         --East
133         select_east_i <= source_south;
134         e_exp   <= south_i;
135
136         --West
137         select_west_i <= source_east;
138         w_exp   <= east_i;
139
140         --IP
141         select_ip_i <= source_west;
142         i_exp   <= west_i;
```

```
143
144         when "0001" =>  -- From south
145           next_state  <= "0010";
146           --NoC signals
147           --North
148           select_north_i  <= source_west;
149           n_exp   <= west_i;
150
151           --South
152           select_south_i  <= source_ip;
153           s_exp   <= ip_i;
154
155           --East
156           select_east_i <= source_north;
157           e_exp   <= north_i;
158
159           --West
160           select_west_i <= source_south;
161           w_exp   <= south_i;
162
163           --IP
164           select_ip_i <= source_east;
165           i_exp   <= east_i;
166
167         when "0010" =>  -- From east
168           next_state  <= "0011";
169           --NoC signals
170           --North
171           select_north_i  <= source_east;
172           n_exp   <= east_i;
173
174           --South
175           select_south_i  <= source_west;
176           s_exp   <= west_i;
177
178           --East
```

```
179           select_east_i <= source_ip;
180           e_exp    <= ip_i;
181
182           --West
183           select_west_i <= source_north;
184           w_exp    <= north_i;
185
186           --IP
187           select_ip_i <= source_south;
188           i_exp    <= south_i;
189
190         when "0011" =>  -- From west
191           next_state  <= "0100";
192           --NoC signals
193           --North
194           select_north_i  <= source_south;
195           n_exp    <= south_i;
196
197           --South
198           select_south_i  <= source_east;
199           s_exp    <= east_i;
200
201           --East
202           select_east_i <= source_west;
203           e_exp    <= west_i;
204
205           --West
206           select_west_i <= source_ip;
207           w_exp    <= ip_i;
208
209           --IP
210           select_ip_i <= source_north;
211           i_exp    <= north_i;
212
213         when others =>
214           next_state  <= "0000";
```

```
215       end case;
216     end process test;
217
218
219     mux : noc_router_mux
220       port map(
221         --I/O ports
222         clk_i   => clk,
223         rst_i   => rst,
224         -- Source selector
225         select_north_i  => select_north_i,
226         select_south_i  => select_south_i,
227         select_east_i => select_east_i,
228         select_west_i => select_west_i,
229         select_ip_i => select_ip_i,
230
231         --data in
232         north_i   => north_i,
233         south_i   => south_i,
234         east_i    => east_i,
235         west_i    => west_i,
236         ip_i    => ip_i,
237
238         --data out
239         north_o   => north_o,
240         south_o   => south_o,
241         east_o    => east_o,
242         west_o    => west_o,
243         ip_o    => ip_o
244
245     );
246
247       north_i   <=  CONV_STD_LOGIC_VECTOR(0, NOC_DATA_WIDTH);
248       south_i   <=  CONV_STD_LOGIC_VECTOR(1, NOC_DATA_WIDTH);
249       east_i    <=  CONV_STD_LOGIC_VECTOR(2, NOC_DATA_WIDTH);
250       west_i(NOC_DATA_WIDTH-11 downto 0)    <=  CONV_STD_LOGIC_VECTOR(3, NOC_DATA_WIDTH-10);
```

```
251     ip_i    <=  CONV_STD_LOGIC_VECTOR(4, NOC_DATA_WIDTH);
252
253     west_i(NOC_DATA_WIDTH-1 downto NOC_DATA_WIDTH-11) <= "00100100010";
254
255
256   end behav;
```

A.2.9 router_tb.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Routerr test bench
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 11/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity router_tb is
22
23 end router_tb;
24
25 architecture behav of router_tb is
26
27   component noc_router
28   port(
29     --I/O ports
30     clk_i    : in bit_t;
31     rst_i    : in bit_t;
32
33     --NoC signals
34     --North
```

```
35      noc_n_i   : in noc_data;
36      noc_n_o   : out noc_data;
37
38      --South
39      noc_s_i   : in noc_data;
40      noc_s_o   : out noc_data;
41
42      --East
43      noc_e_i   : in noc_data;
44      noc_e_o   : out noc_data;
45
46      --West
47      noc_w_i   : in noc_data;
48      noc_w_o   : out noc_data;
49
50      --IP
51      ip_i      : in noc_data;
52      ip_o      : out noc_data
53          );
54  end component;
55
56
57  SIGNAL clk    : bit_t;
58  SIGNAL rst    : bit_t;
59
60  --NoC signals
61    --North
62  SIGNAL  noc_n_i   : noc_data;
63  SIGNAL  noc_n_o   : noc_data;
64
65    --South
66  SIGNAL  noc_s_i   : noc_data;
67  SIGNAL  noc_s_o   : noc_data;
68
69    --East
70  SIGNAL  noc_e_i   : noc_data;
```

```
71   SIGNAL  noc_e_o    : noc_data;
72
73     --West
74   SIGNAL  noc_w_i    : noc_data;
75   SIGNAL  noc_w_o    : noc_data;
76
77     --IP
78   SIGNAL  ip_i       : noc_data;
79   SIGNAL  ip_o       : noc_data;
80
81   SIGNAL state      : std_logic_vector(3 downto 0);
82   SIGNAL next_state : std_logic_vector(3 downto 0);
83
84   --Expect
85   SIGNAL n_exp      : noc_data;
86   SIGNAL s_exp      : noc_data;
87   SIGNAL e_exp      : noc_data;
88   SIGNAL w_exp      : noc_data;
89   SIGNAL i_exp      : noc_data;
90
91   SIGNAL almost1    : std_logic_vector(NOC_DATA_WIDTH-3 downto 0);
92   SIGNAL almost2    : std_logic_vector(NOC_DATA_WIDTH-3 downto 0);
93   SIGNAL almost3    : std_logic_vector(NOC_DATA_WIDTH-3 downto 0);
94   SIGNAL almost4    : std_logic_vector(NOC_DATA_WIDTH-3 downto 0);
95   SIGNAL almost5    : std_logic_vector(NOC_DATA_WIDTH-3 downto 0);
96
97 begin
98   almost1   <= CONV_STD_LOGIC_VECTOR(1, NOC_DATA_WIDTH-2);
99   almost2   <= CONV_STD_LOGIC_VECTOR(2, NOC_DATA_WIDTH-2);
100  almost3   <= CONV_STD_LOGIC_VECTOR(3, NOC_DATA_WIDTH-2);
101  almost4   <= CONV_STD_LOGIC_VECTOR(4, NOC_DATA_WIDTH-2);
102  almost5   <= CONV_STD_LOGIC_VECTOR(5, NOC_DATA_WIDTH-2);
103
104  clock: process
105  begin  -- process clock
106    clk <= '0';
```

```
107       wait for 5 ns;
108       clk <= '1';
109       wait for 5 ns;
110     end process clock;
111
112     process
113     begin  -- process
114       rst <= '0';
115       wait for 12 ns;
116       rst <= '1';
117       wait;
118     end process;
119
120     next_state_register : process(clk, rst)
121     begin
122       if rst = '0' then -- asynchronous reset (active low=
123         state <= "0000";
124       elsif clk'event and clk = '1' then
125         state <= next_state;
126       end if;
127     end process next_state_register;
128
129     test :  process(state)
130     begin
131       case(state) is
132         when "0000" =>  --All north
133           next_state  <= "0001";
134           --NoC signals
135           --North
136           noc_n_i   <= dest_north & almost1;
137         --  n_exp   <= "111";
138           --South
139           noc_s_i   <= dest_north & almost2;
140         --  s_exp   <= source_none;
141           --East
142           noc_e_i   <= dest_north & almost3;
```

```
143   --  e_exp   <= source_none;
144     --West
145     noc_w_i   <= dest_north & almost4;
146   --  w_exp   <= source_none;
147     --IP
148     ip_i      <= dest_north & almost5;
149   --  i_exp   <= source_north;
150   when "0001" =>  -- Two and two
151     next_state  <= "0010";
152     --NoC signals
153     --North
154     noc_n_i   <= dest_west & almost1;
155   --  n_exp   <= "111";
156     --South
157     noc_s_i   <= dest_north & almost2;
158   --  s_exp   <= source_none;
159     --East
160     noc_e_i   <= dest_north & almost3;
161   --  e_exp   <= source_none;
162     --West
163     noc_w_i   <= dest_north & almost4;
164   --  w_exp   <= source_none;
165     --IP
166     ip_i      <= dest_north & almost5;
167   --  i_exp   <= source_north;
168   when "0010" =>  -- Two and two
169     next_state  <= "0011";
170     --NoC signals
171     --North
172     noc_n_i   <= dest_west & almost1;
173   --  n_exp   <= "111";
174     --South
175     noc_s_i   <= dest_north & almost2;
176   --  s_exp   <= source_none;
177     --East
178     noc_e_i   <= dest_north & almost3;
```

```
179         --  e_exp   <= source_none & almost;
180           --West
181           noc_w_i   <= dest_north & almost4;
182         --  w_exp   <= source_none & almost;
183           --IP
184           ip_i    <= dest_north & almost5;
185         --  i_exp   <= source_north & almost;
186         when "0011" =>  -- Two and two
187           next_state  <= "0100";
188           --NoC signals
189           --North
190           noc_n_i   <= dest_north & almost1;
191         --  n_exp   <= "111";
192           --South
193           noc_s_i   <= dest_north & almost2;
194         --  s_exp   <= source_none;
195           --East
196           noc_e_i   <= dest_north & almost3;
197         --  e_exp   <= source_none;
198           --West
199           noc_w_i   <= dest_north & almost4;
200         --  w_exp   <= source_none;
201           --IP
202           ip_i    <= dest_north & almost5;
203         --  i_exp   <= source_north;
204         when "0100" =>  -- Two and two
205           next_state  <= "0101";
206           --NoC signals
207           --North
208           noc_n_i   <= dest_west & almost1;
209         --  n_exp   <= "111";
210           --South
211           noc_s_i   <= dest_north & almost2;
212         --  s_exp   <= source_none;
213           --East
214           noc_e_i   <= dest_north & almost3;
```

```
215         --  e_exp    <= source_none;
216           --West
217           noc_w_i    <= dest_north & almost4;
218         --  w_exp    <= source_none;
219           --IP
220           ip_i     <= dest_north & almost5;
221         --  i_exp    <= source_north;
222         when others =>
223           next_state  <= "0000";
224       end case;
225     end process test;
226
227
228     router : noc_router
229       port map(
230         --I/O ports
231         clk_i    => clk,
232         rst_i    => rst,
233
234         --NoC signals
235         --North
236         noc_n_i    => noc_n_i,
237         noc_n_o    => noc_n_o,
238         --South
239         noc_s_i    => noc_s_i,
240         noc_s_o    => noc_s_o,
241
242         --East
243         noc_e_i    => noc_e_i,
244         noc_e_o    => noc_e_o,
245
246         --West
247         noc_w_i    => noc_w_i,
248         noc_w_o    => noc_w_o,
249
250         --IP
```

```
251       ip_i   => ip_i,
252       ip_o   => ip_o
253
254     );
255
256   end behav;
257
```

A.3 Singel processor architectures

A.3.1 soc.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --      SoC design for multiprocessor
4  --      Editor: Nikolaj Tørring - s042505
5  --      Version: 1.3
6  --      Data last modified: 2/4-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --      Version History
11 --
12 --      v1.0: First working version
13 --      v1.1: Included UART
14 --      v1.2: Fixed running bit, fixed reset problem
15 --      v1.3: Fixed problem with data_slave not updating correct
16 --
17 ----------------------------------------------------------------------
18
19 library IEEE;
20 use IEEE.std_logic_1164.ALL;
21 use IEEE.std_logic_arith.ALL;
22 use work.types.all;
23 library UNISIM;
24 use UNISIM.all;
25
26 entity soc is
27
28   port (
29     clk_i : in std_logic;
30     rst_i : in std_logic;
```

```
31      tx  : out std_logic;
32      rx  : in std_logic;
33      running : out std_logic);
34
35  end soc;
36
37  architecture struc of soc is
38
39    component or1200_top
40      generic(
41        dw    : integer := 32;
42        aw    : integer := 32;
43        ppic_ints : integer := 20
44      );
45      port(
46        clk_i           : in    std_logic;
47        rst_i           : in    std_logic;
48        pic_ints_i      : in    std_logic_vector(ppic_ints-1 downto 0);
49        clmode_i        : in    std_logic_vector(1 downto 0);
50
51        -- Instruction WISHBONE interface
52        iwb_clk_i       : in    std_logic;
53        iwb_rst_i       : in    std_logic;
54        iwb_ack_i       : in    std_logic;
55        iwb_err_i       : in    std_logic;
56        iwb_rty_i       : in    std_logic;
57        iwb_dat_i       : in    std_logic_vector(31 downto 0);
58        iwb_cyc_o       : out   std_logic;
59        iwb_adr_o       : out   std_logic_vector(31 downto 0);
60        iwb_stb_o       : out   std_logic;
61        iwb_we_o        : out   std_logic;
62        iwb_sel_o       : out   std_logic_vector(3 downto 0);
63        iwb_dat_o       : out   std_logic_vector(31 downto 0);
64        iwb_cab_o       : out   std_logic;
65
66        -- Data WISHBONE interface
```

```
67  dwb_clk_i       : in   std_logic;
68  dwb_rst_i       : in   std_logic;
69  dwb_ack_i       : in   std_logic;
70  dwb_err_i       : in   std_logic;
71  dwb_rty_i       : in   std_logic;
72  dwb_dat_i       : in   std_logic_vector(31 downto 0);
73  dwb_cyc_o       : out  std_logic;
74  dwb_adr_o       : out  std_logic_vector(31 downto 0);
75  dwb_stb_o       : out  std_logic;
76  dwb_we_o        : out  std_logic;
77  dwb_sel_o       : out  std_logic_vector(3 downto 0);
78  dwb_dat_o       : out  std_logic_vector(31 downto 0);
79  dwb_cab_o       : out  std_logic;
80
81  -- Debug interface
82  dbg_stall_i     : in   std_logic;
83  dbg_ewt_i       : in   std_logic;
84  dbg_lss_o       : out  std_logic_vector(3 downto 0);
85  dbg_is_o        : out  std_logic_vector(1 downto 0);
86  dbg_wp_o        : out  std_logic_vector(10 downto 0);
87  dbg_bp_o        : out  std_logic;
88  dbg_stb_i       : in   std_logic;
89  dbg_we_i        : in   std_logic;
90  dbg_adr_i       : in   std_logic_vector(31 downto 0);
91  dbg_dat_i       : in   std_logic_vector(31 downto 0);
92  dbg_dat_o       : out  std_logic_vector(31 downto 0);
93  dbg_ack_o       : out  std_logic;
94
95  -- Power Management interface
96  pm_cpustall_i   : in   std_logic;
97  pm_clksd_o      : out  std_logic_vector(3 downto 0);
98  pm_dc_gate_o    : out  std_logic;
99  pm_ic_gate_o    : out  std_logic;
100 pm_dmmu_gate_o  : out  std_logic;
101 pm_immu_gate_o  : out  std_logic;
102 pm_tt_gate_o    : out  std_logic;
```

```
103       pm_cpu_gate_o   : out   std_logic;
104       pm_wakeup_o     : out   std_logic;
105       pm_lvolt_o      : out   std_logic
106     );
107   end component;
108
109   component memory
110     generic (
111       filename  : string := "";    -- memory contents
112       memsize   : natural := 8192);    -- memory size
113 --      membase   : natural := 0);  --membase
114     port (
115       --I/O Ports
116       wb_clk_i    : in bit_t;    -- The Clock
117       wb_rst_i    : in bit_t;    -- The reset signal
118
119       --WB slave i/f
120       wb_data_i   : in word_t;     -- WB data in
121       wb_data_o   : out word_t;    -- WB data out
122       wb_adr_i    : in word_t;     -- WB adress
123       wb_sel_i    : in std_logic_vector(3 downto 0); --WB select input array
124       wb_we_i     : in bit_t;    -- WB write enable
125       wb_cyc_i    : in bit_t;    -- WB cycle input
126       wb_stb_i    : in bit_t;    -- WB strobe input
127       wb_ack_o    : out bit_t;     -- WB acknowledge output
128       wb_rty_o    : out bit_t;     -- WB acknowledge output
129       wb_err_o    : out bit_t);    -- WB error signal
130   end component memory;
131
132   component core_mem
133     port (
134       --I/O Ports
135       wb_clk_i    : in bit_t;    -- The Clock
136       wb_rst_i    : in bit_t;    -- The reset signal
137
138       --WB slave i/f
```

```
139         wb_data_i    : in word_t;     -- WB data in
140         wb_data_o    : out word_t;    -- WB data out
141         wb_adr_i     : in word_t;     -- WB adress
142         wb_sel_i     : in std_logic_vector(3 downto 0); --WB select input array
143         wb_we_i      : in bit_t;    -- WB write enable
144         wb_cyc_i     : in bit_t;    -- WB cycle input
145         wb_stb_i     : in bit_t;    -- WB strobe input
146         wb_ack_o     : out bit_t;     -- WB acknowledge output
147         wb_err_o     : out bit_t);    -- WB error signal
148     end component core_mem;
149
150     component uart_top
151       port (
152         --I/O Ports
153         wb_clk_i     : in bit_t;    -- The Clock
154
155         --WB slave signal
156         wb_rst_i     : in bit_t;    -- The reset signal
157         wb_adr_i     : in std_logic_vector(4 downto 0) ;    -- WB adress
158         wb_dat_i     : in word_t;     -- WB data in
159         wb_dat_o     : out word_t;    -- WB data out
160         wb_we_i      : in bit_t;    -- WB write enable
161         wb_stb_i     : in bit_t;    -- WB strobe input
162         wb_cyc_i     : in bit_t;    -- WB cycle input
163         wb_ack_o     : out bit_t;     -- WB acknowledge output
164         wb_sel_i     : in std_logic_vector(3 downto 0); -- WB select input array
165
166         int_o      : out bit_t;       -- interrupt request
167
168         -- UART signals
169         -- serial input/output
170         stx_pad_o    : out bit_t;  -- Transmit data
171         srx_pad_i    : in bit_t; -- Recive Data
172
173         -- modem signals
174         rts_pad_o    : out bit_t;  -- Request to send
```

```
175       cts_pad_i    : in bit_t; -- Clear to send
176       dtr_pad_o    : out bit_t;  --
177       dsr_pad_i    : in bit_t; -- Data set ready
178       ri_pad_i     : in bit_t; -- Ring indicator
179       dcd_pad_i    : in bit_t  -- Data Carrier Detect
180     );
181
182   end component uart_top;
183
184   ------------------------------------------------------------------
185   --
186   --  ICON core component declaration
187   --
188   ------------------------------------------------------------------
189   component icon
190     port
191     (
192       control0    :   out std_logic_vector(35 downto 0);
193       control1    :   out std_logic_vector(35 downto 0)
194     );
195   end component;
196
197   ------------------------------------------------------------------
198   --
199   --  ILA core component declaration
200   --
201   ------------------------------------------------------------------
202   component ila
203     port
204     (
205       control     : in    std_logic_vector(35 downto 0);
206       clk         : in    std_logic;
207       data        : in    std_logic_vector(97 downto 0);
208       trig0       : in    std_logic_vector(31 downto 0)
209     );
210   end component;
```

```
211
212
213     ------------------------------------------------------------------
214     --
215     --  VIO core component declaration
216     --
217     ------------------------------------------------------------------
218     component vio
219       port
220       (
221         control     : in    std_logic_vector(35 downto 0);
222         async_out   :   out std_logic_vector(7 downto 0)
223       );
224     end component;
225
226     component BUFG
227       port (
228         O : out STD_ULOGIC;
229         I : in STD_ULOGIC);
230     end component;
231
232
233
234     --
235     --SIGNALS
236     --
237
238     --Clock signals
239   --  SIGNAL clk    : bit_t;
240     SIGNAL clk_div    : bit_t;
241     SIGNAL clk_12_5MHZ  : bit_t;
242     SIGNAL clkcount   : unsigned(31 downto 0) := "00000000000000000000000000000000";
243
244
245     SIGNAL reset    : bit_t;
246     SIGNAL reset_inv  : bit_t;
```

```
247  SIGNAL pic_ints   : std_logic_vector(19 downto 0);
248  SIGNAL clmode     : std_logic_vector(1 downto 0);
249
250  --Instruction signals
251  SIGNAL iwb_ack    : bit_t;
252  SIGNAL iwb_err    : bit_t;
253  SIGNAL iwb_rty    : bit_t;
254  SIGNAL iwb_dat_master : word_t;
255  SIGNAL iwb_cyc    : bit_t;
256  SIGNAL iwb_adr    : word_t;
257  SIGNAL iwb_stb    : bit_t;
258  SIGNAL iwb_we     : bit_t;
259  SIGNAL iwb_sel    : std_logic_vector(3 downto 0);
260  SIGNAL iwb_dat_slave  : word_t;
261  SIGNAL iwb_cab    : bit_t;
262
263  --Data signals
264  SIGNAL dwb_ack    : bit_t;
265  SIGNAL dwb_ack_mem  : bit_t;
266  SIGNAL dwb_ack_uart : bit_t;
267  SIGNAL dwb_err    : bit_t;
268  SIGNAL dwb_rty    : bit_t;
269  SIGNAL dwb_dat_master : word_t;
270  SIGNAL dwb_dat_master_m : word_t;
271  SIGNAL dwb_dat_master_u : word_t;
272  SIGNAL dwb_cyc    : bit_t;
273  SIGNAL dwb_adr    : word_t;
274  SIGNAL dwb_stb    : bit_t;
275  SIGNAL dwb_stb_mem  : bit_t;
276  SIGNAL dwb_stb_uart : bit_t;
277  SIGNAL dwb_we     : bit_t;
278  SIGNAL dwb_sel    : std_logic_vector(3 downto 0);
279  SIGNAL dwb_dat_mem  : word_t;
280  SIGNAL dwb_dat_uart : word_t;
281  SIGNAL dwb_dat_slave  : word_t;
282  SIGNAL dwb_cab    : bit_t;
```

```
283
284   --Zero signals
285   SIGNAL zero    : bit_t;
286   SIGNAL zero32  : word_t;
287
288   --Serial signal
289   SIGNAL tx_pad    : bit_t;
290
291   ------------------------------------------------------------------
292   --
293   --  ILA core signal declarations
294   --
295   ------------------------------------------------------------------
296   signal control0    : std_logic_vector(35 downto 0);
297   signal clk         : std_logic;
298   signal data        : std_logic_vector(97 downto 0);
299   signal trig0       : std_logic_vector(31 downto 0);
300
301   ------------------------------------------------------------------
302   --
303   --  VIO core signal declarations
304   --
305   ------------------------------------------------------------------
306   signal control1    : std_logic_vector(35 downto 0);
307   signal async_out   : std_logic_vector(7 downto 0);
308
309   ------------------------------------------------------------------
310   --
311   --  ICON core signal declarations
312   --
313   ------------------------------------------------------------------
314   --  signal control0       : std_logic_vector(35 downto 0);
315   --  signal control1       : std_logic_vector(35 downto 0);
316
317   begin
318     Clock_buf : BUFG
```

```
319     port map (O => clk_12_5MHZ,
320     I => clk_div);
321
322     process(clk_i)
323     begin
324       if clk_i'event and clk_i = '1' then
325         clkcount <= clkcount + 1;
326       end if;
327     end process;
328
329     clk_div <= conv_std_logic_vector(clkcount, 3)(2);
330     running <= conv_std_logic_vector(clkcount, 24)(23) and reset;
331
332     zero32      <= "00000000000000000000000000000000";
333     zero      <= '0';
334     pic_ints      <= "0000000000000000000";
335     clmode        <= "00";                      -- Same clock for WISHBONE and CPU
336
337
338     reset       <= rst_i;
339     reset_inv     <= not reset;
340
341
342     theCPU: or1200_top
343       PORT MAP(
344         clk_i   => clk_12_5MHZ,
345         rst_i   => reset_inv,
346         pic_ints_i  => pic_ints,
347         clmode_i  => clmode,
348
349         -- Instruction WISHBONE interface
350         iwb_clk_i => clk_12_5MHZ,
351         iwb_rst_i => reset_inv,
352         iwb_ack_i => iwb_ack,
353         iwb_err_i => iwb_err,
354         iwb_rty_i => iwb_rty,
```

```
355  iwb_dat_i => iwb_dat_slave,
356      iwb_cyc_o => iwb_cyc,
357  iwb_adr_o => iwb_adr,
358  iwb_stb_o => iwb_stb,
359  iwb_we_o  => iwb_we,
360  iwb_sel_o => iwb_sel,
361  iwb_dat_o => iwb_dat_master,
362  iwb_cab_o => iwb_cab,
363
364  -- Data WISHBONE interface
365  dwb_clk_i => clk_12_5MHZ,
366  dwb_rst_i => reset_inv,
367  dwb_ack_i => dwb_ack,
368  dwb_err_i => dwb_err,
369  dwb_rty_i => dwb_rty,
370  dwb_dat_i => dwb_dat_slave,
371  dwb_cyc_o => dwb_cyc,
372  dwb_adr_o => dwb_adr,
373  dwb_stb_o => dwb_stb,
374  dwb_we_o  => dwb_we,
375  dwb_sel_o => dwb_sel,
376  dwb_dat_o => dwb_dat_master,
377  dwb_cab_o => dwb_cab,
378
379  -- Debug interface
380  dbg_stall_i => zero,
381  dbg_ewt_i => zero,
382  dbg_lss_o => open,
383  dbg_is_o  => open,
384  dbg_wp_o  => open,
385  dbg_bp_o  => open,
386  dbg_stb_i => zero,
387  dbg_we_i  => zero,
388  dbg_adr_i => zero32,
389  dbg_dat_i => zero32,
390  dbg_dat_o => open,
```

```
391     dbg_ack_o => open,
392
393     -- Power Management interface
394     pm_cpustall_i => zero,
395     pm_clksd_o  => open,
396     pm_dc_gate_o  => open,
397     pm_ic_gate_o  => open,
398     pm_dmmu_gate_o  => open,
399     pm_immu_gate_o  => open,
400     pm_tt_gate_o  => open,
401     pm_cpu_gate_o => open,
402     pm_wakeup_o => open,
403     pm_lvolt_o  => open
404   );
405
406 --  imem: memory
407 --    generic map (
408 --      filename  => "nop.hex",
409 --      memsize   => 8192)
410   imem: core_mem
411     port map (
412       --I/O ports
413       wb_clk_i  => clk_12_5MHZ,
414       wb_rst_i  => reset,
415
416       --WB slave i/f
417       wb_data_i => iwb_dat_master,
418       wb_data_o => iwb_dat_slave,
419       wb_adr_i  => iwb_adr,
420       wb_sel_i  => iwb_sel,
421       wb_we_i   => iwb_we,
422       wb_cyc_i  => iwb_cyc,
423       wb_stb_i  => iwb_stb,
424       wb_ack_o  => iwb_ack,
425 --      wb_rty_o  => iwb_rty,
426       wb_err_o  => iwb_err
```

```
427     );
428  --  dmem: memory
429  --    generic map (
430  --       filename  => "nop.hex",
431  --       memsize   => 32768)
432       dmem: core_mem
433         port map (
434           --I/O ports
435           wb_clk_i  => clk_12_5MHZ,
436           wb_rst_i  => reset,
437
438           --WB slave i/f
439           wb_data_i => dwb_dat_master_m,
440           wb_data_o => dwb_dat_mem,
441           wb_adr_i  => dwb_adr,
442           wb_sel_i  => dwb_sel,
443           wb_we_i   => dwb_we,
444           wb_cyc_i  => dwb_cyc,
445           wb_stb_i  => dwb_stb_mem,
446           wb_ack_o  => dwb_ack_mem,
447  --         wb_rty    => dwb_rty,
448           wb_err_o  => dwb_err
449         );
450
451
452
453       uart : uart_top
454         port map (
455           --I/O Ports
456           wb_clk_i  => clk_12_5MHZ,
457           wb_rst_i  => reset_inv,
458
459           --WB slave i/f
460           wb_adr_i  => dwb_adr(4 downto 0),
461           wb_dat_i  => dwb_dat_master_u,
462           wb_dat_o  => dwb_dat_uart,
```

```
463         wb_we_i    => dwb_we,
464         wb_stb_i   => dwb_stb_uart,
465         wb_cyc_i   => dwb_cyc,
466         wb_ack_o   => dwb_ack_uart,
467         wb_sel_i   => dwb_sel,
468
469         --UART signals
470
471         int_o    => open,
472
473         --serial input/output
474         stx_pad_o => tx_pad,
475         srx_pad_i => rx,
476
477         --modem signals
478         rts_pad_o => open,
479         cts_pad_i => '1',
480         dtr_pad_o => open,
481         dsr_pad_i => '1',
482         ri_pad_i  => '1',
483         dcd_pad_i => '1'
484
485       );
486
487     switch_output: process (dwb_adr, dwb_stb, dwb_dat_master)
488     begin
489       if  dwb_adr(31 downto 28) = "1001" then
490         --uart address
491         dwb_stb_uart <= dwb_stb;
492         dwb_stb_mem <= '0';
493         dwb_dat_master_u <= dwb_dat_master;
494         dwb_dat_master_m <= (OTHERS => 'X');
495       else
496         --mem address
497         dwb_stb_mem <= dwb_stb;
498         dwb_stb_uart <= '0';
```

```
499     dwb_dat_master_m <= dwb_dat_master;
500     dwb_dat_master_u <= (OTHERS => 'X');
501   end if;
502 end process data_selector;
503
504 switch_input: process (dwb_ack_uart, dwb_ack_mem, dwb_dat_uart, dwb_dat_mem)
505 begin
506   if dwb_ack_uart = '1' then
507     dwb_ack <= dwb_ack_uart;
508     dwb_dat_slave <= dwb_dat_uart;
509   else
510     dwb_ack <= dwb_ack_mem;
511     dwb_dat_slave <= dwb_dat_mem;
512   end if;
513 end process arbiter;
514
515
516 ------------------------------------------------------------------------------
517 --
518 --  ICON core instance
519 --
520 ------------------------------------------------------------------------------
521 i_icon : icon
522   port map
523   (
524     control0    => control0,
525     control1    => control1
526   );
527
528 ------------------------------------------------------------------------------
529 --
530 --  ILA core instance
531 --
532 ------------------------------------------------------------------------------
533 i_ila : ila
534   port map
```

```
535     (
536       control   => control0,
537       clk       => clk_12_5MHZ,
538       data      => data,
539       trig0     => iwb_adr
540     );
541
542   ------------------------------------------------------------------
543   --
544   --  VIO core instance
545   --
546   ------------------------------------------------------------------
547   i_vio : vio
548     port map
549     (
550       control   => control1,
551       async_out => async_out
552     );
553
554     data <= "00000000000000000000000000000000" & dwb_adr & dwb_dat_master &  dwb_ack & dwb_stb & tx_pad &
          rx;
555     tx <= tx_pad;
556 end struc;
```

A.4 Bus architecture

A.4.1 bus_soc_tb.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Test bench for syntisisble SoC
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 27/3-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity bus_soc_tb is
22
23
24 end bus_soc_tb;
25
26 architecture struc of bus_soc_tb is
27
28   component soc_bus
29     port(
30       clk_i : in bit_t;
```

```
31      rst_i : in bit_t;
32      tx  : out std_logic;          -- RS232 TX
33      rx  : in std_logic;           -- RS232 RX
34      running : out bit_t
35    );
36    end component;
37
38    SIGNAL clk    : bit_t;
39    SIGNAL reset  : bit_t;
40    SIGNAL tx     : bit_t;
41    SIGNAL rx     : bit_t;
42    SIGNAL running    : bit_t;
43
44  begin
45
46    thesystem:soc_bus
47      port map(
48        clk_i    => clk,
49        rst_i    => reset,
50        tx  => tx,
51        rx  => '1',
52        running => running
53      );
54
55
56    process
57    begin   --process
58      reset <= '0';
59      wait for 105 ns;
60      reset <= '1';
61      wait for 1005 ns;
62      reset <= '0';
63      wait for 105 ns;
64      reset <= '1';
65      wait;
66    end process;
```

```
67
68    clock: process
69    begin  -- process clock
70      clk <= '0';
71      wait for 5 ns;
72      clk <= '1';
73      wait for 5 ns;
74    end process clock;
75
76
77
78
79    end struc;
```

A.4.2 conbus_soc.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       SoC design for multiprocessor
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 25/4-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20 library UNISIM;
21 use UNISIM.all;
22
23 entity soc_bus is
24
25   port (
26     clk_i : in std_logic;
27     rst_i : in std_logic;
28     tx  : out std_logic;
29     rx  : in std_logic;
30     running : out std_logic);
31
32 end soc_bus;
33
34 architecture struc of soc_bus is
```

```
35
36  component or1200_top
37    generic(
38      dw    : integer := 32;
39      aw    : integer := 32;
40      ppic_ints : integer := 20
41    );
42    port(
43      clk_i          : in    std_logic;
44      rst_i          : in    std_logic;
45      pic_ints_i     : in    std_logic_vector(ppic_ints-1 downto 0);
46      clmode_i       : in    std_logic_vector(1 downto 0);
47
48      -- Instruction WISHBONE interface
49      iwb_clk_i      : in    std_logic;
50      iwb_rst_i      : in    std_logic;
51      iwb_ack_i      : in    std_logic;
52      iwb_err_i      : in    std_logic;
53      iwb_rty_i      : in    std_logic;
54      iwb_dat_i      : in    std_logic_vector(31 downto 0);
55      iwb_cyc_o      : out   std_logic;
56      iwb_adr_o      : out   std_logic_vector(31 downto 0);
57      iwb_stb_o      : out   std_logic;
58      iwb_we_o       : out   std_logic;
59      iwb_sel_o      : out   std_logic_vector(3 downto 0);
60      iwb_dat_o      : out   std_logic_vector(31 downto 0);
61      iwb_cab_o      : out   std_logic;
62
63      -- Data WISHBONE interface
64      dwb_clk_i      : in    std_logic;
65      dwb_rst_i      : in    std_logic;
66      dwb_ack_i      : in    std_logic;
67      dwb_err_i      : in    std_logic;
68      dwb_rty_i      : in    std_logic;
69      dwb_dat_i      : in    std_logic_vector(31 downto 0);
70      dwb_cyc_o      : out   std_logic;
```

```
71        dwb_adr_o       : out   std_logic_vector(31 downto 0);
72        dwb_stb_o       : out   std_logic;
73        dwb_we_o        : out   std_logic;
74        dwb_sel_o       : out   std_logic_vector(3 downto 0);
75        dwb_dat_o       : out   std_logic_vector(31 downto 0);
76        dwb_cab_o       : out   std_logic;
77
78        -- Debug interface
79        dbg_stall_i     : in    std_logic;
80        dbg_ewt_i       : in    std_logic;
81        dbg_lss_o       : out   std_logic_vector(3 downto 0);
82        dbg_is_o        : out   std_logic_vector(1 downto 0);
83        dbg_wp_o        : out   std_logic_vector(10 downto 0);
84        dbg_bp_o        : out   std_logic;
85        dbg_stb_i       : in    std_logic;
86        dbg_we_i        : in    std_logic;
87        dbg_adr_i       : in    std_logic_vector(31 downto 0);
88        dbg_dat_i       : in  std_logic_vector(31 downto 0);
89        dbg_dat_o       : out std_logic_vector(31 downto 0);
90        dbg_ack_o       : out std_logic;
91
92        -- Power Management interface
93        pm_cpustall_i   : in    std_logic;
94        pm_clksd_o      : out   std_logic_vector(3 downto 0);
95        pm_dc_gate_o    : out   std_logic;
96        pm_ic_gate_o    : out   std_logic;
97        pm_dmmu_gate_o  : out   std_logic;
98        pm_immu_gate_o  : out   std_logic;
99        pm_tt_gate_o    : out   std_logic;
100       pm_cpu_gate_o   : out   std_logic;
101       pm_wakeup_o     : out   std_logic;
102       pm_lvolt_o      : out   std_logic
103     );
104   end component;
105
106   component core_mem
```

```
107     port (
108       --I/O Ports
109       wb_clk_i    : in bit_t;   -- The Clock
110       wb_rst_i    : in bit_t;   -- The reset signal
111
112       --WB slave i/f
113       wb_data_i   : in word_t;    -- WB data in
114       wb_data_o   : out word_t;   -- WB data out
115       wb_adr_i    : in word_t;    -- WB adress
116       wb_sel_i    : in std_logic_vector(3 downto 0); --WB select input array
117       wb_we_i     : in bit_t;   -- WB write enable
118       wb_cyc_i    : in bit_t;   -- WB cycle input
119       wb_stb_i    : in bit_t;   -- WB strobe input
120       wb_ack_o    : out bit_t;    -- WB acknowledge output
121       wb_err_o    : out bit_t);   -- WB error signal
122   end component core_mem;
123
124   component semaphore
125     port (
126       --I/O Ports
127       wb_clk_i    : in bit_t;   -- The Clock
128       wb_rst_i    : in bit_t;   -- The reset signal
129
130       --WB slave i/f
131       wb_data_i   : in word_t;    -- WB data in
132       wb_data_o   : out word_t;   -- WB data out
133       wb_adr_i    : in word_t;    -- WB adress
134       wb_sel_i    : in std_logic_vector(3 downto 0); --WB select input array
135       wb_we_i     : in bit_t;   -- WB write enable
136       wb_cyc_i    : in bit_t;   -- WB cycle input
137       wb_stb_i    : in bit_t;   -- WB strobe input
138       wb_ack_o    : out bit_t;    -- WB acknowledge output
139       wb_err_o    : out bit_t);   -- WB error signal
140   end component semaphore;
141
142   component uart_top
```

```
143     port (
144       --I/O Ports
145       wb_clk_i    : in bit_t;   -- The Clock
146
147       --WB slave signal
148       wb_rst_i    : in bit_t;   -- The reset signal
149       wb_adr_i    : in std_logic_vector(4 downto 0) ;   -- WB adress
150       wb_dat_i    : in word_t;    -- WB data in
151       wb_dat_o    : out word_t;   -- WB data out
152       wb_we_i     : in bit_t;   -- WB write enable
153       wb_stb_i    : in bit_t;   -- WB strobe input
154       wb_cyc_i    : in bit_t;   -- WB cycle input
155       wb_ack_o    : out bit_t;    -- WB acknowledge output
156       wb_sel_i    : in std_logic_vector(3 downto 0); -- WB select input array
157
158       int_o     : out bit_t;      -- interrupt request
159
160       -- UART signals
161       -- serial input/output
162       stx_pad_o   : out bit_t;  -- Transmit data
163       srx_pad_i   : in bit_t; -- Recive Data
164
165       -- modem signals
166       rts_pad_o   : out bit_t;  -- Request to send
167       cts_pad_i   : in bit_t; -- Clear to send
168       dtr_pad_o   : out bit_t;  --
169       dsr_pad_i   : in bit_t; -- Data set ready
170       ri_pad_i    : in bit_t; -- Ring indicator
171       dcd_pad_i   : in bit_t  -- Data Carrier Detect
172     );
173
174   end component uart_top;
175
176   component wb_conbus_top
177     generic(
178     --Imem 0x00000000
```

```
179     s0_addr_w       : integer := 17;
180 --    s0_addr         : integer := 0;
181     s0_addr         : integer := 0;
182     --Dmem 0x00004000
183     s1_addr_w       : integer := 17;
184 --    s1_addr         : integer := 8192;
185     s1_addr         : integer := 1;
186     s27_addr_w      : integer := 8;
187     --UART 0x90000000
188 --    s2_addr         : integer := -1879048192;
189     s2_addr         : integer := 144;
190     --Semaphore 0x40000000
191 --    s3_addr         : integer := 1073741824;
192     s3_addr         : integer := 64;
193     --Not used
194     s4_addr         : integer := 27;
195     s5_addr         : integer := 28;
196     s6_addr         : integer := 29;
197     s7_addr         : integer := 30
198   );
199   port(
200
201     --I/O ports
202     clk_i           : in     std_logic;
203     rst_i           : in     std_logic;
204
205     --Master 0
206     m0_dat_i        : in     std_logic_vector(31 downto 0);
207     m0_dat_o        : out    std_logic_vector(31 downto 0);
208     m0_adr_i        : in     std_logic_vector(31 downto 0);
209     m0_sel_i        : in     std_logic_vector(3 downto 0);
210     m0_we_i         : in     std_logic;
211     m0_cyc_i        : in     std_logic;
212     m0_stb_i        : in     std_logic;
213     m0_ack_o        : out    std_logic;
214     m0_err_o        : out    std_logic;
```

```
215     m0_rty_o        : out    std_logic;
216     m0_cab_i        : in     std_logic;
217
218     --Master 1
219     m1_dat_i        : in     std_logic_vector(31 downto 0);
220     m1_dat_o        : out    std_logic_vector(31 downto 0);
221     m1_adr_i        : in     std_logic_vector(31 downto 0);
222     m1_sel_i        : in     std_logic_vector(3 downto 0);
223     m1_we_i         : in     std_logic;
224     m1_cyc_i        : in     std_logic;
225     m1_stb_i        : in     std_logic;
226     m1_ack_o        : out    std_logic;
227     m1_err_o        : out    std_logic;
228     m1_rty_o        : out    std_logic;
229     m1_cab_i        : in     std_logic;
230
231     --Master 2
232     m2_dat_i        : in     std_logic_vector(31 downto 0);
233     m2_dat_o        : out    std_logic_vector(31 downto 0);
234     m2_adr_i        : in     std_logic_vector(31 downto 0);
235     m2_sel_i        : in     std_logic_vector(3 downto 0);
236     m2_we_i         : in     std_logic;
237     m2_cyc_i        : in     std_logic;
238     m2_stb_i        : in     std_logic;
239     m2_ack_o        : out    std_logic;
240     m2_err_o        : out    std_logic;
241     m2_rty_o        : out    std_logic;
242     m2_cab_i        : in     std_logic;
243
244     --Master 3
245     m3_dat_i        : in     std_logic_vector(31 downto 0);
246     m3_dat_o        : out    std_logic_vector(31 downto 0);
247     m3_adr_i        : in     std_logic_vector(31 downto 0);
248     m3_sel_i        : in     std_logic_vector(3 downto 0);
249     m3_we_i         : in     std_logic;
250     m3_cyc_i        : in     std_logic;
```

```
251  m3_stb_i     : in    std_logic;
252  m3_ack_o     : out   std_logic;
253  m3_err_o     : out   std_logic;
254  m3_rty_o     : out   std_logic;
255  m3_cab_i     : in    std_logic;
256
257  --Master 4
258  m4_dat_i     : in    std_logic_vector(31 downto 0);
259  m4_dat_o     : out   std_logic_vector(31 downto 0);
260  m4_adr_i     : in    std_logic_vector(31 downto 0);
261  m4_sel_i     : in    std_logic_vector(3 downto 0);
262  m4_we_i      : in    std_logic;
263  m4_cyc_i     : in    std_logic;
264  m4_stb_i     : in    std_logic;
265  m4_ack_o     : out   std_logic;
266  m4_err_o     : out   std_logic;
267  m4_rty_o     : out   std_logic;
268  m4_cab_i     : in    std_logic;
269
270  --Master 5
271  m5_dat_i     : in    std_logic_vector(31 downto 0);
272  m5_dat_o     : out   std_logic_vector(31 downto 0);
273  m5_adr_i     : in    std_logic_vector(31 downto 0);
274  m5_sel_i     : in    std_logic_vector(3 downto 0);
275  m5_we_i      : in    std_logic;
276  m5_cyc_i     : in    std_logic;
277  m5_stb_i     : in    std_logic;
278  m5_ack_o     : out   std_logic;
279  m5_err_o     : out   std_logic;
280  m5_rty_o     : out   std_logic;
281  m5_cab_i     : in    std_logic;
282
283  --Master 6
284  m6_dat_i     : in    std_logic_vector(31 downto 0);
285  m6_dat_o     : out   std_logic_vector(31 downto 0);
286  m6_adr_i     : in    std_logic_vector(31 downto 0);
```

```
287     m6_sel_i        : in     std_logic_vector(3 downto 0);
288     m6_we_i         : in     std_logic;
289     m6_cyc_i        : in     std_logic;
290     m6_stb_i        : in     std_logic;
291     m6_ack_o        : out    std_logic;
292     m6_err_o        : out    std_logic;
293     m6_rty_o        : out    std_logic;
294     m6_cab_i        : in     std_logic;
295
296     --Master 7
297     m7_dat_i        : in     std_logic_vector(31 downto 0);
298     m7_dat_o        : out    std_logic_vector(31 downto 0);
299     m7_adr_i        : in     std_logic_vector(31 downto 0);
300     m7_sel_i        : in     std_logic_vector(3 downto 0);
301     m7_we_i         : in     std_logic;
302     m7_cyc_i        : in     std_logic;
303     m7_stb_i        : in     std_logic;
304     m7_ack_o        : out    std_logic;
305     m7_err_o        : out    std_logic;
306     m7_rty_o        : out    std_logic;
307     m7_cab_i        : in     std_logic;
308
309     --Slave 0
310     s0_dat_i        : in     std_logic_vector(31 downto 0);
311     s0_dat_o        : out    std_logic_vector(31 downto 0);
312     s0_adr_o        : out    std_logic_vector(31 downto 0);
313     s0_sel_o        : out    std_logic_vector(3 downto 0);
314     s0_we_o         : out    std_logic;
315     s0_cyc_o        : out    std_logic;
316     s0_stb_o        : out    std_logic;
317     s0_ack_i        : in     std_logic;
318     s0_err_i        : in     std_logic;
319     s0_rty_i        : in     std_logic;
320     s0_cab_o        : out    std_logic;
321
322     --Slave 1
```

```
323   s1_dat_i    : in    std_logic_vector(31 downto 0);
324   s1_dat_o    : out   std_logic_vector(31 downto 0);
325   s1_adr_o    : out   std_logic_vector(31 downto 0);
326   s1_sel_o    : out   std_logic_vector(3 downto 0);
327   s1_we_o     : out   std_logic;
328   s1_cyc_o    : out   std_logic;
329   s1_stb_o    : out   std_logic;
330   s1_ack_i    : in    std_logic;
331   s1_err_i    : in    std_logic;
332   s1_rty_i    : in    std_logic;
333   s1_cab_o    : out   std_logic;
334
335   --Slave 2
336   s2_dat_i    : in    std_logic_vector(31 downto 0);
337   s2_dat_o    : out   std_logic_vector(31 downto 0);
338   s2_adr_o    : out   std_logic_vector(31 downto 0);
339   s2_sel_o    : out   std_logic_vector(3 downto 0);
340   s2_we_o     : out   std_logic;
341   s2_cyc_o    : out   std_logic;
342   s2_stb_o    : out   std_logic;
343   s2_ack_i    : in    std_logic;
344   s2_err_i    : in    std_logic;
345   s2_rty_i    : in    std_logic;
346   s2_cab_o    : out   std_logic;
347
348   --Slave 3
349   s3_dat_i    : in    std_logic_vector(31 downto 0);
350   s3_dat_o    : out   std_logic_vector(31 downto 0);
351   s3_adr_o    : out   std_logic_vector(31 downto 0);
352   s3_sel_o    : out   std_logic_vector(3 downto 0);
353   s3_we_o     : out   std_logic;
354   s3_cyc_o    : out   std_logic;
355   s3_stb_o    : out   std_logic;
356   s3_ack_i    : in    std_logic;
357   s3_err_i    : in    std_logic;
358   s3_rty_i    : in    std_logic;
```

```
359       s3_cab_o         : out    std_logic;
360
361       --Slave 4
362       s4_dat_i         : in     std_logic_vector(31 downto 0);
363       s4_dat_o         : out    std_logic_vector(31 downto 0);
364       s4_adr_o         : out    std_logic_vector(31 downto 0);
365       s4_sel_o         : out    std_logic_vector(3 downto 0);
366       s4_we_o          : out    std_logic;
367       s4_cyc_o         : out    std_logic;
368       s4_stb_o         : out    std_logic;
369       s4_ack_i         : in     std_logic;
370       s4_err_i         : in     std_logic;
371       s4_rty_i         : in     std_logic;
372       s4_cab_o         : out    std_logic;
373
374       --Slave 5
375       s5_dat_i         : in     std_logic_vector(31 downto 0);
376       s5_dat_o         : out    std_logic_vector(31 downto 0);
377       s5_adr_o         : out    std_logic_vector(31 downto 0);
378       s5_sel_o         : out    std_logic_vector(3 downto 0);
379       s5_we_o          : out    std_logic;
380       s5_cyc_o         : out    std_logic;
381       s5_stb_o         : out    std_logic;
382       s5_ack_i         : in     std_logic;
383       s5_err_i         : in     std_logic;
384       s5_rty_i         : in     std_logic;
385       s5_cab_o         : out    std_logic;
386
387       --Slave 6
388       s6_dat_i         : in     std_logic_vector(31 downto 0);
389       s6_dat_o         : out    std_logic_vector(31 downto 0);
390       s6_adr_o         : out    std_logic_vector(31 downto 0);
391       s6_sel_o         : out    std_logic_vector(3 downto 0);
392       s6_we_o          : out    std_logic;
393       s6_cyc_o         : out    std_logic;
394       s6_stb_o         : out    std_logic;
```

```
395       s6_ack_i        : in     std_logic;
396       s6_err_i        : in     std_logic;
397       s6_rty_i        : in     std_logic;
398       s6_cab_o        : out    std_logic;
399
400       --Slave 7
401       s7_dat_i        : in     std_logic_vector(31 downto 0);
402       s7_dat_o        : out    std_logic_vector(31 downto 0);
403       s7_adr_o        : out    std_logic_vector(31 downto 0);
404       s7_sel_o        : out    std_logic_vector(3 downto 0);
405       s7_we_o         : out    std_logic;
406       s7_cyc_o        : out    std_logic;
407       s7_stb_o        : out    std_logic;
408       s7_ack_i        : in     std_logic;
409       s7_err_i        : in     std_logic;
410       s7_rty_i        : in     std_logic;
411       s7_cab_o        : out    std_logic
412       );
413     end component;
414
415     ------------------------------------------------------------------
416     --
417     --  ICON core component declaration
418     --
419     ------------------------------------------------------------------
420     component icon
421       port
422       (
423         control0    :   out std_logic_vector(35 downto 0);
424         control1    :   out std_logic_vector(35 downto 0)
425       );
426     end component;
427
428     ------------------------------------------------------------------
429     --
430     --  ILA core component declaration
```

```
431   --
432   ------------------------------------------------------------------
433   component ila
434     port
435     (
436       control     : in    std_logic_vector(35 downto 0);
437       clk         : in    std_logic;
438       data        : in    std_logic_vector(97 downto 0);
439       trig0       : in    std_logic_vector(31 downto 0)
440     );
441   end component;
442
443
444   ------------------------------------------------------------------
445   --
446   --  VIO core component declaration
447   --
448   ------------------------------------------------------------------
449   component vio
450     port
451     (
452       control     : in    std_logic_vector(35 downto 0);
453       async_out   :   out std_logic_vector(7 downto 0)
454     );
455   end component;
456
457   component BUFG
458     port (
459       O : out STD_ULOGIC;
460       I : in STD_ULOGIC);
461   end component;
462
463   component IBUFG
464     port (
465       O : out STD_ULOGIC;
466       I : in STD_ULOGIC);
```

```
467     end component;
468
469
470
471     --
472     --SIGNALS
473     --
474
475     --Clock signals
476   --  SIGNAL clk    : bit_t;
477     SIGNAL clk_div    : bit_t;
478     SIGNAL clk_12_5MHZ  : bit_t;
479     SIGNAL clkcount   : unsigned(31 downto 0) := "00000000000000000000000000000000";
480
481
482     SIGNAL reset    : bit_t;
483     SIGNAL reset_inv  : bit_t;
484     SIGNAL pic_ints   : std_logic_vector(19 downto 0);
485     SIGNAL clmode   : std_logic_vector(1 downto 0);
486
487     --Instruction signals
488   --  SIGNAL iwb_ack    : bit_t;
489   --  SIGNAL iwb_err    : bit_t;
490   --  SIGNAL iwb_rty    : bit_t;
491   --  SIGNAL iwb_dat_master : word_t;
492   --  SIGNAL iwb_cyc    : bit_t;
493   --  SIGNAL iwb_adr    : word_t;
494   --  SIGNAL iwb_stb    : bit_t;
495   --  SIGNAL iwb_we   : bit_t;
496   --  SIGNAL iwb_sel    : std_logic_vector(3 downto 0);
497   --  SIGNAL iwb_dat_slave  : word_t;
498   --  SIGNAL iwb_cab    : bit_t;
499
500     --Data signals
501   --  SIGNAL dwb_ack    : bit_t;
502   --  SIGNAL dwb_ack_mem  : bit_t;
```

```
503  --SIGNAL dwb_ack_uart : bit_t;
504  --  SIGNAL dwb_err     : bit_t;
505  --  SIGNAL dwb_rty     : bit_t;
506  --  SIGNAL dwb_dat_master : word_t;
507  --  SIGNAL dwb_cyc     : bit_t;
508  --  SIGNAL dwb_adr     : word_t;
509  --  SIGNAL dwb_stb     : bit_t;
510  --  SIGNAL dwb_stb_mem  : bit_t;
511  --  SIGNAL dwb_stb_uart : bit_t;
512  --  SIGNAL dwb_we    : bit_t;
513  --  SIGNAL dwb_sel     : std_logic_vector(3 downto 0);
514  --  SIGNAL dwb_dat_mem  : word_t;
515  --  SIGNAL dwb_dat_uart : word_t;
516  --  SIGNAL dwb_dat_slave  : word_t;
517  --  SIGNAL dwb_cab     : bit_t;
518
519    -- Master 0 Interface
520    SIGNAL  m0_dat_i, m0_dat_o, m0_adr_o  : word_t;
521    SIGNAL  m0_sel_o  : std_logic_vector(3 downto 0);
522    SIGNAL  m0_we_o, m0_cyc_o,m0_stb_o, m0_ack_i, m0_err_i, m0_rty_i, m0_cab_o  : bit_t;
523
524    -- Master 1 Interface
525    SIGNAL  m1_dat_i, m1_dat_o, m1_adr_o  : word_t;
526    SIGNAL  m1_sel_o  : std_logic_vector(3 downto 0);
527    SIGNAL  m1_we_o, m1_cyc_o,m1_stb_o, m1_ack_i, m1_err_i, m1_rty_i, m1_cab_o  : bit_t;
528
529    -- Master 2 Interface
530    SIGNAL  m2_dat_i, m2_dat_o, m2_adr_o  : word_t;
531    SIGNAL  m2_sel_o  : std_logic_vector(3 downto 0);
532    SIGNAL  m2_we_o, m2_cyc_o,m2_stb_o, m2_ack_i, m2_err_i, m2_rty_i, m2_cab_o  : bit_t;
533
534    -- Master 3 Interface
535    SIGNAL  m3_dat_i, m3_dat_o, m3_adr_o  : word_t;
536    SIGNAL  m3_sel_o  : std_logic_vector(3 downto 0);
537    SIGNAL  m3_we_o, m3_cyc_o,m3_stb_o, m3_ack_i, m3_err_i, m3_rty_i, m3_cab_o  : bit_t;
538
```

```
539     -- Master 4 Interface
540     SIGNAL  m4_dat_i, m4_dat_o, m4_adr_o  : word_t;
541     SIGNAL  m4_sel_o  : std_logic_vector(3 downto 0);
542     SIGNAL  m4_we_o, m4_cyc_o,m4_stb_o, m4_ack_i, m4_err_i, m4_rty_i, m4_cab_o  : bit_t;
543
544     -- Master 5 Interface
545     SIGNAL  m5_dat_i, m5_dat_o, m5_adr_o  : word_t;
546     SIGNAL  m5_sel_o  : std_logic_vector(3 downto 0);
547     SIGNAL  m5_we_o, m5_cyc_o,m5_stb_o, m5_ack_i, m5_err_i, m5_rty_i, m5_cab_o  : bit_t;
548
549     -- Master 6 Interface
550     SIGNAL  m6_dat_i, m6_dat_o, m6_adr_o  : word_t;
551     SIGNAL  m6_sel_o  : std_logic_vector(3 downto 0);
552     SIGNAL  m6_we_o, m6_cyc_o,m6_stb_o, m6_ack_i, m6_err_i, m6_rty_i, m6_cab_o  : bit_t;
553
554     -- Master 7 Interface
555     SIGNAL  m7_dat_i, m7_dat_o, m7_adr_o  : word_t;
556     SIGNAL  m7_sel_o  : std_logic_vector(3 downto 0);
557     SIGNAL  m7_we_o, m7_cyc_o,m7_stb_o, m7_ack_i, m7_err_i, m7_rty_i, m7_cab_o  : bit_t;
558
559     -- Slaver 0 Interface
560     SIGNAL  s0_dat_i, s0_dat_o, s0_adr_i :word_t;
561     SIGNAL  s0_sel_i  : std_logic_vector(3 downto 0);
562     SIGNAL  s0_we_i, s0_cyc_i,s0_stb_i, s0_ack_o, s0_err_o, s0_rty_o, s0_cab_i  : bit_t;
563
564     -- Slaver 1 Interface
565     SIGNAL  s1_dat_i, s1_dat_o, s1_adr_i :word_t;
566     SIGNAL  s1_sel_i  : std_logic_vector(3 downto 0);
567     SIGNAL  s1_we_i, s1_cyc_i,s1_stb_i, s1_ack_o, s1_err_o, s1_rty_o, s1_cab_i  : bit_t;
568
569     -- Slaver 2 Interface
570     SIGNAL  s2_dat_i, s2_dat_o, s2_adr_i :word_t;
571     SIGNAL  s2_sel_i  : std_logic_vector(3 downto 0);
572     SIGNAL  s2_we_i, s2_cyc_i,s2_stb_i, s2_ack_o, s2_err_o, s2_rty_o, s2_cab_i  : bit_t;
573
574     -- Slaver 3 Interface
```

```
575     SIGNAL  s3_dat_i, s3_dat_o, s3_adr_i :word_t;
576     SIGNAL  s3_sel_i  : std_logic_vector(3 downto 0);
577     SIGNAL  s3_we_i, s3_cyc_i,s3_stb_i, s3_ack_o, s3_err_o, s3_rty_o, s3_cab_i  : bit_t;
578
579     -- Slaver 4 Interface
580     SIGNAL  s4_dat_i, s4_dat_o, s4_adr_i :word_t;
581     SIGNAL  s4_sel_i  : std_logic_vector(3 downto 0);
582     SIGNAL  s4_we_i, s4_cyc_i,s4_stb_i, s4_ack_o, s4_err_o, s4_rty_o, s4_cab_i  : bit_t;
583
584     -- Slaver 5 Interface
585     SIGNAL  s5_dat_i, s5_dat_o, s5_adr_i :word_t;
586     SIGNAL  s5_sel_i  : std_logic_vector(3 downto 0);
587     SIGNAL  s5_we_i, s5_cyc_i,s5_stb_i, s5_ack_o, s5_err_o, s5_rty_o, s5_cab_i  : bit_t;
588
589     -- Slaver 6 Interface
590     SIGNAL  s6_dat_i, s6_dat_o, s6_adr_i :word_t;
591     SIGNAL  s6_sel_i  : std_logic_vector(3 downto 0);
592     SIGNAL  s6_we_i, s6_cyc_i,s6_stb_i, s6_ack_o, s6_err_o, s6_rty_o, s6_cab_i  : bit_t;
593
594     -- Slaver 7 Interface
595     SIGNAL  s7_dat_i, s7_dat_o, s7_adr_i :word_t;
596     SIGNAL  s7_sel_i  : std_logic_vector(3 downto 0);
597     SIGNAL  s7_we_i, s7_cyc_i,s7_stb_i, s7_ack_o, s7_err_o, s7_rty_o, s7_cab_i  : bit_t;
598
599     --ack signals
600     SIGNAL ack0 : bit_t;
601     SIGNAL ack1 : bit_t;
602     SIGNAL ack2 : bit_t;
603     SIGNAL ack3 : bit_t;
604     SIGNAL ack4 : bit_t;
605     SIGNAL ack5 : bit_t;
606     SIGNAL ack6 : bit_t;
607     SIGNAL ack7 : bit_t;
608
609
610
```

```
611     --Zero signals
612     SIGNAL zero   : bit_t;
613     SIGNAL zero32   : word_t;
614
615     --Serial signal
616     SIGNAL tx_pad   : bit_t;
617
618     ------------------------------------------------------------------
619     --
620     --  ILA core signal declarations
621     --
622     ------------------------------------------------------------------
623     signal control0    : std_logic_vector(35 downto 0);
624     signal clk         : std_logic;
625     signal data        : std_logic_vector(97 downto 0);
626     signal trig0       : std_logic_vector(31 downto 0);
627
628     ------------------------------------------------------------------
629     --
630     --  VIO core signal declarations
631     --
632     ------------------------------------------------------------------
633     signal control1    : std_logic_vector(35 downto 0);
634     signal async_out   : std_logic_vector(7 downto 0);
635
636     ------------------------------------------------------------------
637     --
638     --  ICON core signal declarations
639     --
640     ------------------------------------------------------------------
641   --  signal control0       : std_logic_vector(35 downto 0);
642   --  signal control1       : std_logic_vector(35 downto 0);
643
644
645   begin
646     Clock_buf : BUFG
```

```
647   port map (O => clk_12_5MHZ,
648   I => clk_div);
649
650   rst_buf: IBUFG
651   port map (O => reset,
652   I => rst_i);
653
654   process(clk_i)
655   begin
656     if clk_i'event and clk_i = '1' then
657       clkcount <= clkcount + 1;
658     end if;
659   end process;
660
661   clk_div <= conv_std_logic_vector(clkcount, 3)(2);
662   running <= conv_std_logic_vector(clkcount, 24)(23) and reset;
663
664   zero32      <= "00000000000000000000000000000000";
665   zero      <= '0';
666   pic_ints      <= "0000000000000000";
667   clmode        <= "00";                    -- Same clock for WISHBONE and CPU
668
669   reset_inv     <= not reset;
670
671
672   CPU0: or1200_top
673     PORT MAP(
674       clk_i   => clk_12_5MHZ,
675       rst_i   => reset_inv,
676       pic_ints_i  => pic_ints,
677       clmode_i  => clmode,
678
679       -- Instruction WISHBONE interface
680       iwb_clk_i => clk_12_5MHZ,
681       iwb_rst_i => reset_inv,
682       iwb_ack_i => ack0,
```

```
683   iwb_err_i => m0_err_i,
684   iwb_rty_i => m0_rty_i,
685   iwb_dat_i => m0_dat_i,
686      iwb_cyc_o => m0_cyc_o,
687   iwb_adr_o => m0_adr_o,
688   iwb_stb_o => m0_stb_o,
689   iwb_we_o  => m0_we_o,
690   iwb_sel_o => m0_sel_o,
691   iwb_dat_o => m0_dat_o,
692   iwb_cab_o => m0_cab_o,
693
694   -- Data WISHBONE interface
695   dwb_clk_i => clk_12_5MHZ,
696   dwb_rst_i => reset_inv,
697   dwb_ack_i => ack1,
698   dwb_err_i => m1_err_i,
699   dwb_rty_i => m1_rty_i,
700   dwb_dat_i => m1_dat_i,
701   dwb_cyc_o => m1_cyc_o,
702   dwb_adr_o => m1_adr_o,
703   dwb_stb_o => m1_stb_o,
704   dwb_we_o  => m1_we_o,
705   dwb_sel_o => m1_sel_o,
706   dwb_dat_o => m1_dat_o,
707   dwb_cab_o => m1_cab_o,
708
709   -- Debug interface
710   dbg_stall_i => zero,
711   dbg_ewt_i => zero,
712   dbg_lss_o => open,
713   dbg_is_o  => open,
714   dbg_wp_o  => open,
715   dbg_bp_o  => open,
716   dbg_stb_i => zero,
717   dbg_we_i  => zero,
718   dbg_adr_i => zero32,
```

```
719     dbg_dat_i => zero32,
720     dbg_dat_o => open,
721     dbg_ack_o => open,
722
723     -- Power Management interface
724     pm_cpustall_i => zero,
725     pm_clksd_o  => open,
726     pm_dc_gate_o  => open,
727     pm_ic_gate_o  => open,
728     pm_dmmu_gate_o  => open,
729     pm_immu_gate_o  => open,
730     pm_tt_gate_o  => open,
731     pm_cpu_gate_o => open,
732     pm_wakeup_o => open,
733     pm_lvolt_o  => open
734   );
735
736   CPU1: or1200_top
737     PORT MAP(
738       clk_i    => clk_12_5MHZ,
739       rst_i    => reset_inv,
740       pic_ints_i  => pic_ints,
741       clmode_i  => clmode,
742
743       -- Instruction WISHBONE interface
744       iwb_clk_i => clk_12_5MHZ,
745       iwb_rst_i => reset_inv,
746       iwb_ack_i => ack2,
747       iwb_err_i => m2_err_i,
748       iwb_rty_i => m2_rty_i,
749       iwb_dat_i => m2_dat_i,
750         iwb_cyc_o => m2_cyc_o,
751       iwb_adr_o => m2_adr_o,
752       iwb_stb_o => m2_stb_o,
753       iwb_we_o  => m2_we_o,
754       iwb_sel_o => m2_sel_o,
```

```
755  iwb_dat_o => m2_dat_o,
756  iwb_cab_o => m2_cab_o,
757
758  -- Data WISHBONE interface
759  dwb_clk_i => clk_12_5MHZ,
760  dwb_rst_i => reset_inv,
761  dwb_ack_i => ack3,
762  dwb_err_i => m3_err_i,
763  dwb_rty_i => m3_rty_i,
764  dwb_dat_i => m3_dat_i,
765  dwb_cyc_o => m3_cyc_o,
766  dwb_adr_o => m3_adr_o,
767  dwb_stb_o => m3_stb_o,
768  dwb_we_o  => m3_we_o,
769  dwb_sel_o => m3_sel_o,
770  dwb_dat_o => m3_dat_o,
771  dwb_cab_o => m3_cab_o,
772
773  -- Debug interface
774  dbg_stall_i => zero,
775  dbg_ewt_i => zero,
776  dbg_lss_o => open,
777  dbg_is_o  => open,
778  dbg_wp_o  => open,
779  dbg_bp_o  => open,
780  dbg_stb_i => zero,
781  dbg_we_i  => zero,
782  dbg_adr_i => zero32,
783  dbg_dat_i => zero32,
784  dbg_dat_o => open,
785  dbg_ack_o => open,
786
787  -- Power Management interface
788  pm_cpustall_i => zero,
789  pm_clksd_o  => open,
790  pm_dc_gate_o  => open,
```

```
791     pm_ic_gate_o  => open,
792     pm_dmmu_gate_o  => open,
793     pm_immu_gate_o  => open,
794     pm_tt_gate_o  => open,
795     pm_cpu_gate_o => open,
796     pm_wakeup_o => open,
797     pm_lvolt_o  => open
798   );
799
800   CPU2: or1200_top
801     PORT MAP(
802       clk_i    => clk_12_5MHZ,
803       rst_i    => reset_inv,
804       pic_ints_i  => pic_ints,
805       clmode_i   => clmode,
806
807       -- Instruction WISHBONE interface
808       iwb_clk_i => clk_12_5MHZ,
809       iwb_rst_i => reset_inv,
810       iwb_ack_i => ack4,
811       iwb_err_i => m4_err_i,
812       iwb_rty_i => m4_rty_i,
813       iwb_dat_i => m4_dat_i,
814           iwb_cyc_o => m4_cyc_o,
815       iwb_adr_o => m4_adr_o,
816       iwb_stb_o => m4_stb_o,
817       iwb_we_o  => m4_we_o,
818       iwb_sel_o => m4_sel_o,
819       iwb_dat_o => m4_dat_o,
820       iwb_cab_o => m4_cab_o,
821
822       -- Data WISHBONE interface
823       dwb_clk_i => clk_12_5MHZ,
824       dwb_rst_i => reset_inv,
825       dwb_ack_i => ack5,
826       dwb_err_i => m5_err_i,
```

```
827   dwb_rty_i => m5_rty_i,
828   dwb_dat_i => m5_dat_i,
829   dwb_cyc_o => m5_cyc_o,
830   dwb_adr_o => m5_adr_o,
831   dwb_stb_o => m5_stb_o,
832   dwb_we_o  => m5_we_o,
833   dwb_sel_o => m5_sel_o,
834   dwb_dat_o => m5_dat_o,
835   dwb_cab_o => m5_cab_o,
836
837   -- Debug interface
838   dbg_stall_i => zero,
839   dbg_ewt_i => zero,
840   dbg_lss_o => open,
841   dbg_is_o  => open,
842   dbg_wp_o  => open,
843   dbg_bp_o  => open,
844   dbg_stb_i => zero,
845   dbg_we_i  => zero,
846   dbg_adr_i => zero32,
847   dbg_dat_i => zero32,
848   dbg_dat_o => open,
849   dbg_ack_o => open,
850
851   -- Power Management interface
852   pm_cpustall_i => zero,
853   pm_clksd_o  => open,
854   pm_dc_gate_o  => open,
855   pm_ic_gate_o  => open,
856   pm_dmmu_gate_o  => open,
857   pm_immu_gate_o  => open,
858   pm_tt_gate_o  => open,
859   pm_cpu_gate_o => open,
860   pm_wakeup_o => open,
861   pm_lvolt_o  => open
862 );
```

```
863
864 --  CPU3: or1200_top
865 --    PORT MAP(
866 --      clk_i      => clk_12_5MHZ,
867 --      rst_i      => reset_inv,
868 --      pic_ints_i  => pic_ints,
869 --      clmode_i   => clmode,
870 --
871 --      -- Instruction WISHBONE interface
872 --      iwb_clk_i => clk_12_5MHZ,
873 --      iwb_rst_i => reset_inv,
874 --      iwb_ack_i => ack6,
875 --      iwb_err_i => m6_err_i,
876 --      iwb_rty_i => m6_rty_i,
877 --      iwb_dat_i => m6_dat_i,
878 --      iwb_cyc_o => m6_cyc_o,
879 --      iwb_adr_o => m6_adr_o,
880 --      iwb_stb_o => m6_stb_o,
881 --      iwb_we_o  => m6_we_o,
882 --      iwb_sel_o => m6_sel_o,
883 --      iwb_dat_o => m6_dat_o,
884 --      iwb_cab_o => m6_cab_o,
885 --
886 --      -- Data WISHBONE interface
887 --      dwb_clk_i => clk_12_5MHZ,
888 --      dwb_rst_i => reset_inv,
889 --      dwb_ack_i => ack7,
890 --      dwb_err_i => m7_err_i,
891 --      dwb_rty_i => m7_rty_i,
892 --      dwb_dat_i => m7_dat_i,
893 --      dwb_cyc_o => m7_cyc_o,
894 --      dwb_adr_o => m7_adr_o,
895 --      dwb_stb_o => m7_stb_o,
896 --      dwb_we_o  => m7_we_o,
897 --      dwb_sel_o => m7_sel_o,
898 --      dwb_dat_o => m7_dat_o,
```

```
899 --    dwb_cab_o => m7_cab_o,
900 --
901 --    -- Debug interface
902 --    dbg_stall_i => zero,
903 --    dbg_ewt_i => zero,
904 --    dbg_lss_o => open,
905 --    dbg_is_o  => open,
906 --    dbg_wp_o  => open,
907 --    dbg_bp_o  => open,
908 --    dbg_stb_i => zero,
909 --    dbg_we_i  => zero,
910 --    dbg_adr_i => zero32,
911 --    dbg_dat_i => zero32,
912 --    dbg_dat_o => open,
913 --    dbg_ack_o => open,
914 --
915 --    -- Power Management interface
916 --    pm_cpustall_i => zero,
917 --    pm_clksd_o  => open,
918 --    pm_dc_gate_o  => open,
919 --    pm_ic_gate_o  => open,
920 --    pm_dmmu_gate_o  => open,
921 --    pm_immu_gate_o  => open,
922 --    pm_tt_gate_o  => open,
923 --    pm_cpu_gate_o => open,
924 --    pm_wakeup_o => open,
925 --    pm_lvolt_o  => open
926 --    );
927
928     imem: core_mem
929       port map (
930         --I/O ports
931         wb_clk_i  => clk_12_5MHZ,
932         wb_rst_i  => reset,
933
934         --WB slave i/f
```

```
935         wb_data_i => s0_dat_i,
936         wb_data_o => s0_dat_o,
937         wb_adr_i  => s0_adr_i,
938         wb_sel_i  => s0_sel_i,
939         wb_we_i   => s0_we_i,
940         wb_cyc_i  => s0_cyc_i,
941         wb_stb_i  => s0_stb_i,
942         wb_ack_o  => s0_ack_o,
943         wb_err_o  => s0_err_o
944       );
945
946
947     dmem: core_mem
948       port map (
949         --I/O ports
950         wb_clk_i  => clk_12_5MHZ,
951         wb_rst_i  => reset,
952
953         --WB slave i/f
954         wb_data_i => s1_dat_i,
955         wb_data_o => s1_dat_o,
956         wb_adr_i  => s1_adr_i,
957         wb_sel_i  => s1_sel_i,
958         wb_we_i   => s1_we_i,
959         wb_cyc_i  => s1_cyc_i,
960         wb_stb_i  => s1_stb_i,
961         wb_ack_o  => s1_ack_o,
962         wb_err_o  => s1_err_o
963       );
964
965     sema: semaphore
966       port map (
967         --I/O ports
968         wb_clk_i  => clk_12_5MHZ,
969         wb_rst_i  => reset,
970
```

```
971   --WB slave i/f
972   wb_data_i => s3_dat_i,
973   wb_data_o => s3_dat_o,
974   wb_adr_i  => s3_adr_i,
975   wb_sel_i  => s3_sel_i,
976   wb_we_i   => s3_we_i,
977   wb_cyc_i  => s3_cyc_i,
978   wb_stb_i  => s3_stb_i,
979   wb_ack_o  => s3_ack_o,
980   wb_err_o  => s3_err_o
981 );
982
983
984 uart : uart_top
985   port map (
986     --I/O Ports
987     wb_clk_i  => clk_12_5MHZ,
988     wb_rst_i  => reset_inv,
989
990     --WB slave i/f
991     wb_adr_i  => s2_adr_i(4 downto 0),
992     wb_dat_i  => s2_dat_i,
993     wb_dat_o  => s2_dat_o,
994     wb_we_i   => s2_we_i,
995     wb_stb_i  => s2_stb_i,
996     wb_cyc_i  => s2_cyc_i,
997     wb_ack_o  => s2_ack_o,
998     wb_sel_i  => s2_sel_i,
999
1000    --UART signals
1001
1002    int_o     => open,
1003
1004    --serial input/output
1005    stx_pad_o => tx_pad,
1006    srx_pad_i => rx,
```

```
1007
1008        --modem signals
1009        rts_pad_o => open,
1010        cts_pad_i => '1',
1011        dtr_pad_o => open,
1012        dsr_pad_i => '1',
1013        ri_pad_i  => '1',
1014        dcd_pad_i => '1'
1015
1016      );
1017
1018
1019    conbus : wb_conbus_top
1020    port map (
1021
1022      --I/O ports
1023      clk_i   => clk_12_5MHZ,
1024      rst_i   => reset_inv, --active high
1025
1026      --Master 0
1027      m0_dat_i  => m0_dat_o,
1028      m0_dat_o  => m0_dat_i,
1029      m0_adr_i  => m0_adr_o,
1030      m0_sel_i  => m0_sel_o,
1031      m0_we_i   => m0_we_o,
1032      m0_cyc_i  => m0_cyc_o,
1033      m0_stb_i  => m0_stb_o,
1034      m0_ack_o  => m0_ack_i,
1035      m0_err_o  => open, -- m0_err_i,
1036      m0_rty_o  => open, --m0_rty_i,
1037      m0_cab_i  => m0_cab_o,
1038
1039      --Master 1
1040      m1_dat_i  => m1_dat_o,
1041      m1_dat_o  => m1_dat_i,
1042      m1_adr_i  => m1_adr_o,
```

```
1043    m1_sel_i  => m1_sel_o,
1044    m1_we_i   => m1_we_o,
1045    m1_cyc_i  => m1_cyc_o,
1046    m1_stb_i  => m1_stb_o,
1047    m1_ack_o  => m1_ack_i,
1048    m1_err_o  => open, --m1_err_i,
1049    m1_rty_o  => open, --m1_rty_i,
1050    m1_cab_i  => m1_cab_o,
1051
1052    --Master 2
1053    m2_dat_i  => m2_dat_o,
1054    m2_dat_o  => m2_dat_i,
1055    m2_adr_i  => m2_adr_o,
1056    m2_sel_i  => m2_sel_o,
1057    m2_we_i   => m2_we_o,
1058    m2_cyc_i  => m2_cyc_o,
1059    m2_stb_i  => m2_stb_o,
1060    m2_ack_o  => m2_ack_i,
1061    m2_err_o  => open, --m2_err_i,
1062    m2_rty_o  => open, --m2_rty_i,
1063    m2_cab_i  => m2_cab_o,
1064
1065    --Master 3
1066    m3_dat_i  => m3_dat_o,
1067    m3_dat_o  => m3_dat_i,
1068    m3_adr_i  => m3_adr_o,
1069    m3_sel_i  => m3_sel_o,
1070    m3_we_i   => m3_we_o,
1071    m3_cyc_i  => m3_cyc_o,
1072    m3_stb_i  => m3_stb_o,
1073    m3_ack_o  => m3_ack_i,
1074    m3_err_o  => open, --m3_err_i,
1075    m3_rty_o  => open, --m3_rty_i,
1076    m3_cab_i  => m3_cab_o,
1077
1078    --Master 4
```

```
1079      m4_dat_i  => m4_dat_o,
1080      m4_dat_o  => m4_dat_i,
1081      m4_adr_i  => m4_adr_o,
1082      m4_sel_i  => m4_sel_o,
1083      m4_we_i   => m4_we_o,
1084      m4_cyc_i  => m4_cyc_o,
1085      m4_stb_i  => m4_stb_o,
1086      m4_ack_o  => m4_ack_i,
1087      m4_err_o  => open, --m4_err_i,
1088      m4_rty_o  => open, --m4_rty_i,
1089      m4_cab_i  => m4_cab_o,
1090
1091      --Master 5
1092      m5_dat_i  => m5_dat_o,
1093      m5_dat_o  => m5_dat_i,
1094      m5_adr_i  => m5_adr_o,
1095      m5_sel_i  => m5_sel_o,
1096      m5_we_i   => m5_we_o,
1097      m5_cyc_i  => m5_cyc_o,
1098      m5_stb_i  => m5_stb_o,
1099      m5_ack_o  => m5_ack_i,
1100      m5_err_o  => open, --m5_err_i,
1101      m5_rty_o  => open, --m5_rty_i,
1102      m5_cab_i  => m5_cab_o,
1103
1104      --Master 6
1105      m6_dat_i  => m6_dat_o,
1106      m6_dat_o  => m6_dat_i,
1107      m6_adr_i  => m6_adr_o,
1108      m6_sel_i  => m6_sel_o,
1109      m6_we_i   => m6_we_o,
1110      m6_cyc_i  => m6_cyc_o,
1111      m6_stb_i  => m6_stb_o,
1112      m6_ack_o  => m6_ack_i,
1113      m6_err_o  => open, --m6_err_i,
1114      m6_rty_o  => open, --m6_rty_i,
```

```
1115    m6_cab_i   => m6_cab_o,
1116
1117    --Master 7
1118    m7_dat_i  => m7_dat_o,
1119    m7_dat_o  => m7_dat_i,
1120    m7_adr_i  => m7_adr_o,
1121    m7_sel_i  => m7_sel_o,
1122    m7_we_i   => m7_we_o,
1123    m7_cyc_i  => m7_cyc_o,
1124    m7_stb_i  => m7_stb_o,
1125    m7_ack_o  => m7_ack_i,
1126    m7_err_o  => open, --m7_err_i,
1127    m7_rty_o  => open, --m7_rty_i,
1128    m7_cab_i  => m7_cab_o,
1129
1130    --Slave 0
1131    s0_dat_i  => s0_dat_o,
1132    s0_dat_o  => s0_dat_i,
1133    s0_adr_o  => s0_adr_i,
1134    s0_sel_o  => s0_sel_i,
1135    s0_we_o   => s0_we_i,
1136    s0_cyc_o  => s0_cyc_i,
1137    s0_stb_o  => s0_stb_i,
1138    s0_ack_i  => s0_ack_o,
1139    s0_err_i  => s0_err_o,
1140    s0_rty_i  => s0_rty_o,
1141    s0_cab_o  => s0_cab_i,
1142
1143    --Slave 1
1144    s1_dat_i  => s1_dat_o,
1145    s1_dat_o  => s1_dat_i,
1146    s1_adr_o  => s1_adr_i,
1147    s1_sel_o  => s1_sel_i,
1148    s1_we_o   => s1_we_i,
1149    s1_cyc_o  => s1_cyc_i,
1150    s1_stb_o  => s1_stb_i,
```

```
1151      s1_ack_i  => s1_ack_o,
1152      s1_err_i  => s1_err_o,
1153      s1_rty_i  => s1_rty_o,
1154      s1_cab_o  => s1_cab_i,
1155
1156      --Slave 2
1157      s2_dat_i  => s2_dat_o,
1158      s2_dat_o  => s2_dat_i,
1159      s2_adr_o  => s2_adr_i,
1160      s2_sel_o  => s2_sel_i,
1161      s2_we_o   => s2_we_i,
1162      s2_cyc_o  => s2_cyc_i,
1163      s2_stb_o  => s2_stb_i,
1164      s2_ack_i  => s2_ack_o,
1165      s2_err_i  => s2_err_o,
1166      s2_rty_i  => s2_rty_o,
1167      s2_cab_o  => s2_cab_i,
1168
1169      --Slave 3
1170      s3_dat_i  => s3_dat_o,
1171      s3_dat_o  => s3_dat_i,
1172      s3_adr_o  => s3_adr_i,
1173      s3_sel_o  => s3_sel_i,
1174      s3_we_o   => s3_we_i,
1175      s3_cyc_o  => s3_cyc_i,
1176      s3_stb_o  => s3_stb_i,
1177      s3_ack_i  => s3_ack_o,
1178      s3_err_i  => s3_err_o,
1179      s3_rty_i  => s3_rty_o,
1180      s3_cab_o  => s3_cab_i,
1181
1182      --Slave 4
1183      s4_dat_i  => s4_dat_o,
1184      s4_dat_o  => s4_dat_i,
1185      s4_adr_o  => s4_adr_i,
1186      s4_sel_o  => s4_sel_i,
```

```
1187  s4_we_o   => s4_we_i,
1188  s4_cyc_o  => s4_cyc_i,
1189  s4_stb_o  => s4_stb_i,
1190  s4_ack_i  => s4_ack_o,
1191  s4_err_i  => s4_err_o,
1192  s4_rty_i  => s4_rty_o,
1193  s4_cab_o  => s4_cab_i,
1194
1195  --Slave 5
1196  s5_dat_i  => s5_dat_o,
1197  s5_dat_o  => s5_dat_i,
1198  s5_adr_o  => s5_adr_i,
1199  s5_sel_o  => s5_sel_i,
1200  s5_we_o   => s5_we_i,
1201  s5_cyc_o  => s5_cyc_i,
1202  s5_stb_o  => s5_stb_i,
1203  s5_ack_i  => s5_ack_o,
1204  s5_err_i  => s5_err_o,
1205  s5_rty_i  => s5_rty_o,
1206  s5_cab_o  => s5_cab_i,
1207
1208  --Slave 6
1209  s6_dat_i  => s6_dat_o,
1210  s6_dat_o  => s6_dat_i,
1211  s6_adr_o  => s6_adr_i,
1212  s6_sel_o  => s6_sel_i,
1213  s6_we_o   => s6_we_i,
1214  s6_cyc_o  => s6_cyc_i,
1215  s6_stb_o  => s6_stb_i,
1216  s6_ack_i  => s6_ack_o,
1217  s6_err_i  => s6_err_o,
1218  s6_rty_i  => s6_rty_o,
1219  s6_cab_o  => s6_cab_i,
1220
1221  --Slave 7
1222  s7_dat_i  => s7_dat_o,
```

```
1223      s7_dat_o  => s7_dat_i,
1224      s7_adr_o  => s7_adr_i,
1225      s7_sel_o  => s7_sel_i,
1226      s7_we_o   => s7_we_i,
1227      s7_cyc_o  => s7_cyc_i,
1228      s7_stb_o  => s7_stb_i,
1229      s7_ack_i  => s7_ack_o,
1230      s7_err_i  => s7_err_o,
1231      s7_rty_i  => s7_rty_o,
1232      s7_cab_o  => s7_cab_i
1233
1234      );
1235
1236    ------------------------------------------------------------------
1237    --
1238    --  ICON core instance
1239    --
1240    ------------------------------------------------------------------
1241    i_icon : icon
1242      port map
1243      (
1244        control0    => control0,
1245        control1    => control1
1246      );
1247
1248  ------------------------------------------------------------------
1249    --
1250    --  ILA core instance
1251    --
1252    ------------------------------------------------------------------
1253    i_ila : ila
1254      port map
1255      (
1256        control   => control0,
1257        clk       => clk_12_5MHZ,
1258        data      => data,
```

```
1259        trig0     => m0_adr_o
1260      );
1261
1262    ------------------------------------------------------------------
1263    --
1264    --  VIO core instance
1265    --
1266    ------------------------------------------------------------------
1267    i_vio : vio
1268      port map
1269      (
1270        control   => control1,
1271        async_out => async_out
1272      );
1273
1274    --Set unconected cyc signals
1275  --  m0_cyc_o <= '0';
1276  --  m1_cyc_o <= '0';
1277  --  m2_cyc_o <= '0';
1278  --  m3_cyc_o <= '0';
1279  --  m4_cyc_o <= '0';
1280  --  m5_cyc_o <= '0';
1281  --  m6_cyc_o <= '0';
1282  --  m7_cyc_o <= '0';
1283
1284    --Set unconected err signals
1285    m0_err_i  <= '0';
1286    m1_err_i  <= '0';
1287    m2_err_i  <= '0';
1288    m3_err_i  <= '0';
1289    m4_err_i  <= '0';
1290    m5_err_i  <= '0';
1291    m6_err_i  <= '0';
1292    m7_err_i  <= '0';
1293
1294    --Set unconected rty signals
```

```
1295    m0_rty_i  <= '0';
1296    m1_rty_i  <= '0';
1297    m2_rty_i  <= '0';
1298    m3_rty_i  <= '0';
1299    m4_rty_i  <= '0';
1300    m5_rty_i  <= '0';
1301    m6_rty_i  <= '0';
1302    m7_rty_i  <= '0';
1303
1304
1305
1306      data <= "00000000000000000000000000000000" & m1_adr_o & m1_dat_o &  m1_ack_i & m1_stb_o & tx_pad & rx
              ;
1307      tx <= tx_pad;
1308
1309    ackbuff: process(clk_12_5MHZ, m0_ack_i, m1_ack_i,  m2_ack_i,  m3_ack_i,  m4_ack_i,  m5_ack_i,
          m6_ack_i,  m7_ack_i, reset)
1310    begin
1311      if(reset = '0') then
1312        ack0    <= '0';
1313        ack1    <= '0';
1314        ack2    <= '0';
1315        ack3    <= '0';
1316        ack4    <= '0';
1317        ack5    <= '0';
1318        ack6    <= '0';
1319        ack7    <= '0';
1320
1321      else
1322        -- set m0_ack
1323        if m0_ack_i = '0' or m0_ack_i = '1' then
1324          ack0    <= m0_ack_i;
1325        else
1326          ack0    <= '0';
1327        end if;
1328        -- set m1_ack
```

```
1329  if m1_ack_i = '0' or m1_ack_i = '1' then
1330    ack1    <= m1_ack_i;
1331  else
1332    ack1    <= '0';
1333  end if;
1334  -- set m2_ack
1335  if m2_ack_i = '0' or m2_ack_i = '1' then
1336    ack2    <= m2_ack_i;
1337  else
1338    ack2    <= '0';
1339  end if;
1340  -- set m3_ack
1341  if m3_ack_i = '0' or m3_ack_i = '1' then
1342    ack3    <= m3_ack_i;
1343  else
1344    ack3    <= '0';
1345  end if;
1346  -- set m3_ack
1347  if m4_ack_i = '0' or m4_ack_i = '1' then
1348    ack4    <= m4_ack_i;
1349  else
1350    ack4    <= '0';
1351  end if;
1352  -- set m5_ack
1353  if m5_ack_i = '0' or m5_ack_i = '1' then
1354    ack5    <= m5_ack_i;
1355  else
1356    ack5    <= '0';
1357  end if;
1358  -- set m6_ack
1359  if m6_ack_i = '0' or m6_ack_i = '1' then
1360    ack6    <= m6_ack_i;
1361  else
1362    ack6    <= '0';
1363  end if;
1364  -- set m7_ack
```

```
1365      if m7_ack_i = '0' or m7_ack_i = '1' then
1366        ack7    <= m7_ack_i;
1367      else
1368        ack7    <= '0';
1369      end if;
1370    end if;
1371    end process ackbuff;
1372  end struc;
```

A.5 Tree architecture

A.5.1 noc_tree.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Table for tree NOV
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 16/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity noc_tree is
22
23     port(
24     --I/O Ports
25     wb_clk_i     : in bit_t;    -- The Clock
26     wb_rst_i     : in bit_t;    -- The reset signal
27
28 --WishBone Master 0
29     m0wb_dat_i     : in word_t;
30     m0wb_dat_o     : out word_t;
```

```
31      m0wb_ack_o    : out bit_t;
32      m0wb_adr_i    : in word_t;
33      m0wb_cyc_i    : in bit_t;
34      m0wb_stb_i    : in bit_t;
35      m0wb_err_o    : out bit_t;
36      m0wb_rty_o    : out bit_t;
37      m0wb_sel_i    : in std_logic_vector(3 downto 0);
38      m0wb_we_i   : in bit_t;
39
40      --WishBone Master 1
41      m1wb_dat_i    : in word_t;
42      m1wb_dat_o    : out word_t;
43      m1wb_ack_o    : out bit_t;
44      m1wb_adr_i    : in word_t;
45      m1wb_cyc_i    : in bit_t;
46      m1wb_stb_i    : in bit_t;
47      m1wb_err_o    : out bit_t;
48      m1wb_rty_o    : out bit_t;
49      m1wb_sel_i    : in std_logic_vector(3 downto 0);
50      m1wb_we_i   : in bit_t;
51
52      --WishBone Master 2
53      m2wb_dat_i    : in word_t;
54      m2wb_dat_o    : out word_t;
55      m2wb_ack_o    : out bit_t;
56      m2wb_adr_i    : in word_t;
57      m2wb_cyc_i    : in bit_t;
58      m2wb_stb_i    : in bit_t;
59      m2wb_err_o    : out bit_t;
60      m2wb_rty_o    : out bit_t;
61      m2wb_sel_i    : in std_logic_vector(3 downto 0);
62      m2wb_we_i   : in bit_t;
63
64      --WishBone Master 3
65      m3wb_dat_i    : in word_t;
66      m3wb_dat_o    : out word_t;
```

```
67   m3wb_ack_o    : out bit_t;
68   m3wb_adr_i    : in word_t;
69   m3wb_cyc_i    : in bit_t;
70   m3wb_stb_i    : in bit_t;
71   m3wb_err_o    : out bit_t;
72   m3wb_rty_o    : out bit_t;
73   m3wb_sel_i    : in std_logic_vector(3 downto 0);
74   m3wb_we_i     : in bit_t;
75
76   --WishBone Master 4
77   m4wb_dat_i    : in word_t;
78   m4wb_dat_o    : out word_t;
79   m4wb_ack_o    : out bit_t;
80   m4wb_adr_i    : in word_t;
81   m4wb_cyc_i    : in bit_t;
82   m4wb_stb_i    : in bit_t;
83   m4wb_err_o    : out bit_t;
84   m4wb_rty_o    : out bit_t;
85   m4wb_sel_i    : in std_logic_vector(3 downto 0);
86   m4wb_we_i     : in bit_t;
87
88   --WishBone Master 5
89   m5wb_dat_i    : in word_t;
90   m5wb_dat_o    : out word_t;
91   m5wb_ack_o    : out bit_t;
92   m5wb_adr_i    : in word_t;
93   m5wb_cyc_i    : in bit_t;
94   m5wb_stb_i    : in bit_t;
95   m5wb_err_o    : out bit_t;
96   m5wb_rty_o    : out bit_t;
97   m5wb_sel_i    : in std_logic_vector(3 downto 0);
98   m5wb_we_i     : in bit_t;
99
100  --WishBone Master 6
101  m6wb_dat_i    : in word_t;
102  m6wb_dat_o    : out word_t;
```

```
103     m6wb_ack_o    : out bit_t;
104     m6wb_adr_i    : in word_t;
105     m6wb_cyc_i    : in bit_t;
106     m6wb_stb_i    : in bit_t;
107     m6wb_err_o    : out bit_t;
108     m6wb_rty_o    : out bit_t;
109     m6wb_sel_i    : in std_logic_vector(3 downto 0);
110     m6wb_we_i   : in bit_t;
111
112     --WishBone Master 7
113     m7wb_dat_i    : in word_t;
114     m7wb_dat_o    : out word_t;
115     m7wb_ack_o    : out bit_t;
116     m7wb_adr_i    : in word_t;
117     m7wb_cyc_i    : in bit_t;
118     m7wb_stb_i    : in bit_t;
119     m7wb_err_o    : out bit_t;
120     m7wb_rty_o    : out bit_t;
121     m7wb_sel_i    : in std_logic_vector(3 downto 0);
122     m7wb_we_i   : in bit_t;
123
124     --WishBone Slave 0
125     s0wb_dat_i  : in word_t;
126     s0wb_dat_o  : out word_t;
127     s0wb_ack_i  : in bit_t;
128     s0wb_adr_o  : out word_t;
129     s0wb_cyc_o  : out bit_t;
130     s0wb_stb_o  : out bit_t;
131     s0wb_err_i  : in bit_t;
132     s0wb_rty_i  : in bit_t;
133     s0wb_sel_o  : out std_logic_vector(3 downto 0);
134     s0wb_we_o : out bit_t;
135
136     --WishBone Slave 1
137     s1wb_dat_i  : in word_t;
138     s1wb_dat_o  : out word_t;
```

```
139     s1wb_ack_i  : in bit_t;
140     s1wb_adr_o  : out word_t;
141     s1wb_cyc_o  : out bit_t;
142     s1wb_stb_o  : out bit_t;
143     s1wb_err_i  : in bit_t;
144     s1wb_rty_i  : in bit_t;
145     s1wb_sel_o  : out std_logic_vector(3 downto 0);
146     s1wb_we_o : out bit_t;
147
148     --WishBone Slave 2
149     s2wb_dat_i  : in word_t;
150     s2wb_dat_o  : out word_t;
151     s2wb_ack_i  : in bit_t;
152     s2wb_adr_o  : out word_t;
153     s2wb_cyc_o  : out bit_t;
154     s2wb_stb_o  : out bit_t;
155     s2wb_err_i  : in bit_t;
156     s2wb_rty_i  : in bit_t;
157     s2wb_sel_o  : out std_logic_vector(3 downto 0);
158     s2wb_we_o : out bit_t;
159
160     --WishBone Slave 3
161     s3wb_dat_i  : in word_t;
162     s3wb_dat_o  : out word_t;
163     s3wb_ack_i  : in bit_t;
164     s3wb_adr_o  : out word_t;
165     s3wb_cyc_o  : out bit_t;
166     s3wb_stb_o  : out bit_t;
167     s3wb_err_i  : in bit_t;
168     s3wb_rty_i  : in bit_t;
169     s3wb_sel_o  : out std_logic_vector(3 downto 0);
170     s3wb_we_o : out bit_t
171
172 --      --WishBone Slave 4
173 --      s4wb_dat_i  : in word_t;
174 --      s4wb_dat_o  : out word_t;
```

```
175  --    s4wb_ack_i  : in bit_t;
176  --    s4wb_adr_o  : out word_t;
177  --    s4wb_cyc_o  : out bit_t;
178  --    s4wb_stb_o  : out bit_t;
179  --    s4wb_err_i  : in bit_t;
180  --    s4wb_rty_i  : in bit_t;
181  --    s4wb_sel_o  : out std_logic_vector(3 downto 0);
182  --    s4wb_we_o   : out bit_t;
183  --
184  --  --WishBone Slave 5
185  --    s5wb_dat_i  : in word_t;
186  --    s5wb_dat_o  : out word_t;
187  --    s5wb_ack_i  : in bit_t;
188  --    s5wb_adr_o  : out word_t;
189  --    s5wb_cyc_o  : out bit_t;
190  --    s5wb_stb_o  : out bit_t;
191  --    s5wb_err_i  : in bit_t;
192  --    s5wb_rty_i  : in bit_t;
193  --    s5wb_sel_o  : out std_logic_vector(3 downto 0);
194  --    s5wb_we_o   : out bit_t;
195  --
196  --  --WishBone Slave 6
197  --    s6wb_dat_i  : in word_t;
198  --    s6wb_dat_o  : out word_t;
199  --    s6wb_ack_i  : in bit_t;
200  --    s6wb_adr_o  : out word_t;
201  --    s6wb_cyc_o  : out bit_t;
202  --    s6wb_stb_o  : out bit_t;
203  --    s6wb_err_i  : in bit_t;
204  --    s6wb_rty_i  : in bit_t;
205  --    s6wb_sel_o  : out std_logic_vector(3 downto 0);
206  --    s6wb_we_o   : out bit_t;
207  --
208  --  --WishBone Slave 7
209  --    s7wb_dat_i  : in word_t;
210  --    s7wb_dat_o  : out word_t;
```

```
211 --    s7wb_ack_i  : in bit_t;
212 --    s7wb_adr_o  : out word_t;
213 --    s7wb_cyc_o  : out bit_t;
214 --    s7wb_stb_o  : out bit_t;
215 --    s7wb_err_i  : in bit_t;
216 --    s7wb_rty_i  : in bit_t;
217 --    s7wb_sel_o  : out std_logic_vector(3 downto 0);
218 --    s7wb_we_o : out bit_t
219
220       );
221
222   end noc_tree;
223
224   architecture struc of noc_tree is
225     component noc_na_slave is
226     port(
227       --Common WishBone signals
228       wb_clk_i  : in bit_t;
229       wb_rst_i  : in bit_t;
230       wb_data_i : in word_t;
231       wb_data_o : out word_t;
232
233       --WishBone Master
234       wb_ack_i  : in bit_t;
235       wb_adr_o  : out word_t;
236       wb_cyc_o  : out bit_t;
237       wb_stb_o  : out bit_t;
238       wb_err_i  : in bit_t;
239       wb_rty_i  : in bit_t;
240       wb_sel_o  : out std_logic_vector(3 downto 0);
241       wb_we_o   : out bit_t;
242
243       --NoC signals
244       noc_in    : in noc_data;
245       noc_out   : out noc_data
246           );
```

```
247   end component;
248
249   component noc_router is
250
251   port(
252     --I/O ports
253     clk_i    : in bit_t;
254     rst_i    : in bit_t;
255
256     --NoC signals
257     --North
258     noc_n_i    : in noc_data;
259     noc_n_o    : out noc_data;
260
261     --South
262     noc_s_i    : in noc_data;
263     noc_s_o    : out noc_data;
264
265     --East
266     noc_e_i    : in noc_data;
267     noc_e_o    : out noc_data;
268
269     --West
270     noc_w_i    : in noc_data;
271     noc_w_o    : out noc_data;
272
273     --IP
274     ip_i     : in noc_data;
275     ip_o     : out noc_data
276         );
277   end component;
278
279   component noc_na_master is
280
281   generic(
282     X    : integer := 0;
```

```
283       Y   : integer := 0
284     );
285
286     port(
287       --Common WishBone signals
288       wb_clk_i  : in bit_t;
289       wb_rst_i  : in bit_t;
290       wb_data_i : in word_t;
291       wb_data_o : out word_t;
292
293
294
295       --WishBone Master
296       wb_ack_o  : out bit_t;
297       wb_adr_i  : in word_t;
298       wb_cyc_i  : in bit_t;
299       wb_err_o  : out bit_t;
300       wb_rty_o  : out bit_t;
301       wb_sel_i  : in std_logic_vector(3 downto 0);
302       wb_we_i   : in bit_t;
303       wb_stb_i  : in bit_t;
304
305       --NoC signals
306       noc_in    : in noc_data;
307       noc_out   : out noc_data
308           );
309     end component;
310
311     --NOC signals
312     --Masters to r1
313     SIGNAL  m0r10     : noc_data;
314     SIGNAL  m1r10     : noc_data;
315     SIGNAL  m2r11     : noc_data;
316     SIGNAL  m3r11     : noc_data;
317     SIGNAL  m4r12     : noc_data;
318     SIGNAL  m5r12     : noc_data;
```

```
319  SIGNAL  m6r13    : noc_data;
320  SIGNAL  m7r13    : noc_data;
321  --r1 to master
322  SIGNAL  r10m0    : noc_data;
323  SIGNAL  r10m1    : noc_data;
324  SIGNAL  r11m2    : noc_data;
325  SIGNAL  r11m3    : noc_data;
326  SIGNAL  r12m4    : noc_data;
327  SIGNAL  r12m5    : noc_data;
328  SIGNAL  r13m6    : noc_data;
329  SIGNAL  r13m7    : noc_data;
330  --r1 to r2
331  SIGNAL  r10r20     : noc_data;
332  SIGNAL  r11r20     : noc_data;
333  SIGNAL  r12r21     : noc_data;
334  SIGNAL  r13r21     : noc_data;
335  --r2 to r1
336  SIGNAL  r20r10     : noc_data;
337  SIGNAL  r20r11     : noc_data;
338  SIGNAL  r21r12     : noc_data;
339  SIGNAL  r21r13     : noc_data;
340  --r2 to r3
341  SIGNAL  r20r30     : noc_data;
342  SIGNAL  r21r30     : noc_data;
343  --r3 to r2
344  SIGNAL  r30r20     : noc_data;
345  SIGNAL  r30r21     : noc_data;
346  --r3 to r4
347  SIGNAL  r30r40     : noc_data;
348  --r4 to r3
349  SIGNAL  r40r30     : noc_data;
350  --r4 to r5
351  SIGNAL  r40r50     : noc_data;
352  SIGNAL  r40r51     : noc_data;
353  --r5 to r4
354  SIGNAL  r50r40     : noc_data;
```

```
355     SIGNAL  r51r40      : noc_data;
356     --r5 to slaves
357     SIGNAL  r50s0     : noc_data;
358     SIGNAL  r50s1     : noc_data;
359     SIGNAL  r51s2     : noc_data;
360     SIGNAL  r51s3     : noc_data;
361     --slaves to r5
362     SIGNAL  s0r50     : noc_data;
363     SIGNAL  s1r50     : noc_data;
364     SIGNAL  s2r51     : noc_data;
365     SIGNAL  s3r51     : noc_data;
366
367     --
368     SIGNAL nc     : noc_data; --No connection signal
369   begin
370
371     nc  <= (OTHERS => '0');
372
373     --Master connections
374
375     m0 : noc_na_master
376       port map(
377       --Common WishBone signals
378       wb_clk_i  => wb_clk_i,
379       wb_rst_i  => wb_rst_i,
380       wb_data_i => m0wb_dat_i,
381       wb_data_o => m0wb_dat_o,
382
383       --WishBone Master
384       wb_ack_o  => m0wb_ack_o,
385       wb_adr_i  => m0wb_adr_i,
386       wb_cyc_i  => m0wb_cyc_i,
387       wb_err_o  => m0wb_err_o,
388       wb_rty_o  => m0wb_rty_o,
389       wb_sel_i  => m0wb_sel_i,
390       wb_we_i   => m0wb_we_i,
```

```
391       wb_stb_i  => m0wb_stb_i,
392
393       --NoC signals
394       noc_in    => r10m0,
395       noc_out   => m0r10
396       );
397
398     m1 : noc_na_master
399       port map(
400       --Common WishBone signals
401       wb_clk_i  => wb_clk_i,
402       wb_rst_i  => wb_rst_i,
403       wb_data_i => m1wb_dat_i,
404       wb_data_o => m1wb_dat_o,
405
406       --WishBone Master
407       wb_ack_o  => m1wb_ack_o,
408       wb_adr_i  => m1wb_adr_i,
409       wb_cyc_i  => m1wb_cyc_i,
410       wb_err_o  => m1wb_err_o,
411       wb_rty_o  => m1wb_rty_o,
412       wb_sel_i  => m1wb_sel_i,
413       wb_we_i   => m1wb_we_i,
414       wb_stb_i  => m1wb_stb_i,
415
416       --NoC signals
417       noc_in    => r10m1,
418       noc_out   => m1r10
419       );
420
421
422     m2 : noc_na_master
423       port map(
424       --Common WishBone signals
425       wb_clk_i  => wb_clk_i,
426       wb_rst_i  => wb_rst_i,
```

```
427       wb_data_i => m2wb_dat_i,
428       wb_data_o => m2wb_dat_o,
429
430       --WishBone Master
431       wb_ack_o  => m2wb_ack_o,
432       wb_adr_i  => m2wb_adr_i,
433       wb_cyc_i  => m2wb_cyc_i,
434       wb_err_o  => m2wb_err_o,
435       wb_rty_o  => m2wb_rty_o,
436       wb_sel_i  => m2wb_sel_i,
437       wb_we_i   => m2wb_we_i,
438       wb_stb_i  => m2wb_stb_i,
439
440       --NoC signals
441       noc_in    => r11m2,
442       noc_out   => m2r11
443       );
444
445
446     m3 : noc_na_master
447       port map(
448       --Common WishBone signals
449       wb_clk_i  => wb_clk_i,
450       wb_rst_i  => wb_rst_i,
451       wb_data_i => m3wb_dat_i,
452       wb_data_o => m3wb_dat_o,
453
454       --WishBone Master
455       wb_ack_o  => m3wb_ack_o,
456       wb_adr_i  => m3wb_adr_i,
457       wb_cyc_i  => m3wb_cyc_i,
458       wb_err_o  => m3wb_err_o,
459       wb_rty_o  => m3wb_rty_o,
460       wb_sel_i  => m3wb_sel_i,
461       wb_we_i   => m3wb_we_i,
462       wb_stb_i  => m3wb_stb_i,
```

```
463
464       --NoC signals
465       noc_in     => r11m3,
466       noc_out    => m3r11
467       );
468
469
470     m4 : noc_na_master
471       port map(
472       --Common WishBone signals
473       wb_clk_i  => wb_clk_i,
474       wb_rst_i  => wb_rst_i,
475       wb_data_i => m4wb_dat_i,
476       wb_data_o => m4wb_dat_o,
477
478       --WishBone Master
479       wb_ack_o  => m4wb_ack_o,
480       wb_adr_i  => m4wb_adr_i,
481       wb_cyc_i  => m4wb_cyc_i,
482       wb_err_o  => m4wb_err_o,
483       wb_rty_o  => m4wb_rty_o,
484       wb_sel_i  => m4wb_sel_i,
485       wb_we_i   => m4wb_we_i,
486       wb_stb_i  => m4wb_stb_i,
487
488       --NoC signals
489       noc_in     => r12m4,
490       noc_out    => m4r12
491       );
492
493
494     m5 : noc_na_master
495       port map(
496       --Common WishBone signals
497       wb_clk_i  => wb_clk_i,
498       wb_rst_i  => wb_rst_i,
```

```
499     wb_data_i => m5wb_dat_i,
500     wb_data_o => m5wb_dat_o,
501
502     --WishBone Master
503     wb_ack_o  => m5wb_ack_o,
504     wb_adr_i  => m5wb_adr_i,
505     wb_cyc_i  => m5wb_cyc_i,
506     wb_err_o  => m5wb_err_o,
507     wb_rty_o  => m5wb_rty_o,
508     wb_sel_i  => m5wb_sel_i,
509     wb_we_i   => m5wb_we_i,
510     wb_stb_i  => m5wb_stb_i,
511
512     --NoC signals
513     noc_in    => r12m5,
514     noc_out   => m5r12
515     );
516
517
518   m6 : noc_na_master
519     port map(
520     --Common WishBone signals
521     wb_clk_i  => wb_clk_i,
522     wb_rst_i  => wb_rst_i,
523     wb_data_i => m6wb_dat_i,
524     wb_data_o => m6wb_dat_o,
525
526     --WishBone Master
527     wb_ack_o  => m6wb_ack_o,
528     wb_adr_i  => m6wb_adr_i,
529     wb_cyc_i  => m6wb_cyc_i,
530     wb_err_o  => m6wb_err_o,
531     wb_rty_o  => m6wb_rty_o,
532     wb_sel_i  => m6wb_sel_i,
533     wb_we_i   => m6wb_we_i,
534     wb_stb_i  => m6wb_stb_i,
```

```
535
536     --NoC signals
537     noc_in     => r13m6,
538     noc_out    => m6r13
539     );
540
541
542   m7 : noc_na_master
543     port map(
544     --Common WishBone signals
545     wb_clk_i  => wb_clk_i,
546     wb_rst_i  => wb_rst_i,
547     wb_data_i => m7wb_dat_i,
548     wb_data_o => m7wb_dat_o,
549
550     --WishBone Master
551     wb_ack_o  => m7wb_ack_o,
552     wb_adr_i  => m7wb_adr_i,
553     wb_cyc_i  => m7wb_cyc_i,
554     wb_err_o  => m7wb_err_o,
555     wb_rty_o  => m7wb_rty_o,
556     wb_sel_i  => m7wb_sel_i,
557     wb_we_i   => m7wb_we_i,
558     wb_stb_i  => m7wb_stb_i,
559
560     --NoC signals
561     noc_in     => r13m7,
562     noc_out    => m7r13
563     );
564
565   -- Route connections
566
567   -- First line
568
569   r10 : noc_router
570
```

```
571   port map (
572     --I/O ports
573     clk_i    => wb_clk_i,
574     rst_i    => wb_rst_i,
575
576     --NoC signals
577     --North
578     noc_n_i    => m0r10,
579     noc_n_o    => r10m0,
580
581     --South
582     noc_s_i    => m1r10,
583     noc_s_o    => r10m1,
584
585     --East
586     noc_e_i    => r20r10,
587     noc_e_o    => r10r20,
588
589     --West
590     noc_w_i    => nc,
591     noc_w_o    => open,
592
593     --IP
594     ip_i       => nc,
595     ip_o       => open
596           );
597
598   r11 : noc_router
599
600   port map (
601     --I/O ports
602     clk_i    => wb_clk_i,
603     rst_i    => wb_rst_i,
604
605     --NoC signals
606     --North
```

```
607       noc_n_i    => m2r11,
608       noc_n_o    => r11m2,
609
610       --South
611       noc_s_i    => m3r11,
612       noc_s_o    => r11m3,
613
614       --East
615       noc_e_i    => r20r11,
616       noc_e_o    => r11r20,
617       --West
618       noc_w_i    => nc,
619       noc_w_o    => open,
620
621       --IP
622       ip_i     => nc,
623       ip_o     => open
624           );
625
626     r12 : noc_router
627
628     port map (
629       --I/O ports
630       clk_i    => wb_clk_i,
631       rst_i    => wb_rst_i,
632
633       --NoC signals
634       --North
635       noc_n_i    => m4r12,
636       noc_n_o    => r12m4,
637
638       --South
639       noc_s_i    => m5r12,
640       noc_s_o    => r12m5,
641
642       --East
```

```
643     noc_e_i   => r21r12,
644     noc_e_o   => r12r21,
645
646     --West
647     noc_w_i   => nc,
648     noc_w_o   => open,
649
650     --IP
651     ip_i      => nc,
652     ip_o      => open
653         );
654
655   r13 : noc_router
656
657   port map (
658     --I/O ports
659     clk_i     => wb_clk_i,
660     rst_i     => wb_rst_i,
661
662     --NoC signals
663     --North
664     noc_n_i   => m6r13,
665     noc_n_o   => r13m6,
666
667     --South
668     noc_s_i   => m7r13,
669     noc_s_o   => r13m7,
670
671     --East
672     noc_e_i   => r21r13,
673     noc_e_o   => r13r21,
674
675     --West
676     noc_w_i   => nc,
677     noc_w_o   => open,
678
```

```
679     --IP
680     ip_i     => nc,
681     ip_o     => open
682           );
683
684     --Second stage
685
686     r20 : noc_router
687
688     port map (
689       --I/O ports
690       clk_i   => wb_clk_i,
691       rst_i   => wb_rst_i,
692
693       --NoC signals
694       --North
695       noc_n_i   => r10r20,
696       noc_n_o   => r20r10,
697
698       --South
699       noc_s_i   => r11r20,
700       noc_s_o   => r20r11,
701
702       --East
703       noc_e_i   => r30r20,
704       noc_e_o   => r20r30,
705
706       --West
707       noc_w_i   => nc,
708       noc_w_o   => open,
709
710       --IP
711       ip_i     => nc,
712       ip_o     => open
713           );
714
```

```
715   r21 : noc_router
716
717   port map (
718     --I/O ports
719     clk_i    => wb_clk_i,
720     rst_i    => wb_rst_i,
721
722     --NoC signals
723     --North
724     noc_n_i    => r12r21,
725     noc_n_o    => r21r12,
726
727     --South
728     noc_s_i    => r13r21,
729     noc_s_o    => r21r13,
730
731     --East
732     noc_e_i    => r30r21,
733     noc_e_o    => r21r30,
734
735     --West
736     noc_w_i    => nc,
737     noc_w_o    => open,
738
739     --IP
740     ip_i       => nc,
741     ip_o       => open
742         );
743
744   r30 : noc_router
745
746   port map (
747     --I/O ports
748     clk_i    => wb_clk_i,
749     rst_i    => wb_rst_i,
750
```

```
751   --NoC signals
752   --North
753   noc_n_i    => r20r30,
754   noc_n_o    => r30r20,
755
756   --South
757   noc_s_i    => r21r30,
758   noc_s_o    => r30r21,
759
760   --East
761   noc_e_i    => r40r30,
762   noc_e_o    => r30r40,
763
764   --West
765   noc_w_i    => nc,
766   noc_w_o    => open,
767
768   --IP
769   ip_i       => nc,
770   ip_o       => open
771         );
772
773   r40 : noc_router
774
775   port map (
776     --I/O ports
777     clk_i    => wb_clk_i,
778     rst_i    => wb_rst_i,
779
780     --NoC signals
781     --North
782     noc_n_i    => r50r40,
783     noc_n_o    => r40r50,
784
785     --South
786     noc_s_i    => r51r40,
```

```
787     noc_s_o   => r40r51,
788
789     --East
790     noc_e_i   => nc,
791     noc_e_o   => open,
792
793     --West
794     noc_w_i   => r30r40,
795     noc_w_o   => r40r30,
796
797     --IP
798     ip_i      => nc,
799     ip_o      => open
800         );
801
802   r50 : noc_router
803
804   port map (
805     --I/O ports
806     clk_i   => wb_clk_i,
807     rst_i   => wb_rst_i,
808
809     --NoC signals
810     --North
811     noc_n_i   => s0r50,
812     noc_n_o   => r50s0,
813
814     --South
815     noc_s_i   => s1r50,
816     noc_s_o   => r50s1,
817
818     --East
819     noc_e_i   => nc,
820     noc_e_o   => open,
821
822     --West
```

```
823     noc_w_i   => r40r50,
824     noc_w_o   => r50r40,
825
826     --IP
827     ip_i      => nc,
828     ip_o      => open
829         );
830
831   r51 : noc_router
832
833   port map (
834     --I/O ports
835     clk_i     => wb_clk_i,
836     rst_i     => wb_rst_i,
837
838     --NoC signals
839     --North
840     noc_n_i   => s2r51,
841     noc_n_o   => r51s2,
842
843     --South
844     noc_s_i   => s3r51,
845     noc_s_o   => r51s3,
846
847     --East
848     noc_e_i   => nc,
849     noc_e_o   => open,
850
851     --West
852     noc_w_i   => r40r51,
853     noc_w_o   => r51r40,
854
855     --IP
856     ip_i      => nc,
857     ip_o      => open
858         );
```

```
859
860   --Slave connections
861
862   s0 : noc_na_slave
863   port map(
864     --Common WishBone signals
865     wb_clk_i  => wb_clk_i,
866     wb_rst_i  => wb_rst_i,
867     wb_data_i => s0wb_dat_i,
868     wb_data_o => s0wb_dat_o,
869
870     --WishBone Master
871     wb_ack_i  => s0wb_ack_i,
872     wb_adr_o  => s0wb_adr_o,
873     wb_cyc_o  => s0wb_cyc_o,
874     wb_stb_o  => s0wb_stb_o,
875     wb_err_i  => s0wb_err_i,
876     wb_rty_i  => s0wb_rty_i,
877     wb_sel_o  => s0wb_sel_o,
878     wb_we_o   => s0wb_we_o,
879
880     --NoC signals
881     noc_in    => r50s0,
882     noc_out   => s0r50
883       );
884   s1 : noc_na_slave
885   port map(
886     --Common WishBone signals
887     wb_clk_i  => wb_clk_i,
888     wb_rst_i  => wb_rst_i,
889     wb_data_i => s1wb_dat_i,
890     wb_data_o => s1wb_dat_o,
891
892     --WishBone Master
893     wb_ack_i  => s1wb_ack_i,
894     wb_adr_o  => s1wb_adr_o,
```

```
895       wb_cyc_o  => s1wb_cyc_o,
896       wb_stb_o  => s1wb_stb_o,
897       wb_err_i  => s1wb_err_i,
898       wb_rty_i  => s1wb_rty_i,
899       wb_sel_o  => s1wb_sel_o,
900       wb_we_o   => s1wb_we_o,
901
902       --NoC signals
903       noc_in    => r50s1,
904       noc_out   => s1r50
905           );
906
907     s2 : noc_na_slave
908     port map(
909       --Common WishBone signals
910       wb_clk_i  => wb_clk_i,
911       wb_rst_i  => wb_rst_i,
912       wb_data_i => s2wb_dat_i,
913       wb_data_o => s2wb_dat_o,
914
915       --WishBone Master
916       wb_ack_i  => s2wb_ack_i,
917       wb_adr_o  => s2wb_adr_o,
918       wb_cyc_o  => s2wb_cyc_o,
919       wb_stb_o  => s2wb_stb_o,
920       wb_err_i  => s2wb_err_i,
921       wb_rty_i  => s2wb_rty_i,
922       wb_sel_o  => s2wb_sel_o,
923       wb_we_o   => s2wb_we_o,
924
925       --NoC signals
926       noc_in    => r51s2,
927       noc_out   => s2r51
928           );
929
930     s3 : noc_na_slave
```

```
931   port map(
932     --Common WishBone signals
933     wb_clk_i  => wb_clk_i,
934     wb_rst_i  => wb_rst_i,
935     wb_data_i => s3wb_dat_i,
936     wb_data_o => s3wb_dat_o,
937
938     --WishBone Master
939     wb_ack_i  => s3wb_ack_i,
940     wb_adr_o  => s3wb_adr_o,
941     wb_cyc_o  => s3wb_cyc_o,
942     wb_stb_o  => s3wb_stb_o,
943     wb_err_i  => s3wb_err_i,
944     wb_rty_i  => s3wb_rty_i,
945     wb_sel_o  => s3wb_sel_o,
946     wb_we_o   => s3wb_we_o,
947
948     --NoC signals
949     noc_in    => r51s3,
950     noc_out   => s3r51
951         );
952
953
954   end struc;
```

A.5.2 tree_table.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Table for tree NOV
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 16/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity noc_table is
22
23   generic(
24       --Source
25       x_s : integer := 0;
26       y_s : integer := 0
27     );
28     port(
29       address : in std_logic_vector(addr_wdth-1 downto 0);
30       route : out std_logic_vector(route_wdth-1 downto 0)
31     );
32
33 end noc_table;
34
```

```
35 architecture struc of noc_table is
36   SIGNAL fill : std_logic_vector(route_wdth-1 downto 5*2) := (OTHERS => '0');
37 begin
38
39   route_lookup : process(address)
40   begin
41     if(address(31 downto 14) = CONv_STD_LOGIC_VECTOR(0, 31-14+1)) then  --Mem1(Imem)
42       route <= dest_east & dest_east & dest_east & dest_north & dest_north & fill;
43     elsif(address(31 downto 14) = CONv_STD_LOGIC_VECTOR(1, 31-14+1)) then --Mem2(Dmem)
44       --make route
45       route <= dest_east & dest_east & dest_east & dest_north & dest_south & fill;
46     elsif(address(31 downto 24) = CONv_STD_LOGIC_VECTOR(64, 31-24+1)) then  --Semaphore
47       --make route
48       route <= dest_east & dest_east & dest_east & dest_south & dest_south & fill;
49     elsif(address(31 downto 24) = CONv_STD_LOGIC_VECTOR(144, 31-24+1)) then --UART
50       --make route
51       route <= dest_east & dest_east & dest_east & dest_south & dest_north & fill;
52     else
53       --no dest
54       route <= (OTHERS => 'Z');
55     end if;
56   end process route_lookup;
57
58 end struc;
```

A.5.3 tree_noc_tb.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Test bench for Tree Noc Architecture
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 18/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity tree_soc_tb is
22
23
24 end tree_soc_tb;
25
26 architecture struc of tree_soc_tb is
27
28   component soc_tree
29     port(
30       clk_i : in bit_t;
31       rst_i : in bit_t;
32       tx  : out bit_t;              -- RS232 TX
33       rx  : in bit_t;               -- RS232 RX
34       running : out bit_t
```

```
35      );
36    end component;
37
38    SIGNAL clk     : bit_t;
39    SIGNAL reset   : bit_t;
40    SIGNAL tx      : bit_t;
41    SIGNAL rx      : bit_t;
42    SIGNAL running : bit_t;
43
44    begin
45
46    thesystem:soc_tree
47      port map(
48        clk_i   => clk,
49        rst_i   => reset,
50        tx  => tx,
51        rx  => '1',
52        running => running
53      );
54
55
56    process   --process
57    begin
58      reset <= '0';
59      wait for 105 ns;
60      reset <= '1';
61      wait for 1005 ns;
62      reset <= '0';
63      wait for 105 ns;
64      reset <= '1';
65      wait;
66    end process;
67
68    clock: process
69    begin  -- process clock
70      clk <= '0';
```

```
71      wait for 5 ns;
72      clk <= '1';
73      wait for 5 ns;
74    end process clock;
75
76
77
78
79  end struc;
```

A.5.4 tree_soc.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --      SoC design for tree NoC
4  --      Editor: Nikolaj Tørring - s042505
5  --      Version: 1.0
6  --      Data last modified: 25/4-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --      Version History
11 --
12 --      v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20 library UNISIM;
21 use UNISIM.all;
22
23 entity soc_tree is
24
25   port (
26     clk_i : in std_logic;
27     rst_i : in std_logic;
28     tx  : out std_logic;
29     rx  : in std_logic;
30     running : out std_logic);
31
32 end soc_tree;
33
34 architecture struc of soc_tree is
```

```
35
36    component or1200_top
37      generic(
38        dw    : integer := 32;
39        aw    : integer := 32;
40        ppic_ints : integer := 20
41      );
42      port(
43        clk_i           : in    std_logic;
44        rst_i           : in    std_logic;
45        pic_ints_i      : in    std_logic_vector(ppic_ints-1 downto 0);
46        clmode_i        : in    std_logic_vector(1 downto 0);
47
48        -- Instruction WISHBONE interface
49        iwb_clk_i       : in    std_logic;
50        iwb_rst_i       : in    std_logic;
51        iwb_ack_i       : in    std_logic;
52        iwb_err_i       : in    std_logic;
53        iwb_rty_i       : in    std_logic;
54        iwb_dat_i       : in    std_logic_vector(31 downto 0);
55        iwb_cyc_o       : out   std_logic;
56        iwb_adr_o       : out   std_logic_vector(31 downto 0);
57        iwb_stb_o       : out   std_logic;
58        iwb_we_o        : out   std_logic;
59        iwb_sel_o       : out   std_logic_vector(3 downto 0);
60        iwb_dat_o       : out   std_logic_vector(31 downto 0);
61        iwb_cab_o       : out   std_logic;
62
63        -- Data WISHBONE interface
64        dwb_clk_i       : in    std_logic;
65        dwb_rst_i       : in    std_logic;
66        dwb_ack_i       : in    std_logic;
67        dwb_err_i       : in    std_logic;
68        dwb_rty_i       : in    std_logic;
69        dwb_dat_i       : in    std_logic_vector(31 downto 0);
70        dwb_cyc_o       : out   std_logic;
```

```
71    dwb_adr_o       : out   std_logic_vector(31 downto 0);
72    dwb_stb_o       : out   std_logic;
73    dwb_we_o        : out   std_logic;
74    dwb_sel_o       : out   std_logic_vector(3 downto 0);
75    dwb_dat_o       : out   std_logic_vector(31 downto 0);
76    dwb_cab_o       : out   std_logic;
77
78    -- Debug interface
79    dbg_stall_i     : in    std_logic;
80    dbg_ewt_i       : in    std_logic;
81    dbg_lss_o       : out   std_logic_vector(3 downto 0);
82    dbg_is_o        : out   std_logic_vector(1 downto 0);
83    dbg_wp_o        : out   std_logic_vector(10 downto 0);
84    dbg_bp_o        : out   std_logic;
85    dbg_stb_i       : in    std_logic;
86    dbg_we_i        : in    std_logic;
87    dbg_adr_i       : in    std_logic_vector(31 downto 0);
88    dbg_dat_i       : in    std_logic_vector(31 downto 0);
89    dbg_dat_o       : out std_logic_vector(31 downto 0);
90    dbg_ack_o       : out std_logic;
91
92    -- Power Management interface
93    pm_cpustall_i   : in    std_logic;
94    pm_clksd_o      : out   std_logic_vector(3 downto 0);
95    pm_dc_gate_o    : out   std_logic;
96    pm_ic_gate_o    : out   std_logic;
97    pm_dmmu_gate_o  : out   std_logic;
98    pm_immu_gate_o  : out   std_logic;
99    pm_tt_gate_o    : out   std_logic;
100   pm_cpu_gate_o   : out   std_logic;
101   pm_wakeup_o     : out   std_logic;
102   pm_lvolt_o      : out   std_logic
103   );
104   end component;
105
106   component core_mem
```

```
107     port (
108       --I/O Ports
109       wb_clk_i     : in bit_t;    -- The Clock
110       wb_rst_i     : in bit_t;    -- The reset signal
111
112       --WB slave i/f
113       wb_data_i    : in word_t;     -- WB data in
114       wb_data_o    : out word_t;    -- WB data out
115       wb_adr_i     : in word_t;     -- WB adress
116       wb_sel_i     : in std_logic_vector(3 downto 0); --WB select input array
117       wb_we_i      : in bit_t;    -- WB write enable
118       wb_cyc_i     : in bit_t;    -- WB cycle input
119       wb_stb_i     : in bit_t;    -- WB strobe input
120       wb_ack_o     : out bit_t;     -- WB acknowledge output
121       wb_err_o     : out bit_t);    -- WB error signal
122   end component core_mem;
123
124   component semaphore
125     port (
126       --I/O Ports
127       wb_clk_i     : in bit_t;    -- The Clock
128       wb_rst_i     : in bit_t;    -- The reset signal
129
130       --WB slave i/f
131       wb_data_i    : in word_t;     -- WB data in
132       wb_data_o    : out word_t;    -- WB data out
133       wb_adr_i     : in word_t;     -- WB adress
134       wb_sel_i     : in std_logic_vector(3 downto 0); --WB select input array
135       wb_we_i      : in bit_t;    -- WB write enable
136       wb_cyc_i     : in bit_t;    -- WB cycle input
137       wb_stb_i     : in bit_t;    -- WB strobe input
138       wb_ack_o     : out bit_t;     -- WB acknowledge output
139       wb_err_o     : out bit_t);    -- WB error signal
140   end component semaphore;
141
142   component uart_top
```

```
143     port (
144       --I/O Ports
145       wb_clk_i    : in bit_t;   -- The Clock
146
147       --WB slave signal
148       wb_rst_i    : in bit_t;   -- The reset signal
149       wb_adr_i    : in std_logic_vector(4 downto 0) ;   -- WB adress
150       wb_dat_i    : in word_t;    -- WB data in
151       wb_dat_o    : out word_t;   -- WB data out
152       wb_we_i     : in bit_t;   -- WB write enable
153       wb_stb_i    : in bit_t;   -- WB strobe input
154       wb_cyc_i    : in bit_t;   -- WB cycle input
155       wb_ack_o    : out bit_t;    -- WB acknowledge output
156       wb_sel_i    : in std_logic_vector(3 downto 0); -- WB select input array
157
158       int_o     : out bit_t;      -- interrupt request
159
160       -- UART signals
161       -- serial input/output
162       stx_pad_o   : out bit_t;  -- Transmit data
163       srx_pad_i   : in bit_t; -- Recive Data
164
165       -- modem signals
166       rts_pad_o   : out bit_t;  -- Request to send
167       cts_pad_i   : in bit_t; -- Clear to send
168       dtr_pad_o   : out bit_t;  --
169       dsr_pad_i   : in bit_t; -- Data set ready
170       ri_pad_i    : in bit_t; -- Ring indicator
171       dcd_pad_i   : in bit_t  -- Data Carrier Detect
172     );
173
174   end component uart_top;
175
176   component noc_tree
177     port(
178
```

```
179     --I/O ports
180     wb_clk_i  : in  bit_t;
181     wb_rst_i  : in  bit_t;
182
183     --Master 0
184     m0wb_dat_i  : in  word_t;
185     m0wb_dat_o  : out    word_t;
186     m0wb_adr_i  : in     word_t;
187     m0wb_sel_i  : in     std_logic_vector(3 downto 0);
188     m0wb_we_i : in     bit_t;
189     m0wb_cyc_i  : in     bit_t;
190     m0wb_stb_i  : in     bit_t;
191     m0wb_ack_o  : out    bit_t;
192     m0wb_err_o  : out    bit_t;
193     m0wb_rty_o  : out    bit_t;
194
195     --Master 1
196     m1wb_dat_i  : in     word_t;
197     m1wb_dat_o  : out    word_t;
198     m1wb_adr_i  : in     word_t;
199     m1wb_sel_i  : in     std_logic_vector(3 downto 0);
200     m1wb_we_i : in     bit_t;
201     m1wb_cyc_i  : in     bit_t;
202     m1wb_stb_i  : in     bit_t;
203     m1wb_ack_o  : out    bit_t;
204     m1wb_err_o  : out    bit_t;
205     m1wb_rty_o  : out    bit_t;
206
207     --Master 2
208     m2wb_dat_i  : in     word_t;
209     m2wb_dat_o  : out    word_t;
210     m2wb_adr_i  : in     word_t;
211     m2wb_sel_i  : in     std_logic_vector(3 downto 0);
212     m2wb_we_i : in     bit_t;
213     m2wb_cyc_i  : in     bit_t;
214     m2wb_stb_i  : in     bit_t;
```

```
215   m2wb_ack_o   : out   bit_t;
216   m2wb_err_o   : out   bit_t;
217   m2wb_rty_o   : out   bit_t;
218
219   --Master 3
220   m3wb_dat_i   : in    word_t;
221   m3wb_dat_o   : out   word_t;
222   m3wb_adr_i   : in    word_t;
223   m3wb_sel_i   : in    std_logic_vector(3 downto 0);
224   m3wb_we_i    : in    bit_t;
225   m3wb_cyc_i   : in    bit_t;
226   m3wb_stb_i   : in    bit_t;
227   m3wb_ack_o   : out   bit_t;
228   m3wb_err_o   : out   bit_t;
229   m3wb_rty_o   : out   bit_t;
230
231   --Master 4
232   m4wb_dat_i   : in    word_t;
233   m4wb_dat_o   : out   word_t;
234   m4wb_adr_i   : in    word_t;
235   m4wb_sel_i   : in    std_logic_vector(3 downto 0);
236   m4wb_we_i    : in    bit_t;
237   m4wb_cyc_i   : in    bit_t;
238   m4wb_stb_i   : in    bit_t;
239   m4wb_ack_o   : out   bit_t;
240   m4wb_err_o   : out   bit_t;
241   m4wb_rty_o   : out   bit_t;
242
243   --Master 5
244   m5wb_dat_i   : in    word_t;
245   m5wb_dat_o   : out   word_t;
246   m5wb_adr_i   : in    word_t;
247   m5wb_sel_i   : in    std_logic_vector(3 downto 0);
248   m5wb_we_i    : in    bit_t;
249   m5wb_cyc_i   : in    bit_t;
250   m5wb_stb_i   : in    bit_t;
```

```
251     m5wb_ack_o  : out    bit_t;
252     m5wb_err_o  : out    bit_t;
253     m5wb_rty_o  : out    bit_t;
254
255     --Master 6
256     m6wb_dat_i  : in     word_t;
257     m6wb_dat_o  : out    word_t;
258     m6wb_adr_i  : in     word_t;
259     m6wb_sel_i  : in     std_logic_vector(3 downto 0);
260     m6wb_we_i : in     bit_t;
261     m6wb_cyc_i  : in     bit_t;
262     m6wb_stb_i  : in     bit_t;
263     m6wb_ack_o  : out    bit_t;
264     m6wb_err_o  : out    bit_t;
265     m6wb_rty_o  : out    bit_t;
266
267     --Master 7
268     m7wb_dat_i  : in     word_t;
269     m7wb_dat_o  : out    word_t;
270     m7wb_adr_i  : in     word_t;
271     m7wb_sel_i  : in     std_logic_vector(3 downto 0);
272     m7wb_we_i : in     bit_t;
273     m7wb_cyc_i  : in     bit_t;
274     m7wb_stb_i  : in     bit_t;
275     m7wb_ack_o  : out    bit_t;
276     m7wb_err_o  : out    bit_t;
277     m7wb_rty_o  : out    bit_t;
278
279     --Slave 0
280     s0wb_dat_i  : in     word_t;
281     s0wb_dat_o  : out    word_t;
282     s0wb_adr_o  : out    word_t;
283     s0wb_sel_o  : out    std_logic_vector(3 downto 0);
284     s0wb_we_o : out    bit_t;
285     s0wb_cyc_o  : out    bit_t;
286     s0wb_stb_o  : out    bit_t;
```

```
287   s0wb_ack_i  : in    bit_t;
288   s0wb_err_i  : in    bit_t;
289   s0wb_rty_i  : in    bit_t;
290
291   --Slave 1
292   s1wb_dat_i  : in    word_t;
293   s1wb_dat_o  : out   word_t;
294   s1wb_adr_o  : out   word_t;
295   s1wb_sel_o  : out   std_logic_vector(3 downto 0);
296   s1wb_we_o   : out   bit_t;
297   s1wb_cyc_o  : out   bit_t;
298   s1wb_stb_o  : out   bit_t;
299   s1wb_ack_i  : in    bit_t;
300   s1wb_err_i  : in    bit_t;
301   s1wb_rty_i  : in    bit_t;
302
303   --Slave 2
304   s2wb_dat_i  : in    word_t;
305   s2wb_dat_o  : out   word_t;
306   s2wb_adr_o  : out   word_t;
307   s2wb_sel_o  : out   std_logic_vector(3 downto 0);
308   s2wb_we_o   : out   bit_t;
309   s2wb_cyc_o  : out   bit_t;
310   s2wb_stb_o  : out   bit_t;
311   s2wb_ack_i  : in    bit_t;
312   s2wb_err_i  : in    bit_t;
313   s2wb_rty_i  : in    bit_t;
314
315   --Slave 3
316   s3wb_dat_i  : in  word_t;
317   s3wb_dat_o  : out word_t;
318   s3wb_adr_o  : out word_t;
319   s3wb_sel_o  : out std_logic_vector(3 downto 0);
320   s3wb_we_o   : out bit_t;
321   s3wb_cyc_o  : out bit_t;
322   s3wb_stb_o  : out bit_t;
```

```
323       s3wb_ack_i   : in  bit_t;
324       s3wb_err_i   : in  bit_t;
325       s3wb_rty_i   : in  bit_t
326       );
327     end component;
328
329     ------------------------------------------------------------------
330     --
331     --  ICON core component declaration
332     --
333     ------------------------------------------------------------------
334     component icon
335       port
336       (
337         control0    :   out std_logic_vector(35 downto 0);
338         control1    :   out std_logic_vector(35 downto 0)
339       );
340     end component;
341
342     ------------------------------------------------------------------
343     --
344     --  ILA core component declaration
345     --
346     ------------------------------------------------------------------
347     component ila
348       port
349       (
350         control     : in    std_logic_vector(35 downto 0);
351         clk         : in    std_logic;
352         data        : in    std_logic_vector(97 downto 0);
353         trig0       : in    std_logic_vector(31 downto 0)
354       );
355     end component;
356
357
358     ------------------------------------------------------------------
```

```
359 --
360 --  VIO core component declaration
361 --
362 ------------------------------------------------------------
363 component vio
364   port
365   (
366     control      : in    std_logic_vector(35 downto 0);
367     async_out    :   out std_logic_vector(7 downto 0)
368   );
369 end component;
370
371 component BUFG
372   port (
373     O : out STD_ULOGIC;
374     I : in STD_ULOGIC);
375 end component;
376
377 component IBUFG
378   port (
379     O : out STD_ULOGIC;
380     I : in STD_ULOGIC);
381 end component;
382
383
384
385 --
386 --SIGNALS
387 --
388
389 --Clock signals
390 --  SIGNAL clk     : bit_t;
391 SIGNAL clk_div     : bit_t;
392 SIGNAL clk_12_5MHZ  : bit_t;
393 SIGNAL clkcount    : unsigned(31 downto 0) := "00000000000000000000000000000000";
394
```

```
395
396     SIGNAL reset    : bit_t;
397     SIGNAL reset_inv  : bit_t;
398     SIGNAL pic_ints   : std_logic_vector(19 downto 0);
399     SIGNAL clmode   : std_logic_vector(1 downto 0);
400
401
402     -- Master 0 Interface
403     SIGNAL  m0_dat_i, m0_dat_o, m0_adr_o  : word_t;
404     SIGNAL  m0_sel_o  : std_logic_vector(3 downto 0);
405     SIGNAL  m0_we_o, m0_cyc_o,m0_stb_o, m0_ack_i, m0_err_i, m0_rty_i, m0_cab_o  : bit_t;
406
407     -- Master 1 Interface
408     SIGNAL  m1_dat_i, m1_dat_o, m1_adr_o  : word_t;
409     SIGNAL  m1_sel_o  : std_logic_vector(3 downto 0);
410     SIGNAL  m1_we_o, m1_cyc_o,m1_stb_o, m1_ack_i, m1_err_i, m1_rty_i, m1_cab_o  : bit_t;
411
412     -- Master 2 Interface
413     SIGNAL  m2_dat_i, m2_dat_o, m2_adr_o  : word_t;
414     SIGNAL  m2_sel_o  : std_logic_vector(3 downto 0);
415     SIGNAL  m2_we_o, m2_cyc_o,m2_stb_o, m2_ack_i, m2_err_i, m2_rty_i, m2_cab_o  : bit_t;
416
417     -- Master 3 Interface
418     SIGNAL  m3_dat_i, m3_dat_o, m3_adr_o  : word_t;
419     SIGNAL  m3_sel_o  : std_logic_vector(3 downto 0);
420     SIGNAL  m3_we_o, m3_cyc_o,m3_stb_o, m3_ack_i, m3_err_i, m3_rty_i, m3_cab_o  : bit_t;
421
422     -- Master 4 Interface
423     SIGNAL  m4_dat_i, m4_dat_o, m4_adr_o  : word_t;
424     SIGNAL  m4_sel_o  : std_logic_vector(3 downto 0);
425     SIGNAL  m4_we_o, m4_cyc_o,m4_stb_o, m4_ack_i, m4_err_i, m4_rty_i, m4_cab_o  : bit_t;
426
427     -- Master 5 Interface
428     SIGNAL  m5_dat_i, m5_dat_o, m5_adr_o  : word_t;
429     SIGNAL  m5_sel_o  : std_logic_vector(3 downto 0);
430     SIGNAL  m5_we_o, m5_cyc_o,m5_stb_o, m5_ack_i, m5_err_i, m5_rty_i, m5_cab_o  : bit_t;
```

```
431
432  -- Master 6 Interface
433  SIGNAL  m6_dat_i, m6_dat_o, m6_adr_o  : word_t;
434  SIGNAL  m6_sel_o  : std_logic_vector(3 downto 0);
435  SIGNAL  m6_we_o, m6_cyc_o,m6_stb_o, m6_ack_i, m6_err_i, m6_rty_i, m6_cab_o  : bit_t;
436
437  -- Master 7 Interface
438  SIGNAL  m7_dat_i, m7_dat_o, m7_adr_o  : word_t;
439  SIGNAL  m7_sel_o  : std_logic_vector(3 downto 0);
440  SIGNAL  m7_we_o, m7_cyc_o,m7_stb_o, m7_ack_i, m7_err_i, m7_rty_i, m7_cab_o  : bit_t;
441
442  -- Slaver 0 Interface
443  SIGNAL  s0_dat_i, s0_dat_o, s0_adr_i :word_t;
444  SIGNAL  s0_sel_i  : std_logic_vector(3 downto 0);
445  SIGNAL  s0_we_i, s0_cyc_i,s0_stb_i, s0_ack_o, s0_err_o, s0_rty_o, s0_cab_i  : bit_t;
446
447  -- Slaver 1 Interface
448  SIGNAL  s1_dat_i, s1_dat_o, s1_adr_i :word_t;
449  SIGNAL  s1_sel_i  : std_logic_vector(3 downto 0);
450  SIGNAL  s1_we_i, s1_cyc_i,s1_stb_i, s1_ack_o, s1_err_o, s1_rty_o, s1_cab_i  : bit_t;
451
452  -- Slaver 2 Interface
453  SIGNAL  s2_dat_i, s2_dat_o, s2_adr_i :word_t;
454  SIGNAL  s2_sel_i  : std_logic_vector(3 downto 0);
455  SIGNAL  s2_we_i, s2_cyc_i,s2_stb_i, s2_ack_o, s2_err_o, s2_rty_o, s2_cab_i  : bit_t;
456
457  -- Slaver 3 Interface
458  SIGNAL  s3_dat_i, s3_dat_o, s3_adr_i :word_t;
459  SIGNAL  s3_sel_i  : std_logic_vector(3 downto 0);
460  SIGNAL  s3_we_i, s3_cyc_i,s3_stb_i, s3_ack_o, s3_err_o, s3_rty_o, s3_cab_i  : bit_t;
461
462  -- Slaver 4 Interface
463  SIGNAL  s4_dat_i, s4_dat_o, s4_adr_i :word_t;
464  SIGNAL  s4_sel_i  : std_logic_vector(3 downto 0);
465  SIGNAL  s4_we_i, s4_cyc_i,s4_stb_i, s4_ack_o, s4_err_o, s4_rty_o, s4_cab_i  : bit_t;
466
```

```
467    -- Slaver 5 Interface
468    SIGNAL  s5_dat_i, s5_dat_o, s5_adr_i :word_t;
469    SIGNAL  s5_sel_i  : std_logic_vector(3 downto 0);
470    SIGNAL  s5_we_i, s5_cyc_i,s5_stb_i, s5_ack_o, s5_err_o, s5_rty_o, s5_cab_i  : bit_t;
471
472    -- Slaver 6 Interface
473    SIGNAL  s6_dat_i, s6_dat_o, s6_adr_i :word_t;
474    SIGNAL  s6_sel_i  : std_logic_vector(3 downto 0);
475    SIGNAL  s6_we_i, s6_cyc_i,s6_stb_i, s6_ack_o, s6_err_o, s6_rty_o, s6_cab_i  : bit_t;
476
477    -- Slaver 7 Interface
478    SIGNAL  s7_dat_i, s7_dat_o, s7_adr_i :word_t;
479    SIGNAL  s7_sel_i  : std_logic_vector(3 downto 0);
480    SIGNAL  s7_we_i, s7_cyc_i,s7_stb_i, s7_ack_o, s7_err_o, s7_rty_o, s7_cab_i  : bit_t;
481
482    --ack signals
483  --  SIGNAL ack0 : bit_t;
484  --  SIGNAL ack1 : bit_t;
485  --  SIGNAL ack2 : bit_t;
486  --  SIGNAL ack3 : bit_t;
487  --  SIGNAL ack4 : bit_t;
488  --  SIGNAL ack5 : bit_t;
489  --  SIGNAL ack6 : bit_t;
490  --  SIGNAL ack7 : bit_t;
491
492
493
494    --Zero signals
495    SIGNAL zero   : bit_t;
496    SIGNAL zero32   : word_t;
497
498    --Serial signal
499    SIGNAL tx_pad   : bit_t;
500
501    ------------------------------------------------------------------
502    --
```

```
503   -- ILA core signal declarations
504   --
505   ------------------------------------------------------------------
506   signal control0    : std_logic_vector(35 downto 0);
507   signal clk         : std_logic;
508   signal data        : std_logic_vector(97 downto 0);
509   signal trig0       : std_logic_vector(31 downto 0);
510
511   ------------------------------------------------------------------
512   --
513   -- VIO core signal declarations
514   --
515   ------------------------------------------------------------------
516   signal control1    : std_logic_vector(35 downto 0);
517   signal async_out   : std_logic_vector(7 downto 0);
518
519   ------------------------------------------------------------------
520   --
521   -- ICON core signal declarations
522   --
523   ------------------------------------------------------------------
524   -- signal control0       : std_logic_vector(35 downto 0);
525   -- signal control1       : std_logic_vector(35 downto 0);
526
527
528   begin
529     Clock_buf : BUFG
530     port map (O => clk_12_5MHZ,
531     I => clk_div);
532
533     rst_buf: IBUFG
534     port map (O => reset,
535     I => rst_i);
536
537     process(clk_i)
538     begin
```

```
539       if clk_i'event and clk_i = '1' then
540         clkcount <= clkcount + 1;
541       end if;
542     end process;
543
544     clk_div <= conv_std_logic_vector(clkcount, 3)(2);
545     running <= conv_std_logic_vector(clkcount, 24)(23) and reset;
546
547     zero32      <= "00000000000000000000000000000000";
548     zero      <= '0';
549     pic_ints      <= "00000000000000000000";
550     clmode        <= "00";                    -- Same clock for WISHBONE and CPU
551
552     reset_inv     <= not reset;
553
554
555     CPU0: or1200_top
556       PORT MAP(
557         clk_i   => clk_12_5MHZ,
558         rst_i   => reset_inv,
559         pic_ints_i  => pic_ints,
560         clmode_i  => clmode,
561
562         -- Instruction WISHBONE interface
563         iwb_clk_i => clk_12_5MHZ,
564         iwb_rst_i => reset_inv,
565         iwb_ack_i => m0_ack_i,
566         iwb_err_i => m0_err_i,
567         iwb_rty_i => m0_rty_i,
568         iwb_dat_i => m0_dat_i,
569               iwb_cyc_o => m0_cyc_o,
570         iwb_adr_o => m0_adr_o,
571         iwb_stb_o => m0_stb_o,
572         iwb_we_o  => m0_we_o,
573         iwb_sel_o => m0_sel_o,
574         iwb_dat_o => m0_dat_o,
```

```
575         iwb_cab_o => m0_cab_o,
576
577         -- Data WISHBONE interface
578         dwb_clk_i => clk_12_5MHZ,
579         dwb_rst_i => reset_inv,
580         dwb_ack_i => m1_ack_i,
581         dwb_err_i => m1_err_i,
582         dwb_rty_i => m1_rty_i,
583         dwb_dat_i => m1_dat_i,
584         dwb_cyc_o => m1_cyc_o,
585         dwb_adr_o => m1_adr_o,
586         dwb_stb_o => m1_stb_o,
587         dwb_we_o  => m1_we_o,
588         dwb_sel_o => m1_sel_o,
589         dwb_dat_o => m1_dat_o,
590         dwb_cab_o => m1_cab_o,
591
592         -- Debug interface
593         dbg_stall_i => zero,
594         dbg_ewt_i => zero,
595         dbg_lss_o => open,
596         dbg_is_o  => open,
597         dbg_wp_o  => open,
598         dbg_bp_o  => open,
599         dbg_stb_i => zero,
600         dbg_we_i  => zero,
601         dbg_adr_i => zero32,
602         dbg_dat_i => zero32,
603         dbg_dat_o => open,
604         dbg_ack_o => open,
605
606         -- Power Management interface
607         pm_cpustall_i => zero,
608         pm_clksd_o  => open,
609         pm_dc_gate_o  => open,
610         pm_ic_gate_o  => open,
```

```
611       pm_dmmu_gate_o  => open,
612       pm_immu_gate_o  => open,
613       pm_tt_gate_o  => open,
614       pm_cpu_gate_o => open,
615       pm_wakeup_o => open,
616       pm_lvolt_o  => open
617     );
618
619   CPU1: or1200_top
620     PORT MAP(
621       clk_i   => clk_12_5MHZ,
622       rst_i   => reset_inv,
623       pic_ints_i  => pic_ints,
624       clmode_i  => clmode,
625
626       -- Instruction WISHBONE interface
627       iwb_clk_i => clk_12_5MHZ,
628       iwb_rst_i => reset_inv,
629       iwb_ack_i => m2_ack_i,
630       iwb_err_i => m2_err_i,
631       iwb_rty_i => m2_rty_i,
632       iwb_dat_i => m2_dat_i,
633             iwb_cyc_o => m2_cyc_o,
634       iwb_adr_o => m2_adr_o,
635       iwb_stb_o => m2_stb_o,
636       iwb_we_o  => m2_we_o,
637       iwb_sel_o => m2_sel_o,
638       iwb_dat_o => m2_dat_o,
639       iwb_cab_o => m2_cab_o,
640
641       -- Data WISHBONE interface
642       dwb_clk_i => clk_12_5MHZ,
643       dwb_rst_i => reset_inv,
644       dwb_ack_i => m3_ack_i,
645       dwb_err_i => m3_err_i,
646       dwb_rty_i => m3_rty_i,
```

```
647   dwb_dat_i => m3_dat_i,
648   dwb_cyc_o => m3_cyc_o,
649   dwb_adr_o => m3_adr_o,
650   dwb_stb_o => m3_stb_o,
651   dwb_we_o  => m3_we_o,
652   dwb_sel_o => m3_sel_o,
653   dwb_dat_o => m3_dat_o,
654   dwb_cab_o => m3_cab_o,
655
656   -- Debug interface
657   dbg_stall_i => zero,
658   dbg_ewt_i => zero,
659   dbg_lss_o => open,
660   dbg_is_o  => open,
661   dbg_wp_o  => open,
662   dbg_bp_o  => open,
663   dbg_stb_i => zero,
664   dbg_we_i  => zero,
665   dbg_adr_i => zero32,
666   dbg_dat_i => zero32,
667   dbg_dat_o => open,
668   dbg_ack_o => open,
669
670   -- Power Management interface
671   pm_cpustall_i => zero,
672   pm_clksd_o  => open,
673   pm_dc_gate_o  => open,
674   pm_ic_gate_o  => open,
675   pm_dmmu_gate_o  => open,
676   pm_immu_gate_o  => open,
677   pm_tt_gate_o  => open,
678   pm_cpu_gate_o => open,
679   pm_wakeup_o => open,
680   pm_lvolt_o  => open
681 );
682
```

```
683   CPU2: or1200_top
684     PORT MAP(
685       clk_i   => clk_12_5MHZ,
686       rst_i   => reset_inv,
687       pic_ints_i  => pic_ints,
688       clmode_i  => clmode,
689
690       -- Instruction WISHBONE interface
691       iwb_clk_i => clk_12_5MHZ,
692       iwb_rst_i => reset_inv,
693       iwb_ack_i => m4_ack_i,
694       iwb_err_i => m4_err_i,
695       iwb_rty_i => m4_rty_i,
696       iwb_dat_i => m4_dat_i,
697             iwb_cyc_o => m4_cyc_o,
698       iwb_adr_o => m4_adr_o,
699       iwb_stb_o => m4_stb_o,
700       iwb_we_o  => m4_we_o,
701       iwb_sel_o => m4_sel_o,
702       iwb_dat_o => m4_dat_o,
703       iwb_cab_o => m4_cab_o,
704
705       -- Data WISHBONE interface
706       dwb_clk_i => clk_12_5MHZ,
707       dwb_rst_i => reset_inv,
708       dwb_ack_i => m5_ack_i,
709       dwb_err_i => m5_err_i,
710       dwb_rty_i => m5_rty_i,
711       dwb_dat_i => m5_dat_i,
712       dwb_cyc_o => m5_cyc_o,
713       dwb_adr_o => m5_adr_o,
714       dwb_stb_o => m5_stb_o,
715       dwb_we_o  => m5_we_o,
716       dwb_sel_o => m5_sel_o,
717       dwb_dat_o => m5_dat_o,
718       dwb_cab_o => m5_cab_o,
```

```
719
720         -- Debug interface
721         dbg_stall_i => zero,
722         dbg_ewt_i => zero,
723         dbg_lss_o => open,
724         dbg_is_o  => open,
725         dbg_wp_o  => open,
726         dbg_bp_o  => open,
727         dbg_stb_i => zero,
728         dbg_we_i  => zero,
729         dbg_adr_i => zero32,
730         dbg_dat_i => zero32,
731         dbg_dat_o => open,
732         dbg_ack_o => open,
733
734         -- Power Management interface
735         pm_cpustall_i => zero,
736         pm_clksd_o  => open,
737         pm_dc_gate_o  => open,
738         pm_ic_gate_o  => open,
739         pm_dmmu_gate_o  => open,
740         pm_immu_gate_o  => open,
741         pm_tt_gate_o  => open,
742         pm_cpu_gate_o => open,
743         pm_wakeup_o => open,
744         pm_lvolt_o  => open
745       );
746
747   --  CPU3: or1200_top
748   --    PORT MAP(
749   --      clk_i   => clk_12_5MHZ,
750   --      rst_i   => reset_inv,
751   --      pic_ints_i  => pic_ints,
752   --      clmode_i  => clmode,
753   --
754   --      -- Instruction WISHBONE interface
```

```
755 --    iwb_clk_i => clk_12_5MHZ,
756 --    iwb_rst_i => reset_inv,
757 --    iwb_ack_i => m6_ack_i,
758 --    iwb_err_i => m6_err_i,
759 --    iwb_rty_i => m6_rty_i,
760 --    iwb_dat_i => m6_dat_i,
761 --    iwb_cyc_o => m6_cyc_o,
762 --    iwb_adr_o => m6_adr_o,
763 --    iwb_stb_o => m6_stb_o,
764 --    iwb_we_o  => m6_we_o,
765 --    iwb_sel_o => m6_sel_o,
766 --    iwb_dat_o => m6_dat_o,
767 --    iwb_cab_o => m6_cab_o,
768 --
769 --    -- Data WISHBONE interface
770 --    dwb_clk_i => clk_12_5MHZ,
771 --    dwb_rst_i => reset_inv,
772 --    dwb_ack_i => m7_ack_i,
773 --    dwb_err_i => m7_err_i,
774 --    dwb_rty_i => m7_rty_i,
775 --    dwb_dat_i => m7_dat_i,
776 --    dwb_cyc_o => m7_cyc_o,
777 --    dwb_adr_o => m7_adr_o,
778 --    dwb_stb_o => m7_stb_o,
779 --    dwb_we_o  => m7_we_o,
780 --    dwb_sel_o => m7_sel_o,
781 --    dwb_dat_o => m7_dat_o,
782 --    dwb_cab_o => m7_cab_o,
783 --
784 --    -- Debug interface
785 --    dbg_stall_i => zero,
786 --    dbg_ewt_i => zero,
787 --    dbg_lss_o => open,
788 --    dbg_is_o  => open,
789 --    dbg_wp_o  => open,
790 --    dbg_bp_o  => open,
```

```
791 --    dbg_stb_i => zero,
792 --    dbg_we_i  => zero,
793 --    dbg_adr_i => zero32,
794 --    dbg_dat_i => zero32,
795 --    dbg_dat_o => open,
796 --    dbg_ack_o => open,
797 --
798 --    -- Power Management interface
799 --    pm_cpustall_i => zero,
800 --    pm_clksd_o    => open,
801 --    pm_dc_gate_o  => open,
802 --    pm_ic_gate_o  => open,
803 --    pm_dmmu_gate_o  => open,
804 --    pm_immu_gate_o  => open,
805 --    pm_tt_gate_o  => open,
806 --    pm_cpu_gate_o => open,
807 --    pm_wakeup_o => open,
808 --    pm_lvolt_o  => open
809 --    );
810
811     imem: core_mem
812       port map (
813         --I/O ports
814         wb_clk_i  => clk_12_5MHZ,
815         wb_rst_i  => reset,
816
817         --WB slave i/f
818         wb_data_i => s0_dat_i,
819         wb_data_o => s0_dat_o,
820         wb_adr_i  => s0_adr_i,
821         wb_sel_i  => s0_sel_i,
822         wb_we_i   => s0_we_i,
823         wb_cyc_i  => s0_cyc_i,
824         wb_stb_i  => s0_stb_i,
825         wb_ack_o  => s0_ack_o,
826         wb_err_o  => s0_err_o
```

```
827   );
828
829
830   dmem: core_mem
831     port map (
832       --I/O ports
833       wb_clk_i  => clk_12_5MHZ,
834       wb_rst_i  => reset,
835
836       --WB slave i/f
837       wb_data_i => s1_dat_i,
838       wb_data_o => s1_dat_o,
839       wb_adr_i  => s1_adr_i,
840       wb_sel_i  => s1_sel_i,
841       wb_we_i   => s1_we_i,
842       wb_cyc_i  => s1_cyc_i,
843       wb_stb_i  => s1_stb_i,
844       wb_ack_o  => s1_ack_o,
845       wb_err_o  => s1_err_o
846     );
847
848   sema: semaphore
849     port map (
850       --I/O ports
851       wb_clk_i  => clk_12_5MHZ,
852       wb_rst_i  => reset,
853
854       --WB slave i/f
855       wb_data_i => s3_dat_i,
856       wb_data_o => s3_dat_o,
857       wb_adr_i  => s3_adr_i,
858       wb_sel_i  => s3_sel_i,
859       wb_we_i   => s3_we_i,
860       wb_cyc_i  => s3_cyc_i,
861       wb_stb_i  => s3_stb_i,
862       wb_ack_o  => s3_ack_o,
```

```
863       wb_err_o  => s3_err_o
864     );
865
866
867     uart : uart_top
868       port map (
869         --I/O Ports
870         wb_clk_i  => clk_12_5MHZ,
871         wb_rst_i  => reset_inv,
872
873         --WB slave i/f
874         wb_adr_i  => s2_adr_i(4 downto 0),
875         wb_dat_i  => s2_dat_i,
876         wb_dat_o  => s2_dat_o,
877         wb_we_i   => s2_we_i,
878         wb_stb_i  => s2_stb_i,
879         wb_cyc_i  => s2_cyc_i,
880         wb_ack_o  => s2_ack_o,
881         wb_sel_i  => s2_sel_i,
882
883         --UART signals
884
885         int_o     => open,
886
887         --serial input/output
888         stx_pad_o => tx_pad,
889         srx_pad_i => rx,
890
891         --modem signals
892         rts_pad_o => open,
893         cts_pad_i => '1',
894         dtr_pad_o => open,
895         dsr_pad_i => '1',
896         ri_pad_i  => '1',
897         dcd_pad_i => '1'
898
```

```
899       );
900
901
902     tree : noc_tree
903     port map (
904
905       --I/O ports
906       wb_clk_i     => clk_12_5MHZ,
907       wb_rst_i     => reset,
908       --Master 0
909       m0wb_dat_i   => m0_dat_o,
910       m0wb_dat_o   => m0_dat_i,
911       m0wb_adr_i   => m0_adr_o,
912       m0wb_sel_i   => m0_sel_o,
913       m0wb_we_i    => m0_we_o,
914       m0wb_cyc_i   => m0_cyc_o,
915       m0wb_stb_i   => m0_stb_o,
916       m0wb_ack_o   => m0_ack_i,
917       m0wb_err_o   => open, -- m0_err_i,
918       m0wb_rty_o   => open, --m0_rty_i,
919
920       --Master 1
921       m1wb_dat_i   => m1_dat_o,
922       m1wb_dat_o   => m1_dat_i,
923       m1wb_adr_i   => m1_adr_o,
924       m1wb_sel_i   => m1_sel_o,
925       m1wb_we_i    => m1_we_o,
926       m1wb_cyc_i   => m1_cyc_o,
927       m1wb_stb_i   => m1_stb_o,
928       m1wb_ack_o   => m1_ack_i,
929       m1wb_err_o   => open, --m1_err_i,
930       m1wb_rty_o   => open, --m1_rty_i,
931
932       --Master 2
933       m2wb_dat_i   => m2_dat_o,
934       m2wb_dat_o   => m2_dat_i,
```

```
935   m2wb_adr_i  => m2_adr_o,
936   m2wb_sel_i  => m2_sel_o,
937   m2wb_we_i   => m2_we_o,
938   m2wb_cyc_i  => m2_cyc_o,
939   m2wb_stb_i  => m2_stb_o,
940   m2wb_ack_o  => m2_ack_i,
941   m2wb_err_o  => open, --m2_err_i,
942   m2wb_rty_o  => open, --m2_rty_i,
943
944   --Master 3
945   m3wb_dat_i  => m3_dat_o,
946   m3wb_dat_o  => m3_dat_i,
947   m3wb_adr_i  => m3_adr_o,
948   m3wb_sel_i  => m3_sel_o,
949   m3wb_we_i   => m3_we_o,
950   m3wb_cyc_i  => m3_cyc_o,
951   m3wb_stb_i  => m3_stb_o,
952   m3wb_ack_o  => m3_ack_i,
953   m3wb_err_o  => open, --m3_err_i,
954   m3wb_rty_o  => open, --m3_rty_i,
955
956   --Master 4
957   m4wb_dat_i  => m4_dat_o,
958   m4wb_dat_o  => m4_dat_i,
959   m4wb_adr_i  => m4_adr_o,
960   m4wb_sel_i  => m4_sel_o,
961   m4wb_we_i   => m4_we_o,
962   m4wb_cyc_i  => m4_cyc_o,
963   m4wb_stb_i  => m4_stb_o,
964   m4wb_ack_o  => m4_ack_i,
965   m4wb_err_o  => open, --m4_err_i,
966   m4wb_rty_o  => open, --m4_rty_i,
967
968   --Master 5
969   m5wb_dat_i  => m5_dat_o,
970   m5wb_dat_o  => m5_dat_i,
```

```
971       m5wb_adr_i  => m5_adr_o,
972       m5wb_sel_i  => m5_sel_o,
973       m5wb_we_i   => m5_we_o,
974       m5wb_cyc_i  => m5_cyc_o,
975       m5wb_stb_i  => m5_stb_o,
976       m5wb_ack_o  => m5_ack_i,
977       m5wb_err_o  => open, --m5_err_i,
978       m5wb_rty_o  => open, --m5_rty_i,
979
980       --Master 6
981       m6wb_dat_i  => m6_dat_o,
982       m6wb_dat_o  => m6_dat_i,
983       m6wb_adr_i  => m6_adr_o,
984       m6wb_sel_i  => m6_sel_o,
985       m6wb_we_i   => m6_we_o,
986       m6wb_cyc_i  => m6_cyc_o,
987       m6wb_stb_i  => m6_stb_o,
988       m6wb_ack_o  => m6_ack_i,
989       m6wb_err_o  => open, --m6_err_i,
990       m6wb_rty_o  => open, --m6_rty_i,
991
992       --Master 7
993       m7wb_dat_i  => m7_dat_o,
994       m7wb_dat_o  => m7_dat_i,
995       m7wb_adr_i  => m7_adr_o,
996       m7wb_sel_i  => m7_sel_o,
997       m7wb_we_i   => m7_we_o,
998       m7wb_cyc_i  => m7_cyc_o,
999       m7wb_stb_i  => m7_stb_o,
1000      m7wb_ack_o  => m7_ack_i,
1001      m7wb_err_o  => open, --m7_err_i,
1002      m7wb_rty_o  => open, --m7_rty_i,
1003
1004      --Slave 0
1005      s0wb_dat_i  => s0_dat_o,
1006      s0wb_dat_o  => s0_dat_i,
```

```
1007  s0wb_adr_o  => s0_adr_i,
1008  s0wb_sel_o  => s0_sel_i,
1009  s0wb_we_o => s0_we_i,
1010  s0wb_cyc_o  => s0_cyc_i,
1011  s0wb_stb_o  => s0_stb_i,
1012  s0wb_ack_i  => s0_ack_o,
1013  s0wb_err_i  => s0_err_o,
1014  s0wb_rty_i  => s0_rty_o,
1015
1016  --Slave 1
1017  s1wb_dat_i  => s1_dat_o,
1018  s1wb_dat_o  => s1_dat_i,
1019  s1wb_adr_o  => s1_adr_i,
1020  s1wb_sel_o  => s1_sel_i,
1021  s1wb_we_o => s1_we_i,
1022  s1wb_cyc_o  => s1_cyc_i,
1023  s1wb_stb_o  => s1_stb_i,
1024  s1wb_ack_i  => s1_ack_o,
1025  s1wb_err_i  => s1_err_o,
1026  s1wb_rty_i  => s1_rty_o,
1027
1028  --Slave 2
1029  s2wb_dat_i  => s2_dat_o,
1030  s2wb_dat_o  => s2_dat_i,
1031  s2wb_adr_o  => s2_adr_i,
1032  s2wb_sel_o  => s2_sel_i,
1033  s2wb_we_o => s2_we_i,
1034  s2wb_cyc_o  => s2_cyc_i,
1035  s2wb_stb_o  => s2_stb_i,
1036  s2wb_ack_i  => s2_ack_o,
1037  s2wb_err_i  => s2_err_o,
1038  s2wb_rty_i  => s2_rty_o,
1039
1040  --Slave 3
1041  s3wb_dat_i  => s3_dat_o,
1042  s3wb_dat_o  => s3_dat_i,
```

```
1043      s3wb_adr_o  => s3_adr_i,
1044      s3wb_sel_o  => s3_sel_i,
1045      s3wb_we_o => s3_we_i,
1046      s3wb_cyc_o  => s3_cyc_i,
1047      s3wb_stb_o  => s3_stb_i,
1048      s3wb_ack_i  => s3_ack_o,
1049      s3wb_err_i  => s3_err_o,
1050      s3wb_rty_i  => s3_rty_o
1051
1052      );
1053
1054    ------------------------------------------------------------------
1055    --
1056    --  ICON core instance
1057    --
1058    ------------------------------------------------------------------
1059    i_icon : icon
1060      port map
1061      (
1062        control0    => control0,
1063        control1    => control1
1064      );
1065
1066  ------------------------------------------------------------------
1067    --
1068    --  ILA core instance
1069    --
1070    ------------------------------------------------------------------
1071    i_ila : ila
1072      port map
1073      (
1074        control   => control0,
1075        clk       => clk_12_5MHZ,
1076        data      => data,
1077        trig0     => m0_adr_o
1078      );
```

```
1079 ------------------------------------------------------------------
1080
1081 --
1082 --  VIO core instance
1083 --
1084 ------------------------------------------------------------------
1085 i_vio : vio
1086   port map
1087   (
1088     control   => control1,
1089     async_out => async_out
1090   );
1091
1092   --Set unconected cyc signals
1093 --  m0_cyc_o <= '0';
1094 --  m1_cyc_o <= '0';
1095 --  m2_cyc_o <= '0';
1096 --  m3_cyc_o <= '0';
1097 --  m4_cyc_o <= '0';
1098 --  m5_cyc_o <= '0';
1099 --  m6_cyc_o <= '0';
1100 --  m7_cyc_o <= '0';
1101
1102   --Set unconected err signals
1103   m0_err_i  <= '0';
1104   m1_err_i  <= '0';
1105   m2_err_i  <= '0';
1106   m3_err_i  <= '0';
1107   m4_err_i  <= '0';
1108   m5_err_i  <= '0';
1109   m6_err_i  <= '0';
1110   m7_err_i  <= '0';
1111
1112   --Set unconected rty signals
1113   m0_rty_i  <= '0';
1114   m1_rty_i  <= '0';
```

```
1115    m2_rty_i  <= '0';
1116    m3_rty_i  <= '0';
1117    m4_rty_i  <= '0';
1118    m5_rty_i  <= '0';
1119    m6_rty_i  <= '0';
1120    m7_rty_i  <= '0';
1121    s0_rty_o  <= '0';
1122    s1_rty_o  <= '0';
1123    s2_rty_o  <= '0';
1124    s3_rty_o  <= '0';
1125
1126
1127
1128      data <= "00000000000000000000000000000000" & m1_adr_o & m1_dat_o &  m1_ack_i & m1_stb_o & tx_pad & rx
             ;
1129      tx <= tx_pad;
1130
1131    end struc;
```

A.6 Stree arcitechture

A.6.1 noc_stree.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Table for stree NOC
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 22/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity noc_stree is
22
23     port(
24     --I/O Ports
25     wb_clk_i     : in bit_t;    -- The Clock
26     wb_rst_i     : in bit_t;    -- The reset signal
27
28 --WishBone Master 0
29     m0wb_dat_i     : in word_t;
30     m0wb_dat_o     : out word_t;
```

```
31      m0wb_ack_o    : out bit_t;
32      m0wb_adr_i    : in word_t;
33      m0wb_cyc_i    : in bit_t;
34      m0wb_stb_i    : in bit_t;
35      m0wb_err_o    : out bit_t;
36      m0wb_rty_o    : out bit_t;
37      m0wb_sel_i    : in std_logic_vector(3 downto 0);
38      m0wb_we_i   : in bit_t;
39
40      --WishBone Master 1
41      m1wb_dat_i    : in word_t;
42      m1wb_dat_o    : out word_t;
43      m1wb_ack_o    : out bit_t;
44      m1wb_adr_i    : in word_t;
45      m1wb_cyc_i    : in bit_t;
46      m1wb_stb_i    : in bit_t;
47      m1wb_err_o    : out bit_t;
48      m1wb_rty_o    : out bit_t;
49      m1wb_sel_i    : in std_logic_vector(3 downto 0);
50      m1wb_we_i   : in bit_t;
51
52      --WishBone Master 2
53      m2wb_dat_i    : in word_t;
54      m2wb_dat_o    : out word_t;
55      m2wb_ack_o    : out bit_t;
56      m2wb_adr_i    : in word_t;
57      m2wb_cyc_i    : in bit_t;
58      m2wb_stb_i    : in bit_t;
59      m2wb_err_o    : out bit_t;
60      m2wb_rty_o    : out bit_t;
61      m2wb_sel_i    : in std_logic_vector(3 downto 0);
62      m2wb_we_i   : in bit_t;
63
64      --WishBone Master 3
65      m3wb_dat_i    : in word_t;
66      m3wb_dat_o    : out word_t;
```

```
67  m3wb_ack_o    : out bit_t;
68  m3wb_adr_i    : in word_t;
69  m3wb_cyc_i    : in bit_t;
70  m3wb_stb_i    : in bit_t;
71  m3wb_err_o    : out bit_t;
72  m3wb_rty_o    : out bit_t;
73  m3wb_sel_i    : in std_logic_vector(3 downto 0);
74  m3wb_we_i     : in bit_t;
75
76  --WishBone Master 4
77  m4wb_dat_i    : in word_t;
78  m4wb_dat_o    : out word_t;
79  m4wb_ack_o    : out bit_t;
80  m4wb_adr_i    : in word_t;
81  m4wb_cyc_i    : in bit_t;
82  m4wb_stb_i    : in bit_t;
83  m4wb_err_o    : out bit_t;
84  m4wb_rty_o    : out bit_t;
85  m4wb_sel_i    : in std_logic_vector(3 downto 0);
86  m4wb_we_i     : in bit_t;
87
88  --WishBone Master 5
89  m5wb_dat_i    : in word_t;
90  m5wb_dat_o    : out word_t;
91  m5wb_ack_o    : out bit_t;
92  m5wb_adr_i    : in word_t;
93  m5wb_cyc_i    : in bit_t;
94  m5wb_stb_i    : in bit_t;
95  m5wb_err_o    : out bit_t;
96  m5wb_rty_o    : out bit_t;
97  m5wb_sel_i    : in std_logic_vector(3 downto 0);
98  m5wb_we_i     : in bit_t;
99
100 --WishBone Master 6
101 m6wb_dat_i    : in word_t;
102 m6wb_dat_o    : out word_t;
```

```
103     m6wb_ack_o    : out bit_t;
104     m6wb_adr_i    : in word_t;
105     m6wb_cyc_i    : in bit_t;
106     m6wb_stb_i    : in bit_t;
107     m6wb_err_o    : out bit_t;
108     m6wb_rty_o    : out bit_t;
109     m6wb_sel_i    : in std_logic_vector(3 downto 0);
110     m6wb_we_i   : in bit_t;
111
112     --WishBone Master 7
113     m7wb_dat_i    : in word_t;
114     m7wb_dat_o    : out word_t;
115     m7wb_ack_o    : out bit_t;
116     m7wb_adr_i    : in word_t;
117     m7wb_cyc_i    : in bit_t;
118     m7wb_stb_i    : in bit_t;
119     m7wb_err_o    : out bit_t;
120     m7wb_rty_o    : out bit_t;
121     m7wb_sel_i    : in std_logic_vector(3 downto 0);
122     m7wb_we_i   : in bit_t;
123
124     --WishBone Slave 0
125     s0wb_dat_i  : in word_t;
126     s0wb_dat_o  : out word_t;
127     s0wb_ack_i  : in bit_t;
128     s0wb_adr_o  : out word_t;
129     s0wb_cyc_o  : out bit_t;
130     s0wb_stb_o  : out bit_t;
131     s0wb_err_i  : in bit_t;
132     s0wb_rty_i  : in bit_t;
133     s0wb_sel_o  : out std_logic_vector(3 downto 0);
134     s0wb_we_o : out bit_t;
135
136     --WishBone Slave 1
137     s1wb_dat_i  : in word_t;
138     s1wb_dat_o  : out word_t;
```

```
139     s1wb_ack_i  : in bit_t;
140     s1wb_adr_o  : out word_t;
141     s1wb_cyc_o  : out bit_t;
142     s1wb_stb_o  : out bit_t;
143     s1wb_err_i  : in bit_t;
144     s1wb_rty_i  : in bit_t;
145     s1wb_sel_o  : out std_logic_vector(3 downto 0);
146     s1wb_we_o : out bit_t;
147
148     --WishBone Slave 2
149     s2wb_dat_i  : in word_t;
150     s2wb_dat_o  : out word_t;
151     s2wb_ack_i  : in bit_t;
152     s2wb_adr_o  : out word_t;
153     s2wb_cyc_o  : out bit_t;
154     s2wb_stb_o  : out bit_t;
155     s2wb_err_i  : in bit_t;
156     s2wb_rty_i  : in bit_t;
157     s2wb_sel_o  : out std_logic_vector(3 downto 0);
158     s2wb_we_o : out bit_t;
159
160     --WishBone Slave 3
161     s3wb_dat_i  : in word_t;
162     s3wb_dat_o  : out word_t;
163     s3wb_ack_i  : in bit_t;
164     s3wb_adr_o  : out word_t;
165     s3wb_cyc_o  : out bit_t;
166     s3wb_stb_o  : out bit_t;
167     s3wb_err_i  : in bit_t;
168     s3wb_rty_i  : in bit_t;
169     s3wb_sel_o  : out std_logic_vector(3 downto 0);
170     s3wb_we_o : out bit_t
171
172 --      --WishBone Slave 4
173 --      s4wb_dat_i  : in word_t;
174 --      s4wb_dat_o  : out word_t;
```

```
175  --     s4wb_ack_i  : in bit_t;
176  --     s4wb_adr_o  : out word_t;
177  --     s4wb_cyc_o  : out bit_t;
178  --     s4wb_stb_o  : out bit_t;
179  --     s4wb_err_i  : in bit_t;
180  --     s4wb_rty_i  : in bit_t;
181  --     s4wb_sel_o  : out std_logic_vector(3 downto 0);
182  --     s4wb_we_o : out bit_t;
183  --
184  --     --WishBone Slave 5
185  --     s5wb_dat_i  : in word_t;
186  --     s5wb_dat_o  : out word_t;
187  --     s5wb_ack_i  : in bit_t;
188  --     s5wb_adr_o  : out word_t;
189  --     s5wb_cyc_o  : out bit_t;
190  --     s5wb_stb_o  : out bit_t;
191  --     s5wb_err_i  : in bit_t;
192  --     s5wb_rty_i  : in bit_t;
193  --     s5wb_sel_o  : out std_logic_vector(3 downto 0);
194  --     s5wb_we_o : out bit_t;
195  --
196  --     --WishBone Slave 6
197  --     s6wb_dat_i  : in word_t;
198  --     s6wb_dat_o  : out word_t;
199  --     s6wb_ack_i  : in bit_t;
200  --     s6wb_adr_o  : out word_t;
201  --     s6wb_cyc_o  : out bit_t;
202  --     s6wb_stb_o  : out bit_t;
203  --     s6wb_err_i  : in bit_t;
204  --     s6wb_rty_i  : in bit_t;
205  --     s6wb_sel_o  : out std_logic_vector(3 downto 0);
206  --     s6wb_we_o : out bit_t;
207  --
208  --     --WishBone Slave 7
209  --     s7wb_dat_i  : in word_t;
210  --     s7wb_dat_o  : out word_t;
```

```
211 --    s7wb_ack_i   : in bit_t;
212 --    s7wb_adr_o   : out word_t;
213 --    s7wb_cyc_o   : out bit_t;
214 --    s7wb_stb_o   : out bit_t;
215 --    s7wb_err_i   : in bit_t;
216 --    s7wb_rty_i   : in bit_t;
217 --    s7wb_sel_o   : out std_logic_vector(3 downto 0);
218 --    s7wb_we_o : out bit_t
219
220     );
221
222 end noc_stree;
223
224 architecture struc of noc_stree is
225   component noc_na_slave is
226   port(
227     --Common WishBone signals
228     wb_clk_i  : in bit_t;
229     wb_rst_i  : in bit_t;
230     wb_data_i : in word_t;
231     wb_data_o : out word_t;
232
233     --WishBone Master
234     wb_ack_i  : in bit_t;
235     wb_adr_o  : out word_t;
236     wb_cyc_o  : out bit_t;
237     wb_stb_o  : out bit_t;
238     wb_err_i  : in bit_t;
239     wb_rty_i  : in bit_t;
240     wb_sel_o  : out std_logic_vector(3 downto 0);
241     wb_we_o   : out bit_t;
242
243     --NoC signals
244     noc_in    : in noc_data;
245     noc_out   : out noc_data
246         );
```

```
247   end component;
248
249   component noc_router is
250
251   port(
252     --I/O ports
253     clk_i    : in bit_t;
254     rst_i    : in bit_t;
255
256     --NoC signals
257     --North
258     noc_n_i    : in noc_data;
259     noc_n_o    : out noc_data;
260
261     --South
262     noc_s_i    : in noc_data;
263     noc_s_o    : out noc_data;
264
265     --East
266     noc_e_i    : in noc_data;
267     noc_e_o    : out noc_data;
268
269     --West
270     noc_w_i    : in noc_data;
271     noc_w_o    : out noc_data;
272
273     --IP
274     ip_i     : in noc_data;
275     ip_o     : out noc_data
276         );
277   end component;
278
279   component noc_na_master is
280
281   generic(
282     X    : integer := 0;
```

```
283       Y   : integer := 0
284     );
285
286     port(
287       --Common WishBone signals
288       wb_clk_i  : in bit_t;
289       wb_rst_i  : in bit_t;
290       wb_data_i : in word_t;
291       wb_data_o : out word_t;
292
293
294
295       --WishBone Master
296       wb_ack_o  : out bit_t;
297       wb_adr_i  : in word_t;
298       wb_cyc_i  : in bit_t;
299       wb_err_o  : out bit_t;
300       wb_rty_o  : out bit_t;
301       wb_sel_i  : in std_logic_vector(3 downto 0);
302       wb_we_i   : in bit_t;
303       wb_stb_i  : in bit_t;
304
305       --NoC signals
306       noc_in    : in noc_data;
307       noc_out   : out noc_data
308           );
309     end component;
310
311     --NOC signals
312     --Masters to r1
313     SIGNAL  m0r10     : noc_data;
314     SIGNAL  m1r10     : noc_data;
315     SIGNAL  m2r10     : noc_data;
316     SIGNAL  m3r10     : noc_data;
317     SIGNAL  m4r11     : noc_data;
318     SIGNAL  m5r11     : noc_data;
```

```
319  SIGNAL  m6r11    : noc_data;
320  SIGNAL  m7r11    : noc_data;
321  --r1 to master
322  SIGNAL  r10m0    : noc_data;
323  SIGNAL  r10m1    : noc_data;
324  SIGNAL  r10m2    : noc_data;
325  SIGNAL  r10m3    : noc_data;
326  SIGNAL  r11m4    : noc_data;
327  SIGNAL  r11m5    : noc_data;
328  SIGNAL  r11m6    : noc_data;
329  SIGNAL  r11m7    : noc_data;
330  --r1 to r2
331  SIGNAL  r10r20     : noc_data;
332  SIGNAL  r11r20     : noc_data;
333  --r2 to r1
334  SIGNAL  r20r10     : noc_data;
335  SIGNAL  r20r11     : noc_data;
336  --r2 to r3
337  SIGNAL  r20r30     : noc_data;
338  --r3 to r2
339  SIGNAL  r30r20     : noc_data;
340  --r3 to slaves
341  SIGNAL  r30s0    : noc_data;
342  SIGNAL  r30s1    : noc_data;
343  SIGNAL  r30s2    : noc_data;
344  SIGNAL  r30s3    : noc_data;
345  --slaves to r3
346  SIGNAL  s0r30    : noc_data;
347  SIGNAL  s1r30    : noc_data;
348  SIGNAL  s2r30    : noc_data;
349  SIGNAL  s3r30    : noc_data;
350
351  --
352  SIGNAL nc        : noc_data; --No connection signal
353  begin
354
```

```
355   nc   <= (OTHERS => '0');
356
357   --Master connections
358
359   m0 : noc_na_master
360     port map(
361     --Common WishBone signals
362     wb_clk_i  => wb_clk_i,
363     wb_rst_i  => wb_rst_i,
364     wb_data_i => m0wb_dat_i,
365     wb_data_o => m0wb_dat_o,
366
367     --WishBone Master
368     wb_ack_o  => m0wb_ack_o,
369     wb_adr_i  => m0wb_adr_i,
370     wb_cyc_i  => m0wb_cyc_i,
371     wb_err_o  => m0wb_err_o,
372     wb_rty_o  => m0wb_rty_o,
373     wb_sel_i  => m0wb_sel_i,
374     wb_we_i   => m0wb_we_i,
375     wb_stb_i  => m0wb_stb_i,
376
377     --NoC signals
378     noc_in    => r10m0,
379     noc_out   => m0r10
380     );
381
382   m1 : noc_na_master
383     port map(
384     --Common WishBone signals
385     wb_clk_i  => wb_clk_i,
386     wb_rst_i  => wb_rst_i,
387     wb_data_i => m1wb_dat_i,
388     wb_data_o => m1wb_dat_o,
389
390     --WishBone Master
```

```
391       wb_ack_o  => m1wb_ack_o,
392       wb_adr_i  => m1wb_adr_i,
393       wb_cyc_i  => m1wb_cyc_i,
394       wb_err_o  => m1wb_err_o,
395       wb_rty_o  => m1wb_rty_o,
396       wb_sel_i  => m1wb_sel_i,
397       wb_we_i   => m1wb_we_i,
398       wb_stb_i  => m1wb_stb_i,
399
400       --NoC signals
401       noc_in    => r10m1,
402       noc_out   => m1r10
403       );
404
405
406     m2 : noc_na_master
407       port map(
408       --Common WishBone signals
409       wb_clk_i  => wb_clk_i,
410       wb_rst_i  => wb_rst_i,
411       wb_data_i => m2wb_dat_i,
412       wb_data_o => m2wb_dat_o,
413
414       --WishBone Master
415       wb_ack_o  => m2wb_ack_o,
416       wb_adr_i  => m2wb_adr_i,
417       wb_cyc_i  => m2wb_cyc_i,
418       wb_err_o  => m2wb_err_o,
419       wb_rty_o  => m2wb_rty_o,
420       wb_sel_i  => m2wb_sel_i,
421       wb_we_i   => m2wb_we_i,
422       wb_stb_i  => m2wb_stb_i,
423
424       --NoC signals
425       noc_in    => r10m2,
426       noc_out   => m2r10
```

```
427     );
428
429
430     m3 : noc_na_master
431       port map(
432       --Common WishBone signals
433       wb_clk_i  => wb_clk_i,
434       wb_rst_i  => wb_rst_i,
435       wb_data_i => m3wb_dat_i,
436       wb_data_o => m3wb_dat_o,
437
438       --WishBone Master
439       wb_ack_o  => m3wb_ack_o,
440       wb_adr_i  => m3wb_adr_i,
441       wb_cyc_i  => m3wb_cyc_i,
442       wb_err_o  => m3wb_err_o,
443       wb_rty_o  => m3wb_rty_o,
444       wb_sel_i  => m3wb_sel_i,
445       wb_we_i   => m3wb_we_i,
446       wb_stb_i  => m3wb_stb_i,
447
448       --NoC signals
449       noc_in    => r10m3,
450       noc_out   => m3r10
451       );
452
453
454     m4 : noc_na_master
455       port map(
456       --Common WishBone signals
457       wb_clk_i  => wb_clk_i,
458       wb_rst_i  => wb_rst_i,
459       wb_data_i => m4wb_dat_i,
460       wb_data_o => m4wb_dat_o,
461
462       --WishBone Master
```

```
463     wb_ack_o  => m4wb_ack_o,
464     wb_adr_i  => m4wb_adr_i,
465     wb_cyc_i  => m4wb_cyc_i,
466     wb_err_o  => m4wb_err_o,
467     wb_rty_o  => m4wb_rty_o,
468     wb_sel_i  => m4wb_sel_i,
469     wb_we_i   => m4wb_we_i,
470     wb_stb_i  => m4wb_stb_i,
471
472     --NoC signals
473     noc_in    => r11m4,
474     noc_out   => m4r11
475     );
476
477
478   m5 : noc_na_master
479     port map(
480     --Common WishBone signals
481     wb_clk_i  => wb_clk_i,
482     wb_rst_i  => wb_rst_i,
483     wb_data_i => m5wb_dat_i,
484     wb_data_o => m5wb_dat_o,
485
486     --WishBone Master
487     wb_ack_o  => m5wb_ack_o,
488     wb_adr_i  => m5wb_adr_i,
489     wb_cyc_i  => m5wb_cyc_i,
490     wb_err_o  => m5wb_err_o,
491     wb_rty_o  => m5wb_rty_o,
492     wb_sel_i  => m5wb_sel_i,
493     wb_we_i   => m5wb_we_i,
494     wb_stb_i  => m5wb_stb_i,
495
496     --NoC signals
497     noc_in    => r11m5,
498     noc_out   => m5r11
```

```
499     );
500
501
502     m6 : noc_na_master
503       port map(
504       --Common WishBone signals
505       wb_clk_i  => wb_clk_i,
506       wb_rst_i  => wb_rst_i,
507       wb_data_i => m6wb_dat_i,
508       wb_data_o => m6wb_dat_o,
509
510       --WishBone Master
511       wb_ack_o  => m6wb_ack_o,
512       wb_adr_i  => m6wb_adr_i,
513       wb_cyc_i  => m6wb_cyc_i,
514       wb_err_o  => m6wb_err_o,
515       wb_rty_o  => m6wb_rty_o,
516       wb_sel_i  => m6wb_sel_i,
517       wb_we_i   => m6wb_we_i,
518       wb_stb_i  => m6wb_stb_i,
519
520       --NoC signals
521       noc_in    => r11m6,
522       noc_out   => m6r11
523       );
524
525
526     m7 : noc_na_master
527       port map(
528       --Common WishBone signals
529       wb_clk_i  => wb_clk_i,
530       wb_rst_i  => wb_rst_i,
531       wb_data_i => m7wb_dat_i,
532       wb_data_o => m7wb_dat_o,
533
534       --WishBone Master
```

```
535       wb_ack_o  => m7wb_ack_o,
536       wb_adr_i  => m7wb_adr_i,
537       wb_cyc_i  => m7wb_cyc_i,
538       wb_err_o  => m7wb_err_o,
539       wb_rty_o  => m7wb_rty_o,
540       wb_sel_i  => m7wb_sel_i,
541       wb_we_i   => m7wb_we_i,
542       wb_stb_i  => m7wb_stb_i,
543
544       --NoC signals
545       noc_in    => r11m7,
546       noc_out   => m7r11
547       );
548
549     -- Route connections
550
551     -- First line
552
553     r10 : noc_router
554
555     port map (
556       --I/O ports
557       clk_i   => wb_clk_i,
558       rst_i   => wb_rst_i,
559
560       --NoC signals
561       --North
562       noc_n_i   => m1r10,
563       noc_n_o   => r10m1,
564
565       --South
566       noc_s_i   => m2r10,
567       noc_s_o   => r10m2,
568
569       --East
570       noc_e_i   => r20r10,
```

```
571   noc_e_o   => r10r20,
572
573   --West
574   noc_w_i   => m3r10,
575   noc_w_o   => r10m3,
576
577   --IP
578   ip_i      => m0r10,
579   ip_o      => r10m0
580         );
581
582   r11 : noc_router
583
584   port map (
585     --I/O ports
586     clk_i   => wb_clk_i,
587     rst_i   => wb_rst_i,
588
589     --NoC signals
590     --North
591     noc_n_i   => m5r11,
592     noc_n_o   => r11m5,
593
594     --South
595     noc_s_i   => m6r11,
596     noc_s_o   => r11m6,
597
598     --East
599     noc_e_i   => r20r11,
600     noc_e_o   => r11r20,
601
602     --West
603     noc_w_i   => m7r11,
604     noc_w_o   => r11m7,
605
606     --IP
```

```
607       ip_i     => m4r11,
608       ip_o     => r11m4
609           );
610
611     --Second stage
612
613     r20 : noc_router
614
615     port map (
616       --I/O ports
617       clk_i    => wb_clk_i,
618       rst_i    => wb_rst_i,
619
620       --NoC signals
621       --North
622       noc_n_i    => r10r20,
623       noc_n_o    => r20r10,
624
625       --South
626       noc_s_i    => r11r20,
627       noc_s_o    => r20r11,
628
629       --East
630       noc_e_i    => r30r20,
631       noc_e_o    => r20r30,
632
633       --West
634       noc_w_i    => nc,
635       noc_w_o    => open,
636
637       --IP
638       ip_i     => nc,
639       ip_o     => open
640           );
641
642     --Third stage
```

```
643
644   r30 : noc_router
645
646   port map (
647     --I/O ports
648     clk_i    => wb_clk_i,
649     rst_i    => wb_rst_i,
650
651     --NoC signals
652     --North
653     noc_n_i    => s0r30,
654     noc_n_o    => r30s0,
655
656     --South
657     noc_s_i    => s1r30,
658     noc_s_o    => r30s1,
659
660     --East
661     noc_e_i    => s2r30,
662     noc_e_o    => r30s2,
663
664     --West
665     noc_w_i    => s3r30,
666     noc_w_o    => r30s3,
667
668     --IP
669     ip_i       => r20r30,
670     ip_o       => r30r20
671         );
672
673   --Slave connections
674
675   s0 : noc_na_slave
676   port map(
677     --Common WishBone signals
678     wb_clk_i  => wb_clk_i,
```

```
679       wb_rst_i  => wb_rst_i,
680       wb_data_i => s0wb_dat_i,
681       wb_data_o => s0wb_dat_o,
682
683       --WishBone Master
684       wb_ack_i  => s0wb_ack_i,
685       wb_adr_o  => s0wb_adr_o,
686       wb_cyc_o  => s0wb_cyc_o,
687       wb_stb_o  => s0wb_stb_o,
688       wb_err_i  => s0wb_err_i,
689       wb_rty_i  => s0wb_rty_i,
690       wb_sel_o  => s0wb_sel_o,
691       wb_we_o   => s0wb_we_o,
692
693       --NoC signals
694       noc_in    => r30s0,
695       noc_out   => s0r30
696           );
697     s1 : noc_na_slave
698     port map(
699       --Common WishBone signals
700       wb_clk_i  => wb_clk_i,
701       wb_rst_i  => wb_rst_i,
702       wb_data_i => s1wb_dat_i,
703       wb_data_o => s1wb_dat_o,
704
705       --WishBone Master
706       wb_ack_i  => s1wb_ack_i,
707       wb_adr_o  => s1wb_adr_o,
708       wb_cyc_o  => s1wb_cyc_o,
709       wb_stb_o  => s1wb_stb_o,
710       wb_err_i  => s1wb_err_i,
711       wb_rty_i  => s1wb_rty_i,
712       wb_sel_o  => s1wb_sel_o,
713       wb_we_o   => s1wb_we_o,
714
```

```
715   --NoC signals
716   noc_in     => r30s1,
717   noc_out    => s1r30
718       );
719
720   s2 : noc_na_slave
721   port map(
722     --Common WishBone signals
723     wb_clk_i  => wb_clk_i,
724     wb_rst_i  => wb_rst_i,
725     wb_data_i => s2wb_dat_i,
726     wb_data_o => s2wb_dat_o,
727
728     --WishBone Master
729     wb_ack_i  => s2wb_ack_i,
730     wb_adr_o  => s2wb_adr_o,
731     wb_cyc_o  => s2wb_cyc_o,
732     wb_stb_o  => s2wb_stb_o,
733     wb_err_i  => s2wb_err_i,
734     wb_rty_i  => s2wb_rty_i,
735     wb_sel_o  => s2wb_sel_o,
736     wb_we_o   => s2wb_we_o,
737
738     --NoC signals
739     noc_in    => r30s2,
740     noc_out   => s2r30
741       );
742
743   s3 : noc_na_slave
744   port map(
745     --Common WishBone signals
746     wb_clk_i  => wb_clk_i,
747     wb_rst_i  => wb_rst_i,
748     wb_data_i => s3wb_dat_i,
749     wb_data_o => s3wb_dat_o,
750
```

```
751     --WishBone Master
752     wb_ack_i  => s3wb_ack_i,
753     wb_adr_o  => s3wb_adr_o,
754     wb_cyc_o  => s3wb_cyc_o,
755     wb_stb_o  => s3wb_stb_o,
756     wb_err_i  => s3wb_err_i,
757     wb_rty_i  => s3wb_rty_i,
758     wb_sel_o  => s3wb_sel_o,
759     wb_we_o   => s3wb_we_o,
760
761     --NoC signals
762     noc_in    => r30s3,
763     noc_out   => s3r30
764         );
765
766
767   end struc;
```

A.6.2 stree_table.vhd

```
1  ----------------------------------------------------------------------
2  --
3  --       Table for stree NOC
4  --       Editor: Nikolaj Tørring - s042505
5  --       Version: 1.0
6  --       Data last modified: 16/5-2007
7  --
8  ----------------------------------------------------------------------
9  --
10 --       Version History
11 --
12 --       v1.0: First working version
13 --
14 ----------------------------------------------------------------------
15
16 library IEEE;
17 use IEEE.std_logic_1164.ALL;
18 use IEEE.std_logic_arith.ALL;
19 use work.types.all;
20
21 entity noc_table is
22
23   generic(
24       --Source
25       x_s : integer := 0;
26       y_s : integer := 0
27     );
28     port(
29       address : in std_logic_vector(addr_wdth-1 downto 0);
30       route : out std_logic_vector(route_wdth-1 downto 0)
31     );
32
33 end noc_table;
34
```

```
35 architecture struc of noc_table is
36   SIGNAL fill : std_logic_vector(route_wdth-1 downto 3*2) := (OTHERS => '0');
37 begin
38
39   route_lookup : process(address)
40   begin
41     if(address(31 downto 14) = CONv_STD_LOGIC_VECTOR(0, 31-14+1)) then  --Mem1(Imem)
42       route <= dest_east & dest_east & dest_north & fill;
43     elsif(address(31 downto 14) = CONv_STD_LOGIC_VECTOR(1, 31-14+1)) then --Mem2(Dmem)
44       --make route
45       route <= dest_east & dest_east & dest_south & fill;
46     elsif(address(31 downto 24) = CONv_STD_LOGIC_VECTOR(64, 31-24+1)) then  --Semaphore
47       --make route
48       route <= dest_east & dest_east & dest_west & fill;
49     elsif(address(31 downto 24) = CONv_STD_LOGIC_VECTOR(144, 31-24+1)) then --UART
50       --make route
51       route <= dest_east & dest_east & dest_east & fill;
52     else
53       --no dest
54       route <= (OTHERS => 'Z');
55     end if;
56   end process route_lookup;
57
58 end struc;
```

Appendix B

C and assember files

B.1 Single processor test

B.1.1 reset.S

```
1  #include "board.h"
2  #include "mc.h"
```

```
3
4     .global ___main
5             .section .stack, "aw", @nobits
6   .space  STACK_SIZE
7   _stack:
8
9             .section .vectors, "ax"
10            .org 0x100
11  _reset:
12            l.movhi  r1,hi(_stack-4)
13            l.ori    r1,r1,lo(_stack-4)
14            l.addi   r2,r0,-3
15            l.and    r1,r1,r2
16
17            l.movhi  r2,hi(_main)
18            l.ori    r2,r2,lo(_main)
19            l.jr     r2
20            l.addi   r2,r0,0
21
22  ___main:
23    l.jr  r9
24    l.nop
```

B.1.2 hello.c

```
1  #include "board.h"
2  #include "uart.h"
3
4  #define BOTH_EMPTY (UART_LSR_TEMT | UART_LSR_THRE)
5
6  #define WAIT_FOR_XMITR \
7          do { \
8                  lsr = REG8(UART_BASE + UART_LSR); \
9          } while ((lsr & BOTH_EMPTY) != BOTH_EMPTY)
10
11 #define WAIT_FOR_THRE \
12         do { \
13                 lsr = REG8(UART_BASE + UART_LSR); \
14         } while ((lsr & UART_LSR_THRE) != UART_LSR_THRE)
15
16 #define CHECK_FOR_CHAR (REG8(UART_BASE + UART_LSR) & UART_LSR_DR)
17
18 #define WAIT_FOR_CHAR \
19          do { \
20                 lsr = REG8(UART_BASE + UART_LSR); \
21          } while ((lsr & UART_LSR_DR) != UART_LSR_DR)
22
23 void uart_init(void)
24 {
25   int divisor;
26
27   /* Reset receiver and transmiter */
28   REG8(UART_BASE + UART_FCR) = UART_FCR_ENABLE_FIFO | UART_FCR_CLEAR_RCVR | UART_FCR_CLEAR_XMIT |
         UART_FCR_TRIGGER_14;
29
30   /* Disable all interrupts */
31   REG8(UART_BASE + UART_IER) = 0x00;
32
33   /* Set 8 bit char, 1 stop bit, no parity */
```

```
34    REG8(UART_BASE + UART_LCR) = UART_LCR_WLEN8 & ~(UART_LCR_STOP | UART_LCR_PARITY);
35
36    /* Set baud rate */
37    divisor = IN_CLK/(16 * UART_BAUD_RATE);
38    REG8(UART_BASE + UART_LCR) |= UART_LCR_DLAB;
39    REG8(UART_BASE + UART_DLL) = divisor & 0x000000ff;
40    REG8(UART_BASE + UART_DLM) = (divisor >> 8) & 0x000000ff;
41    REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
42  }
43
44  void uart_putc(char c)
45  {
46    unsigned char lsr;
47
48    WAIT_FOR_THRE;
49    REG8(UART_BASE + UART_TX) = c;
50    if(c == '\n') {
51      WAIT_FOR_THRE;
52      REG8(UART_BASE + UART_TX) = '\r';
53    }
54    WAIT_FOR_XMITR;
55  }
56
57  char uart_getc(void)
58  {
59    unsigned char lsr;
60    char c;
61
62    WAIT_FOR_CHAR;
63    c = REG8(UART_BASE + UART_RX);
64    return c;
65  }
66
67  char *str = " Nik er guud!!\n";
68  int main (void)
69  {
```

```
70   char *s;
71
72   uart_init ();
73   for (s = str; *s; s++)
74     uart_putc (*s);
75
76   while (1)
77     uart_putc (uart_getc () + 1);
78
79   return 0;
80 }
```

B.2 Multi processor test

B.2.1 noc.h

```
1  //#ifndef _NOC_H_
2  //#define _NOC_H_
3
4  #define SEMAPHORE_BASE 0x40000000
5
6  #define SEMAPHORE_ADDRESS(i) (int*) (SEMAPHORE_BASE + 4 * i);
7
8  #define P(x) while(!(*x)) {};
9
10 #define V(x) (*x = -2);
11
12 //#define set(x, value) (*x = value);
```

B.2.2 reset.S

```
1  #include "board.h"
2  #include "mc.h"
3
4    .global ___main
5          .section .stack, "aw", @nobits
6  .space  STACK_SIZE
7  .data
8    .align 4
9    .type _offset, @object
10   .size _offset, 4
11 _offset:
12   .long   0
13
14 _stack:
15
16         .section .vectors, "ax"
17         .org 0x100
18
19   l.movhi r10, 0x4000
20   l.ori   r10, r10, 0x0000
21
22 check:
23   l.lwz   r11, 0(r10)
24   l.sfeqi r11, 0x0000
25   l.bf    check
26   l.nop
27
28
29 _reset:
30   l.movhi r12, hi(_offset)  # Get offset pointer
31   l.ori   r12, r12, lo(_offset)
32   l.lwz   r13, 0(r12)   # Get offset
33         l.movhi r1,hi(_stack-4)   # Get set stackpionter
34         l.ori   r1,r1,lo(_stack-4)
```

```
35          l.addi  r2,r0,-3
36          l.and   r1,r1,r2
37      l.add   r1,r1,r13   # Add stackpointer and offset
38      l.addi  r13, r13,STACK_SIZE # Increase offset
39      l.sw    0(r12), r13   # Save offset
40
41      l.addi  r11, r0, -1   # Get ready to release semaphore
42      l.sw  0(r10), r11   # Release semaphore
43
44      l.addi  r10, r0, 0    # clear r10
45      l.addi  r11, r0, 0    # clear r11
46      l.addi  r12, r0, 0    # clear r12
47      l.addi  r13, r0, 0    # clear r13
48
49          l.movhi r2,hi(_main)
50          l.ori   r2,r2,lo(_main)
51          l.jr    r2
52          l.addi  r2,r0,0
53
54      __main:
55        l.jr  r9
56        l.nop
```

B.2.3 4coretest.c

```
1  #include "board.h"
2  #include "uart.h"
3  #include "noc.h"
4
5  #define BOTH_EMPTY (UART_LSR_TEMT | UART_LSR_THRE)
6
7  #define WAIT_FOR_XMITR \
8          do { \
9                  lsr = REG8(UART_BASE + UART_LSR); \
10         } while ((lsr & BOTH_EMPTY) != BOTH_EMPTY)
11
12 #define WAIT_FOR_THRE \
13         do { \
14                 lsr = REG8(UART_BASE + UART_LSR); \
15         } while ((lsr & UART_LSR_THRE) != UART_LSR_THRE)
16
17 #define CHECK_FOR_CHAR (REG8(UART_BASE + UART_LSR) & UART_LSR_DR)
18
19 #define WAIT_FOR_CHAR \
20          do { \
21                 lsr = REG8(UART_BASE + UART_LSR); \
22          } while ((lsr & UART_LSR_DR) != UART_LSR_DR)
23
24 void uart_init(void)
25 {
26   int divisor;
27
28   /* Reset receiver and transmiter */
29   REG8(UART_BASE + UART_FCR) = UART_FCR_ENABLE_FIFO | UART_FCR_CLEAR_RCVR | UART_FCR_CLEAR_XMIT |
         UART_FCR_TRIGGER_14;
30
31   /* Disable all interrupts */
32   REG8(UART_BASE + UART_IER) = 0x00;
33
```

```
34    /* Set 8 bit char, 1 stop bit, no parity */
35    REG8(UART_BASE + UART_LCR) = UART_LCR_WLEN8 & ~(UART_LCR_STOP | UART_LCR_PARITY);
36
37    /* Set baud rate */
38    divisor = IN_CLK/(16 * UART_BAUD_RATE);
39    REG8(UART_BASE + UART_LCR) |= UART_LCR_DLAB;
40    REG8(UART_BASE + UART_DLL) = divisor & 0x000000ff;
41    REG8(UART_BASE + UART_DLM) = (divisor >> 8) & 0x000000ff;
42    REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
43  }
44
45  void uart_putc(char c)
46  {
47    unsigned char lsr;
48
49    WAIT_FOR_THRE;
50    REG8(UART_BASE + UART_TX) = c;
51    if(c == '\n') {
52      WAIT_FOR_THRE;
53      REG8(UART_BASE + UART_TX) = '\r';
54    }
55    WAIT_FOR_XMITR;
56  }
57
58  char uart_getc(void)
59  {
60    unsigned char lsr;
61    char c;
62
63    WAIT_FOR_CHAR;
64    c = REG8(UART_BASE + UART_RX);
65    return c;
66  }
67
68  volatile int nextJob = 0;
69  volatile int doneCOUNT = 0;
```

```
70
71  volatile int *semJob;
72  volatile int *semUART;
73
74
75  void print0();
76  void print1();
77  void print2();
78  void print3();
79
80  int main (void)
81  {
82    semJob = SEMAPHORE_ADDRESS(1);
83    semUART = SEMAPHORE_ADDRESS(2);
84    semCOUNT = SEMAPHORE_ADDRESS(3);
85
86    P(semJob);
87    switch(nextJob++) {
88      case 0:
89        uartinit();
90        V(semJob);
91        print0();
92        break;
93      case 1:
94        V(semJob);
95        print1();
96        break;
97      case 2:
98        V(semJob);
99        print2();
100       break;
101     case 3:
102       V(semJob);
103       print3();
104       break;
105     default:
```

```
106         V(semJob);
107         break;
108       }
109       return 0;
110     }
111
112     void print0() {
113       P(semUART);
114       uart_putc('H');
115       uart_putc('e');
116       uart_putc('l');
117       uart_putc('l');
118       uart_putc('o');
119       uart_putc('0');
120       uart_putc('\n');
121       V(semUART);
122       while(0) {doneCOUNT++};
123     }
124     void print1() {
125       P(semUART);
126       uart_putc('H');
127       uart_putc('e');
128       uart_putc('l');
129       uart_putc('l');
130       uart_putc('o');
131       uart_putc('1');
132       uart_putc('\n');
133       V(semUART);
134       while(0) {doneCOUNT++};
135     }
136     void print2() {
137       P(semUART);
138       uart_putc('H');
139       uart_putc('e');
140       uart_putc('l');
141       uart_putc('l');
```

```
142   uart_putc('o');
143   uart_putc('2');
144   uart_putc('\n');
145   V(semUART);
146   while(0) {doneCOUNT++};
147 }
148 void print3() {
149   P(semUART);
150   uart_putc('H');
151   uart_putc('e');
152   uart_putc('l');
153   uart_putc('l');
154   uart_putc('o');
155   uart_putc('3');
156   uart_putc('\n');
157   V(semUART);
158   while(0) {doneCOUNT++};
159 }
```

B.2.4 search.c

```
1  #include "board.h"
2  #include "uart.h"
3  #include <string.h>
4
5
6
7  const char *str = "Origins: Short stories date back to the oral story-telling traditions which
       originally produced epics such as the Iliad and Odyssey by Homer. Oral narratives were often told
       in the form of rhyming or rhythmic poetry, often including recurring sections or, in the case of
       Homer, Homeric epithets. Such stylistic effects often acted as mnemonic means for easier recall,
       rendition and adaptation of the story. Short sections of such poems might focus on individual
       narratives that could be told at one sitting. The overall arc of the story would only emerge
       through the telling of multiple sections of the tale. Fables, which tend to be folk tales with an
       explicitly expressed moral, were said by the Greek historian Herodotus to have been invented by a
       Greek slave named Aesop in the 6th century BCE (although other times and nationalities are also
       given for Aesop). These ancient fables are known today as Aesop's Fables. The other ancient form of
        short story, anecdotes, were popular during the years of the Roman Empire. Anecdotes functioned as
        a sort of parable, a brief realistic narration that embodies a point. Many of the surviving Roman
       anecdotes were later collected in the Gesta Romanorum in the 13th or 14th century. Anecdotes
       remained popular in Europe well into the 18th century, when the fictional anecdotal letters of Sir
       Roger de Coverley were published. In Europe, the oral story-telling tradition began to develop into
        written stories in the early 14th century, most notably with Geoffrey Chaucer's Canterbury Tales
       and Giovanni Boccaccio's Decameron. Both of these books are composed of individual short stories (
       which range from farce or humorous anecdotes to well-crafted literary fictions) set within a larger
        narrative story (a frame story), although the frame tale device was not adopted by all writers. At
        the end of the 16th century, some of the most popular short stories in Europe were the darkly
       tragic novella of Matteo Bandello (especially in their French translation). During the Renaissance,
        the term novella was used when referring to short stories. The mid 17th century in France saw the
       development of a refined short novel, the nouvelle, by such authors as Madame de Lafayette. In the
       1690s, traditional fairy tales began to be published (one of the most famous collections was by
       Charles Perrault). The appearance of Antoine Galland's first modern translation of the Thousand and
        One Nights (or Arabian Nights) (from 1704; another translation appeared in 1710-12) would have an
       enormous influence on the 18th century European short stories of Voltaire, Diderot and others. \n
       Modern short stories: Modern short stories emerged as their own genre in the early 19th century.
```

```
      Early examples of short story collections include the Brothers Grimm Fairy Tales (1824-1826),
      Nikolai Gogol's Evenings on a Farm Near Dikanka (1831-1832), Edgar Allan Poe's Tales of the
      Grotesque and Arabesque (1839) and Nathaniel Hawthorne's Twice Told Tales (1842).";
8
9   const char *word = "the";     //the = 44
10
11  const int Jobs = 4;
12
13  volatile int nextJob = 0;
14
15  volatile int count = 0;
16  volatile int jobsdone = 0;
17
18  int length = 0;
19
20  volatile int *semJob;
21  volatile int *semUART;
22  volatile int *semCOUNT;
23
24  int main (void)
25  {
26
27    int top, bottom, partcount;
28  //  char *s;
29
30    semJob = SEMAPHORE_ADDRESS(1);
31    semUART = SEMAPHORE_ADDRESS(2);
32    semCOUNT = SEMAPHORE_ADDRESS(3);
33
34    //Determin length of string
35    length = strlen(*str);
36
37    while(nextJob < Jobs) {
38      //Determin which part of the string to search
39      P(semJob);
40      bottom = length/Jobs*(nextJob++);
```

```
41  top = length/Jobs*(nextJob)-1;
42  if(nextJob == 1) { //first shall initialize the UART
43    V(semJob);
44    uart_init ();
45  }
46  else {
47    V(semJob);
48  }
49
50  //change bottom of range to fit whole word
51  while(bottom != 0 && *str[bottom] != " ") {
52    bottom--;
53  }
54
55  //change top of range to fit whole word
56  while(top != 1000 && *str[top] != " ") {
57    top++;
58  }
59
60  //Search for string
61  partcount = seach(*str, *word, bottom, top);
62
63  P(semUART);
64    uart_putc ('P');
65    uart_putc ('a');
66    uart_putc ('r');
67    uart_putc ('t');
68    uart_putc (' ');
69    uart_putc ('d');
70    uart_putc ('o');
71    uart_putc ('n');
72    uart_putc ('e');
73    uart_putc ('\n');
74  V(semUART);
75
76  P(semCOUNT);
```

```
77        count = count + partcount;
78        if(++jobsdone == Jobs) {
79          char *donetext = "Search is finish \n \"" + *word + "\" appears " + count + " times.";
80          char *s;
81          for (s = donetext; *s; s++) {
82              uart_putc (*s);
83          }
84        } ;
85
86      V(semCOUNT);
87
88    }
89
90    return 0;
91  }
92
93  //Search funktion using Knuth Morris Pratt algorithm
94  int search(char *text, char *word, int start, int end) {
95
96
97    int wordsize = strlen(*word); //Size of word
98    int count = 0;       //No of times word appers in text
99    int m = start;       //the beginning of the current match in text
100   int i = 2;       //in the table: the current position we are computing in T, in the search:the
          position of the current character in word
101   int j = 0;       // Index in word of the next character of the current candidate substring
102   int T[wordsize];     //Jump table
103
104   //Make table
105   T[0] = -1;
106   T[1] = 0;
107
108   while(i < wordsize) {
109     if(*word[i-1] == *word[j]) {
110       T[i] = ++j;
111       i++;
```

```
112
113
114         }
115         else if(j > 0 {
116           j = T[j];
117         }
118         else {
119           T[i] = 0;
120           i++;
121         }
122       }
123
124       //Do the search
125       i = 0;
126       while(m+i< end) {
127         if(*word[i] == *text[i]) {
128           i++;
129           if(i == wordsize) {
130             count++;
131             m = m+i+1;
132             i=0;
133           }
134         }
135         else {
136           m = m+i-T[i];
137           if(i > 0) {
138             i = T[i];
139           }
140         }
141       }
142       return count;
143     }
144
145     #define BOTH_EMPTY (UART_LSR_TEMT | UART_LSR_THRE)
146
147     #define WAIT_FOR_XMITR \
```

```
148         do { \
149                 lsr = REG8(UART_BASE + UART_LSR); \
150         } while ((lsr & BOTH_EMPTY) != BOTH_EMPTY)
151
152 #define WAIT_FOR_THRE \
153         do { \
154                 lsr = REG8(UART_BASE + UART_LSR); \
155         } while ((lsr & UART_LSR_THRE) != UART_LSR_THRE)
156
157 #define CHECK_FOR_CHAR (REG8(UART_BASE + UART_LSR) & UART_LSR_DR)
158
159 #define WAIT_FOR_CHAR \
160          do { \
161                 lsr = REG8(UART_BASE + UART_LSR); \
162          } while ((lsr & UART_LSR_DR) != UART_LSR_DR)
163
164 void uart_init(void)
165 {
166   int divisor;
167
168   /* Reset receiver and transmiter */
169   REG8(UART_BASE + UART_FCR) = UART_FCR_ENABLE_FIFO | UART_FCR_CLEAR_RCVR | UART_FCR_CLEAR_XMIT |
        UART_FCR_TRIGGER_14;
170
171   /* Disable all interrupts */
172   REG8(UART_BASE + UART_IER) = 0x00;
173
174   /* Set 8 bit char, 1 stop bit, no parity */
175   REG8(UART_BASE + UART_LCR) = UART_LCR_WLEN8 & ~(UART_LCR_STOP | UART_LCR_PARITY);
176
177   /* Set baud rate */
178   divisor = IN_CLK/(16 * UART_BAUD_RATE);
179   REG8(UART_BASE + UART_LCR) |= UART_LCR_DLAB;
180   REG8(UART_BASE + UART_DLL) = divisor & 0x000000ff;
181   REG8(UART_BASE + UART_DLM) = (divisor >> 8) & 0x000000ff;
182   REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
```

```
183 }
184
185 void uart_putc(char c)
186 {
187   unsigned char lsr;
188
189   WAIT_FOR_THRE;
190   REG8(UART_BASE + UART_TX) = c;
191   if(c == '\n') {
192     WAIT_FOR_THRE;
193     REG8(UART_BASE + UART_TX) = '\r';
194   }
195   WAIT_FOR_XMITR;
196 }
197
198 char uart_getc(void)
199 {
200   unsigned char lsr;
201   char c;
202
203   WAIT_FOR_CHAR;
204   c = REG8(UART_BASE + UART_RX);
205   return c;
206 }
```

Appendix C

Tables

packet /cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	m0	r10	r20	r30	r40	r50	s0	x	x	x	x	x	x	x
1	m1	r10	r10	r20	r20	r30	r30	r30	r40	r50	s0	x	x	x
2	m2	r11	r20	r20	r30	r30	r40	r50	s0	x	x	x	x	x
3	m3	r11	r11	r20	r20	r20	r30	r30	r30	r30	r40	r50	s0	x
4	m4	r12	r21	r30	r30	r40	r50	s0	x	x	x	x	x	x
5	m5	r12	r12	r21	r21	r30	r30	r30	r30	r40	r50	s0	x	x
6	m6	r13	r21	r21	r30	r30	r30	r40	r50	s0	x	x	x	x
7	m7	r13	r13	r21	r21	r21	r30	r30	r30	r30	r30	r40	r50	s0

Table C.1: The flow of packets when all master connections send out a request at the same time. This table shows the for a `tree`-network

packet /cycle	1	2	3	4	5	6	7	8	9	10	11	12
0	m0	r10	r20	r30	s0							
1	m1	r10	r10	r20	r20	r30	s0					
2	m2	r10	r10	r10	r20	r20	r20	r30	s0			
3	m3	r10	r10	r10	r10	r20	r20	r20	r20	r30	s0	
4	m4	r11	r20	r20	r30	s0						
5	m5	r11	r11	r20	r20	r20	r30	s0				
6	m6	r11	r11	r11	r20	r20	r20	r20	r30	s0		
7	m7	r11	r11	r11	r11	r20	r20	r20	r20	r20	r30	s0

Table C.2: The flow of packets when all master connections send out a request at the same time. This table shows the for a `tree`-network

Bibliography

[1] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1):51, March 2006.

[2] Damjan Lampret. *OpenRISC 1200 IP Core Specification*, 0.7 edition, September 2001.

[3] opencores.org. *UART IP Core Specification*, b.3 edition, August 2002.

[4] opencores.org. *WISHBONE System-on-Chip(SoC) Interconnection Architecture for Portable IP Cores*, b.3 edition, September 2002.

[5] opencores.org. *OpenRISC 1000 Architectural Manual*, April 2006.

[6] opencores.org. Opencores.org. http://www.opencores.org/, June 2007.

[7] Vijay Pande and Stanford University. Folding. http://folding.stanford.edu/.

[8] Arteris (www.arteris.com). A comparison of network-on-chip and busses. *Design & Reuse*, 2005.

[9] Inc Xilinx. Xilinx. http://www.xilinx.com/.